Real-Time Preference Analytics on Data Streams

Referees:
Prof. Dr. Markus Endres
Prof. Dr. Werner Kießling

Day of Defense:
December 14, 2020

Real-Time Preference Analytics on Data Streams

Lena Rudenko
University of Augsburg

Contents

Abstract

In today's world, data and the information contained therein is one of the most important resources and, at the same time, one of the driving forces behind human development. Not all information and data are equally relevant or important. A huge array of numbers are data too, but without context they are completely meaningless. However, even if these numbers are assigned a certain meaning, it does not automatically make this information useful or interesting for every individual. Thus, *personalized* information, carefully selected according to the *interests* and *preferences* of the individual, is of particular value.

The ways in which data and information are transmitted and stored have changed with the development of humanity, from man-to-man legends to social networks where absolutely everyone can create content. Hundreds of millions of Twitter users produce data every second in a *continuous stream*: they write their own posts, comment, "like" or retweet the posts of the other users, etc. We have an ever-increasing amount of data, which is extremely difficult to navigate.

New forms of data require new approaches of their processing and analysis. Stream data processing has received considerable attention in the last decade. The modern applications require the analysis and study of continuous, unstructured and time-varying data instead of static records. One of the most important topics in the field of data processing and analysis is searching for *interesting* and *relevant* information for a *particular user*. The goal of my doctoral thesis is a preference-based analysis of stream data using an efficient algorithm and final aggregation of the resulting tweets.

In this work I present a framework that allows the user to reduce the stream of data to only highly relevant information using preferences applied to a wide range of various tweets' attributes. The resulting set of text messages is additionally revised before output, so that the user receives a compact summary and does not have to read a large number of individual tweets.

Chapter 1

Introduction

Information is a very important resource in modern society. The information a user receives is limited by the media to which he has access. Information is not objective, either. Radio, TV and newspapers decide about *what*, *in what context* and *with what connotation* they inform their audience. Let us imagine a photo, that depicts a glass. This glass is half filled with a liquid. Looking at this picture, everyone can decide for themselves, whether they think the glass is *half full* or it is *half empty*. One can also take note of the picture without drawing conclusions because any further information is missing. Information as it is presented by media often veils the facts with connotation, which makes it hard to interpret and value a fact individually.

The other problem with the presentation of information in the newspapers or on TV is the limited space and broadcasting time they have. This means that *not everything* that happens is reported. The audience only receives what is known to journalists and what they or their editors considered important. If World Cup event and a local youth competition will take place at the same time, it is more likely that the World Cup makes it into headline.

The classical media want to attract more readers/spectators because that also means more money for them. If more customers are interested in football than in boxing, the media will report more about football. If figure skating is a marginal sport with a small audience, no one will create a separate TV-channel to exclusively broadcast figure skating. Skaters will receive attention at best if they win a medal of a world championship or Olympic Games. Such a media policy makes perfect sense, but it also means that a group of people with less common interests or someone who wants to get an unfiltered information are neglected.

With the beginning and development of the Internet, the problem of limited space

in printed media has become irrelevant. And with the spread of social media and internet-enabled mobile devices the problem of coverage of highly specialized topics in the media was also solved. Anyone can maintain a fan page on Facebook or create a Twitter or an Instagram account to post news about their favourites. Even live broadcasts are now possible for individuals with a smartphone and Internet access. Other people interested in the current topic users can read the posts, add their own comments or express their opinions - something that was nearly impossible in traditional media.

However, along with the *opportunities* come new *challenges*. Nowadays, incredibly large amounts of data are produced, both by technology (e.g., sensor measurements) and by users (e.g., messages in Twitter). In 2019 there were 3.2 billion[1] smartphone users worldwide. Even if we take into account that not everyone who has a smartphone is actively using social networks, the amount of content produced every day is enormous. In Twitter alone, 9.1 thousand messages are sent every second[2] and users have often accounts in two or three social networks. It is very difficult to navigate through the information flow, which, moreover, is constantly increasing.

Processing big data and searching for interesting and relevant information in large data sets is a highly important topic in both academia and business. The challenges faced by researchers have changed with changes in every day life.

Most data nowadays are not in the form of *persistent datasets* anymore, but rather come in continuous, rapid and often unstructured and time-varying *data streams*. Thus, modern applications require the study and analysis of time evolving data instead of static records. Examples for stream-based applications include sensor networks, infrastructure and traffic monitoring, electronic trading on stock markets and of course social media. It is not easy to find *relevant content* in a large amount of *evolving* and *unbounded* data. Furthermore, users want to get only information they are interested in. That means, they want to get *personalized results*, which are filtered out from the ever-growing amount of data according to the *user's preferences*.

The aim of my research work is to process and analyze a stream of text messages (tweets) to deliver information according to personal interests (preferences). If exact matches are not available, best alternatives are proposed. In that context, preferences allow to reduce the huge amount of data to highly relevant information and never produce an empty result. However, due to the nature of streams, even the reduced information could be too large on the one hand and have a lot of duplicates and incomplete information snippets on the other hand. Thus, it is not enough to provide personalized information to the user, but it is also important to present the result in a way that it can be easily consumed. Therefore the result needs some post-processing

[1] Smartphone users per year: https://bit.ly/350I0fW

[2] Tweets per second: www.internetlivestats.com/twitter-statistics/

to make the information easily understandable.

The nature of data streams requires a rethinking of preferences in static contexts. Due to the continuous and potentially unlimited character of stream data, it needs to be processed sequentially and incrementally. However, queries on streams run continuously over a period of time and return updated results as new data arrive. There exist many approaches for effective processing of data streams but learning from evolving and unbounded data w.r.t. user's preferences requires new algorithms and methods to accomplish this task effectively.

The data produced by users in social media is different than the data coming from sensor measurements or traffic monitoring. It is extremely diverse in its form and maintenance: likes, posts (as text, photos or video), comments - all this data of diverse type and even more diverse content is distributed more or less unstructured all over the internet. At the same time, all of this data is mostly persistent - but with its dynamical (unstructured) and growing context, its meaning and/or relevance changes over time. For example, information about the first COVID-19 infection is still available, but its meaning to the public has changed from a short notice "new corona virus type found", which was interesting for virologists only, to a date that marks the outbreak of a pandemic by which everyone is affected. At the same time, the vast amount of COVID-19 related information that was added since then now makes it nearly impossible to search for and eventually retrieve this first news posting.

Another feature of social media that needs to be taken into account is the fact that you only see the posts of the people you are following, the people you are friends with, or, if you know the exact user account's name, whose posts you want to see. But you do not have a chance to get some highly relevant information if it comes from an account you do not know about.

Let us think about the following example: you are a great fan of figure skating. As in any other sports, the skaters have large number of competitions at various levels for athletes of all ages - from children to professionals. It is exciting for the true fans of this sport to observe the growth and development of the athletes over the years. However, since figure skating is a niche sport (one that is not popular enough to be of interest for TV channels due to advertising revenues), until recently there was no possibility to watch performances of athletes outside the World Championships or Olympic Games.

In the times before the Internet, people have not even got the text summary about the results of smaller competitions. Nowadays, mostly thanks to social networks such as Facebook, Instagram or Twitter, one gets the posts of fans eyewitness all sorts of competitions from the most distant regions. These contributions are available to all almost live or near real-time and can contain not only the results and emotions but can also be enriched with photos or short videos.

According to their functionality, Facebook and Instagram are better suited for writing long and detailed posts. Twitter, on the other hand, can be used as a medium for live text broadcasting. The short text messages that are published promptly are a very good information source for those who cannot be in person on-site, but have a great interest in a live event.

In Oberstdorf (Germany) every year a figure skating competition Nebelhorn Trophy takes place. There is no television broadcasting and in the year before the Olympic Games, the last licenses for the athletes are issued there. This is something the fans want to know about as soon as possible. The easiest way to get this information is to find tweets from eyewitnesses. But it can be very challenging. In Twitter, just like on Facebook or Instagram, one can follow other users. It makes sense if the person one is following constantly tweets about the topic one is interested in. But in our example with marginal sports some relevant information can be posted by accounts, which had not been in one's area of interest before. It can be casual spectators (their posts are less informative, but they can still be interesting) or the true fans, which understand technical details and have rich background knowledge.

In my thesis I developed a preference-based framework for the analysis of stream data, which allows users to successfully avoid the described difficulties and get the interesting content even if it is posted by the Twitter accounts they are not following. The user should specify his search parameter using preferences, which allow to filter the incoming stream of text messages from Twitter and to deliver personalized results for each user. In the result set are tweets from all accounts that match his search criteria as good as possible. Both the content-related attributes, such as *hashtags* and the user- or account-related ones, as, e.g., the account's *creation date*, the *number of friends* or *followers*, the *language of tweets* can be used in the query. However, to allow the user to specify queries directly on the *tweet* itself, I developed and implemented a *new CONTAINS preference*, which allows the preferred search in text messages. The user can use this CONTAINS preference to list the terms that these texts (tweets) should contain. Once the user has formulated his query, he wants result as soon as possible. Therefore, as part of this work, in order to process user requests efficiently, I developed the *Stream Lattice Skyline algorithm*, which delivers results in real time. Finally, the calculated result has to be presented to the user as conveniently as possible. Hardly anyone enjoys skimming through several tweets in a result set in order to get the most information. And since the number of tweets in result set can be quite large, I *aggregate* them. The aggregated message, which on the one hand does not contain duplicates and on the other hand keeps all relevant information together, is a good way to present calculated result to the user.

In summary, my approach allows the user to search efficiently for relevant topics in a stream of short text messages, and to receive the results in a convenient format.

The contribution of this work is a preference-based stream processing framework. After the introduction, I provide the background information in Chapter 2: I introduce the theory behind the preferences in Section 2.1 and give an overview about data streams and their features, especially with regard to the evaluation of the stream data using preferences in Section 2.2. The short description of Apache Flink as a platform for computations over unbounded and bounded data streams follows in Section 2.3. To conclude Chapter 2 I discuss the micro-blogging service Twitter and tweets from the programmer's point of view and as the source of stream data for my framework in Section 2.4. Chapter 3 deals with the general idea and conception of my framework. In Section 3.1 I first describe how to get tweets as a data stream. The following Section 3.2 describes the ETL process and explains how the incoming stream data have to be transformed for processing with Preference SQL. Section 3.3 addresses the challenges that arise during this processing and evaluation. Finally, in Section 3.4 I describe the concept for presenting the results filtered with Preference SQL in a convenient form. The following Chapter 4 is dedicated to a newly developed and implemented CONTAINS preference. In contrast to the already available preferences, CONTAINS, which is described in Sections 4.1 and 4.2, is able to evaluate tweets (plain texts) taking into account possible misspelled words. The experiments performed with this preference can be found in Section 4.3. Chapter 5 describes one of the most important parts of my work: an algorithm for efficient pareto computation on the streams called Stream Lattice Skyline. I start with the Block-Nested-Loop algorithm version adapted for stream evaluation in Section 5.1, which is doing its job, but shows rather bad runtime. Section 5.2 describes the new developed SLS algorithm with much better runtime that is presented in Section 5.3 with a lot of different experiments. Chapter 6 deals with aggregation and summarization of filtered tweets. A general concept idea from Section 6.1 is discussed in detail in Section 6.2, taking into account all insights gained during implementation. Three basic steps of this approach are described thorough and illustrated with examples. The last Section 6.3 of this chapter summarises the findings of the experiments. Results of the performance tests as well as quality analysis based on user surveys are presented. Chapter 8 concludes this doctoral thesis including a list of issues that deserve further research in future.

Chapter 2

Background

This chapter covers the most important basics from the areas that are relevant to the topic of the current doctoral thesis.

I start with the concept of *user preferences* presented in Section 2.1. It explains the idea of soft constraints and their advantages in personalized information search over hard constraints. The following Section 2.2 deals with the *data streams*. Their most important characteristics and differences to the persistent data sets are explained in this section. The challenges of processing and analyzing of data streams are also part of this section. Section 2.3 focuses on an *Apache Flink* stream-processing framework, which is used in the current work primarily as a stream data provider. Finally, Section 2.4 focuses on *Twitter* as a micro-blogging and social networking platform. Tweets - the short text messages that users send via Twitter - are the focus of my work. Besides the text message content, there is a lot of additional information that you can get from a tweet. What kind of information this is, where to find it and how to make it useful when doing a preference-based search in a stream of tweets is also discussed in this chapter.

2.1 Preferences - Theory

Preferences are ubiquitous in our daily live. We deal with preferences when we book a holiday, go to the cinema, choose the school for our children or buy a new car, go shopping and so on. We use the expression *"I prefer this over that"* without being aware that it is a preference-based wish. In rare cases we want something very specific and are not willing to consider the *alternatives*. In most cases the second or third choice is also fine, if the first choice is not available.

It was only a question of time that the preferences would be used in the field of information search. The search based on hard conditions returns an empty result if the perfect matches do not exist among the data. But the suitable alternatives to the exact user preference are better than no result at all.

The following subchapter deals with preference theory, explains the most important concepts and terms I use in this work and illustrats them with appropriate examples.

2.1.1 Preferences - Basics

For some time now, scientists have been developing various models designed to reflect the complex and often contradictory wishes of users. Some of these models can be found, e.g., in [Arr59, KR93, GL94, AW00, HKP01] or [Cho02]. These works have in common the treatment of the differentiated user desires, which prefer some results to the others.

I use an approach of [Kie02, Kie05], which is easy to handle and models preferences as strict partial order with intuitive interpretation *"I like A more than B"*. This approach is semantically rich, flexible and allows to model user preferences fairly accurately.

The project *"It's a Preference World"* was one of the most important research topics of the Chair for Databases and Information Systems under direction of Professor Werner Kießling at the University of Augsburg (Germany).

Preferences in the real world are expressed with the term *"better than"*, which is intuitively understood by everyone and allows to map them onto strict partial orders.

Definition 1 (Preference)
A preference is defined as $P = (A, <_P)$, where $A = \{A_1, A_2, ..., A_n\}$ is a set of n A_i attributes with domains $dom(A_i)$, $1 <= i <= n$ and $<_P \subseteq dom(A) \times dom(A)$ is a strict partial order (irreflexive and transitive).

The term "$x <_P y$" can be interpreted as *"y is preferred over x"*, thus some values are considered to be better than the other ones. If two values cannot be ordered by

the strict partial order $<_P$, they are called *indifferent*

Definition 2 (Indifference)
If non of two values x, $y \in dom(A)$ with $x \neq y$ is better than the other, they are *indifferent:* $x \sim_P y \Leftrightarrow \neg(x <_P y) \wedge \neg(y <_P x)$

Definition 3 (Substitutable Values)
Given $P = (A, <_P)$, then $\cong_P$ is called *substitutable values relation* iff $\forall x, y, z \in dom(A)$:

a) $x \cong_P y \Rightarrow x \sim_P y$

b) $x \cong_P y \wedge \exists z : z <_P x \Rightarrow z <_P y$

c) $x \cong_P y \wedge \exists x : z <_P y \Rightarrow y <_P z$

d) $\cong$ is reflexive, symmetric and transitive

In this thesis I consider *indifferent* domain values ($x \sim_P y$) as *substitutable* (regular SV-semantics, see [Kie05]).

Preferences should be considered as *soft constraints*: Not only a *perfect match* will be specified, but also the *possible alternatives* in case the perfect one does not exist. In this way, the probability of getting an *empty result* in response to the query is excluded. And this is a fundamental difference between the *preference-based* and the *hard condition* approach, where only the perfect hits are accepted as answer, what often leads to an empty result set.

Preferences refer to categorical as well as numerical data: If the user prefers some language or a certain tweet author it is about *categorical* preferences. But if he is looking for the user with the most followers or wants to restrict the number of posted tweets, there is an another kind of preferences - the *numerical* one. A wide range of various *preference constructors* was defined by [Kie02, Kie05, KEW11, WEMK12]. They are divided into *base preferences* for data on single attributes and *complex preferences*, which combine the other preferences (base or complex) to express more complex queries.

Figure 2.1 shows the "ISA-hierarchy" for most frequently base preferences.

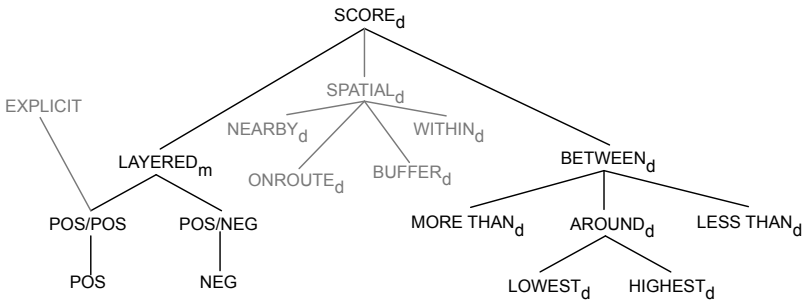


Figure 2.1: Taxonomy of base preference constructors.

Definition 4 (Weak Order Preference (WOP))
A preference $P = (A, <_P)$ is a *weak order preference* (WOP), if negative transitivity holds, i.e., $\forall x, y, z \in dom(A)$:

$$\neg(x <_P y) \wedge \neg(y <_P z) \Rightarrow \neg(x <_P z)$$

All preferences in Figure 2.1 are WOPs, cp. [Kie05]. For this kind of preferences domination and indifference between domain values can be determined by the *level value* computed by a $level_P$ function, which depends on the preference P [PK07, Pre09].

Definition 5 (Level Function)
For a preference P, every domain value can be mapped to a *level value* by a *level function* called $level_P$.

$$level_P : dom(A) \rightarrow \mathbb{N}_0$$

Better domain values have lower level values. The *maximum level value* for P is $max(P)$, the *minimum level value* is 0. For WOPs with *regular SV-semantics* two domain values x and y having the same level value are *equivalent*. And all domain values mapping to the same level are *substitutable*, cp. [Kie05].

Each constructor in the taxonomy in Figure 2.1 is a sub-constructor of $\texttt{SCORE}_d$. Let us discuss the *base preference* constructors in detail.

Definition 6 (SCORE$_d$ Preference)
The SCORE$_d$ preference is the "super-preference". It produces a numerical ranking and therefore allows to describe the other base preference constructors. Better objects are mapped to smaller numerical values and vice versa (cp. [Pre09]).

There is a numerical utility function $f_d : dom(A) \rightarrow \mathbb{R}$, $d > 0$. P is a SCORE$_d$ preference, iff for $x, y \in dom(A) : x <_P y \Leftrightarrow f_d(x) > f_d(y)$ where $f_d(v), v \in dom(A)$, is defined as $f_d(v) = \lceil f(v)/d \rceil$.

The d-value is called the $d-parameter$ and allows the partitioning of the range of f, when dealing with numerical values. Different function values can be mapped to the same number and therefore become substitutable with values in the same *equivalence class*. The d-parameter discretizes the preference domain it is defined on.

Numerical Base Preferences

BETWEEN$_d$, AROUND$_d$, LOWEST$_d$, HIGHEST$_d$, LESS THAN$_d$ and MORE THAN$_d$ base preferences are used on attributes with numerical domains and all of them can be described by SCORE$_d$ (cp. Figure 2.1). Only a score function f suitable to preference constructor is needed.

Definition 7 (BETWEEN$_d$ Preference)
BETWEEN$_d$ is a direct child of SCORE$_d$ and describes the wish for a value between a lower and an upper bound inclusive.

Given the low and up bounds $\in dom(A)$, BETWEEN$_d(A, [low, up])$ for $low \leq up$ and $d > 0$ is defined as the following scoring function:

$$f(v) := \begin{cases} low - v, & \text{if } v < low \\ 0, & \text{if } low \leq v \leq up \\ v - up, & \text{if } up < v \end{cases} \tag{2.1}$$

Let us consider a small example, which shows the usage of BETWEEN$_d$ preference and its benefit comparing to the standard hard constraint BETWEEN-function in SQL.

Example 1 Each tweet contains a lot of attributes that can be used to filter tweets w.r.t. user preferences and wishes (detailed description of the tweet attributes will be provided later in this work). One of them is called `statuses_count` and is responsible for the number of tweets (statuses) the user had already posted in his account. Let us assume, the user is looking for some news about local figure skating competition on Twitter. He restricts the `statuses_count` values to the interval between 50 and 1000 tweets. He believes that only someone who is already familiar with Twitter will

post something about this small competition. In that case the user has posted some tweets in the past and the value of lower bound is set to 50. On the other hand, it is not to be expected that some average user posts a lot, so the upper bound is not set to high. The user set also the value of the d-parameter, because the deviation of 10 tweets does not matter. This preference can be expressed by:

$$\mathrm{P} := \texttt{BETWEEN}_{d=10}(\textit{statuses\_count}, [50, 1000])$$

The visualization of this preference P in Figure 2.2 shows us that each tweet, depending on the total number of messages its author posted, belongs to some level. The best tweets regarding to the preference P are the lowest level tweets, they belong also to the result set. Ideally, these are the tweets from level 0, but theoretically they can be from any level.

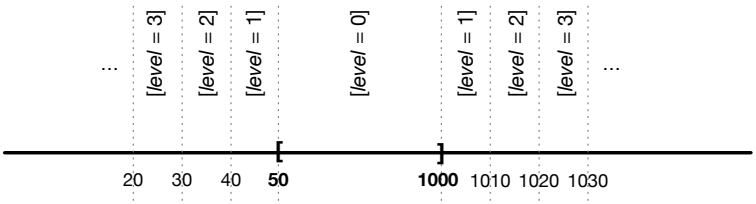


Figure 2.2: $\texttt{BETWEEN}_d$ preference and level values.

With a small change in $\texttt{BETWEEN}_d$(A, [low, up]) we get the $\texttt{AROUND}_d$(A, z) preference constructor.

Definition 8 ($\texttt{AROUND}_d$ Preference)
$\texttt{AROUND}_d$ preference allows to look for the desired value z. So if we set $low = up$ in $\texttt{BETWEEN}_d$ constructor, we get the $\texttt{AROUND}_d$(A, z) one:

$$\texttt{AROUND}_d(\mathrm{A}, \mathrm{z}) := \texttt{BETWEEN}_d(\mathrm{A}, [\mathrm{z}, \mathrm{z}])$$

$\texttt{AROUND}_d$ can also be expressed using the $\texttt{SCORE}_d$ preference with the following scoring function:

$$f(v) := |z - v| \tag{2.2}$$

If the preferred value z is not available, values within a shortest distance of d are acceptable.

Example 2 Another very useful tweet attribute is `followers_count` - the number of users that follow the current account. The more followers an account has, the more prominent it is. E.g., the user is interested in some local news so he is looking for some local politicians and journalists accounts. He believes that the followers number is a good filter criterion. About 300000 followers is too many for a "normal" account but too few for someone on the state level. The following `AROUND` preference describes the user wish highly accurate:

$$\mathrm{P} := \texttt{AROUND}_{d=1000}(\mathit{followers\_count}, 300000)$$

Visualization of the `AROUND` preference with a d-parameter is shown in Figure 2.3. If the tweets with the exact desired followers number are not found, the best alternatives will be provided.

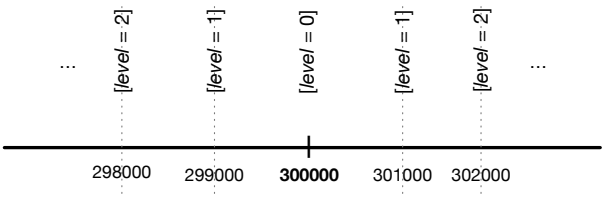


Figure 2.3: $\texttt{AROUND}_d$ preference and level values.

Definition 9 (Extremal Preferences: $\texttt{HIGHEST}_d$ and $\texttt{LOWEST}_d$)
With the help of these preferences the user searches for the values as high or as small as possible. $\texttt{HIGHEST}_d$(A) and $\texttt{LOWEST}_d$(A) can be derived from $\texttt{AROUND}_d$(A, z) in return. In this way we need to set $z = \inf_A$ or $z = \sup_A$ ($\inf_A$ and $\sup_A$ are infimum/supremum of dom(A)) to describe $\texttt{HIGHEST}_d$(A) and $\texttt{LOWEST}_d$(A) preference constructors:

a) $\texttt{HIGHEST}_d := \texttt{AROUND}_d(\mathrm{A}, \sup_A)$

b) $\texttt{LOWEST}_d := \texttt{AROUND}_d(\mathrm{A}, \inf_A)$

To express the extremal preferences with the SCORE_d one, the following scoring functions are defined:

a) for $\texttt{HIGHEST}_d$

$$f(v) := |sup - v| \tag{2.3}$$

b) for $\texttt{LOWEST}_d$

$$f(v) := |v - inf| \tag{2.4}$$

Example 3 Let us take again the attribute `followers_count` used in Example 2. If the user is looking for tweets on a current topic, it could be that he is only interested in the opinion of prominent people. Their accounts have the *most followers*, so with the $\texttt{HIGHEST}_d$ preference the "prominent" accounts can be separated from the "normal" ones:

$$\text{P} := \texttt{HIGHEST}_{d=100000}(followers\_count)$$

Definition 10 $\texttt{MORE THAN}_d$ **and** $\texttt{LESS THAN}_d$
With the help of these preferences the user searches for the values that are smaller or larger (inclusive) than a certain threshold value. $\texttt{MORE THAN}_d$(z, A) and $\texttt{LESS THAN}_d$(z, A) can be derived from $\texttt{BETWEEN}_d$(A, [low, up]) by setting the upper (lower) bound to the supremum (infimum) of the real world. In this way we need to set *up* = *sup* or *low* = *inf* to describe $\texttt{MORE THAN}_d$(A, z) and $\texttt{LESS THAN}_d$(A, z) preference constructors:

a) $\texttt{MORE THAN}_d := \texttt{BETWEEN}_d(\text{A}, [\text{z}, \sup_A])$

b) $\texttt{LESS THAN}_d := \texttt{BETWEEN}_d(\text{A}, [\inf_A, \text{z}])$

The SCORE_d function for $\texttt{MORE THAN}_d$ and $\texttt{LESS THAN}_d$ are technically equivalent to the SCORE_d functions of the extremal preferences $\texttt{HIGHEST}_d$ and $\texttt{LOWEST}_d$ (cp. Functions 2.3 and 2.4 in Definition 9).

Example 4 Using the same attribute `followers_count` as in Example 2, one can imagine that the user is looking for tweets, which were posted by account with a lot of followers. He thinks that it should be *at least* 50000 followers, so with the `MORE THAN` preference such accounts can be found:

$$\text{P} := \texttt{MORE THAN}(50000, followers\_count)$$

Categorical Base Preferences

Numerical base preferences cannot be applied for many domains. E.g., the preferred *language* for selected tweets or used *hashtags* in text messages do not have numerical, but only categorical values. It is possible to map these non-numerical values to numerical ones, but this way is not very intuitive.

There are specially developed preference constructors for easy handling of non-numerical (or categorical) domains. The most general one is $\texttt{LAYERED}_m$. This preference allows to order elements and set them to different sets. $\texttt{LAYERED}_m$ is a sub-constructor of $\texttt{SCORE}_d$ (cp. Figure 2.1) and other categorical preference constructors can be derived from it.

Definition 11 ($\texttt{LAYERED}_m$ Preference)
Let $L = (L_1, ..., L_{m+1})$ with $m \geq 0$ be an ordered list of $m+1$ sets building a partition of $dom(A)$ for an attribute A. P is a $\texttt{LAYERED}_m$ preference if exists a SCORE preference with the following scoring function:

$$f(x) := i - 1 \Leftrightarrow x \in L_i \tag{2.5}$$

To be sure that all domain values are covered by the preference constructor, one of the L_i may be named "others". This special set is disjunct with other sets of $dom(A)$.

Example 5 One of the best known terms related to Twitter is `hashtags`. This metadata tag used in text message by the author helps other users easily find messages with a specific theme or content. `hashtags` are one of the most interesting categorical attributes to specify preferences. Looking for information about the parliamentary elections in United Kingdom, the user, e.g., prefers to find tweets containing hashtags *selectionUK* or *Johnson*. These terms are preferred to *United Kingdom* and *Great Britain* ones. Furthermore the last two are better than *England*. Such preference can be expressed as follows and visualized as shown in in Figure 2.4.

P := $\texttt{LAYERED}_{m=3}$(*hashtags*, {'selectionUK', 'Johnson'},
{'United Kingdom', 'Great Britain'},
{'England'},
others)

Figure 2.4: $\texttt{LAYERED}_m$ preference and level values.

Kießling in [Kie02] introduced some more simple categorical preference constructors that cover the most common cases of $\texttt{LAYERED}_m$ preference.

Definition 12 (Sub-Constructors of $\texttt{LAYERED}_m$)
In POS(A, POS-set) preference the POS-set is preferred over all other values. On the contrary the NEG(A, NEG-set) preference specifies the NEG-set with the values we like at least. POS and NEG preferences can be combined to POS/POS(A, POS1-set, POS2-set) and POS/NEG(A, POS-set, NEG-set) ones and all four of them can be derived from $\texttt{LAYERED}_m$:

a) POS := $\texttt{LAYERED}_1$(A, POS-set, others)

b) NEG := $\texttt{LAYERED}_1$(A, others, NEG-set)

c) POS/POS := $\texttt{LAYERED}_2$(A, POS1-set, POS2-set, others)

d) POS/NEG := $\texttt{LAYERED}_2$(A, POS-set, others, NEG-set)

Example 6 Every tweet obviously has an author. This is the user who posted it. Each Twitter user can be uniquely identified with the attribute `screen_name`. So, if someone, for example, is interested in the presidential election in the USA, he or she could define the preferred and/or unwanted authors with a preference. Maybe you want to have all tweets from *Barack Obama* or *Joe Biden*, but none from *Donald Trump*. This preference can be defined as follows:

P := POS/NEG(*screen_name*, {'BarackObama', 'JoeBiden'}, {realDonaldTrump})[3]

BarackObama and *JoeBiden* are both equally good and better then any other Twitter author, *realDonaldTrump* is the most disliked one.

In Figure 2.1 one can see a group of *geo* preferences: $\texttt{SPATIAL}_d$, $\texttt{NEARBY}_d$, $\texttt{ONROUTE}_d$, $\texttt{BUFFER}_d$ and $\texttt{WITHIN}_d$ (all are dark grey marked), which are not discussed in this work since I do not use them in my stream evaluation framework. For more details about geo preferences I refer to [WKK13]. Also the EXPLICIT preference (highlighted in dark grey in Figure 2.1) is not supported by my framework and is not discussed here, but the datail can be found in [Kie02].

[3] *BarackObama*, *JoeBiden* and *realDonaldTrump* are the Twitter user names of Barack Obama, Joe Biden and Donald Trump.

Complex Preferences

Complex preferences allow to combine several other preferences into a bigger one. For a *complex* preference one has to decide the relative importance of the parts. The combined preferences can be *equally important* or *one* preference can be *more important than the other one*. These two situations are covered with two complex preferences constructors: *Pareto preference* for equal importance and *Prioritized preference* for ordered one.

Definition 13 (Pareto Preference)
Assume preferences $P_1 = (A_1, <_{P_1})$, $P_2 = (A_2, <_{P_2})$, and $x = (x_1, x_2)$, $y = (y_1, y_2) \in dom(A)$, then the Pareto constructor $P := P_1 \otimes P_2$ is defined as:

$$(x_1, x_2) <_P (y_1, y_2) \Leftrightarrow (x_1 <_{P_1} y_1 \wedge (x_2 <_{P_2} y_2 \vee x_2 \cong_{P_2} y_2)) \vee (x_2 <_{P_2} y_2 \wedge (x_1 <_{P_1} y_1 \vee x_1 \cong_{P_1} y_1)).$$

The object p_i dominates another object p_j if p_i is better than p_j in at least one dimension and not worse than p_j in all other dimensions. The object p_i belongs to the *Pareto set* or *Skyline*.

Definition 14 (Prioritization Preference)
Assume preferences $P_1 = (A_1, <_{P_1})$, $P_2 = (A_2, <_{P_2})$, and $x = (x_1, x_2)$, $y = (y_1, y_2) \in dom(A)$, then the Prioritization preference constructor $P := P_1 \,\&\, P_2$ (introduced in [Kie02]) is defined as:

$$(x_1, x_2) <_P (y_1, y_2) \Leftrightarrow (x_1 <_{P_1} y_1 \cong (x_1 \sim_{P_2} y_1 \wedge x_2 <_{P_2} y_2).$$

Prioritization preference defines a *lexicographic order* on a domain, cp. [Cho03]. Preferences found earlier are more important than preferences expressed later in the order. In the complex preference $P := P_1 \,\&\, P_2$, P_1 is more important than P_2, which is only considered if P_1 does not mind.

There are more complex constructors available that I do not discuss in this work because they are not interesting for my framework.

The Better-Than Graph

Preferences, which are strict partial orders, can be visualized using Hasse diagrams - directed and acyclic graphs in which edges between nodes indicate domination of elements w.r.t. preference (see, e.g., [DP02]).

Definition 15 (Better-Than Graph (BTG))
Given a preference $P = (A, <_P)$ on an attribute set A with domain $dom(A)$. The *better-than graph* for P (BTG_P) is then the Hasse diagram of $<_P$ with some additional characteristics:

- one node in the BTG represents one equivalence class in $dom(A)$
- an edge (a_1, a_2) exists in the BTG directed from a_1 to a_2 for each pair of nodes a_1, a_2 for which holds: $a_2 <_P a_1 \wedge (\neg \exists a_3 \in dom(A) : a_2 <_P a_3 <_P a_1)$.

The *level* of a node in a BTG is the length of the longest path leading to it from an undominated node.

Each level value in *base preferences*, which are *weak order preferences*, represents *one* equivalence class. Therefore, the BTG of a base preference is a graph with minimum level $level = 0$ and the maximum level $level = max(P)$ - the maximum level value for the preference P (see Definition 5). There exists exactly *one BTG node* for each value in the value set. Best values have a level value of 0 and belong to the top node; worst values have a level value of $max(P)$ and belong to the bottom node, cp. Figure 2.5.

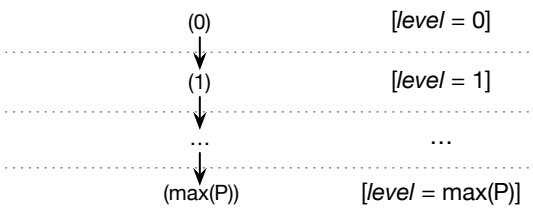


Figure 2.5: Structure of BTGs for WOPs.

Better-than graph for WOPs (see Definition 4) have the following properties (for the proofs see [Pre09]):

Theorem 1 (Properties of BTG for WOPs)
Let $P = (A, <_P)$ be a weak order preference, then:

a) The value of the level function $level_P(v)$ for all $v \in dom(A)$ is equal to the level of v's corresponding node in the BTG.

b) The height of the BTG for P is:

$$height(BTG_P) := max(P) + 1$$

c) The number of different nodes in the BTG for P is:

$$nodes(BTG_P) := height(BTG_P)$$

Let us look at these properties in the following example:

Example 7 The user is looking for tweets with certain *hashtags* using the following `LAYERED` preference:

$$\text{P} := \texttt{LAYERED}_3(hashtags, \{\text{'covid19', 'corona'}\}, \{\text{'pandemic'}\}, \{\text{'lockdown', 'stayhome'}\}, \text{others})$$

Figure 2.6 shows the BTG for this preference and the domain values the nodes are representing.

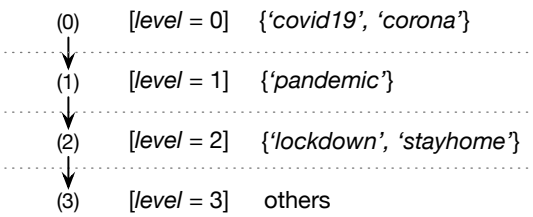


Figure 2.6: BTG for a $\texttt{LAYERED}_3$ preference.

Tweets with hashtags *covid19* and *corona* are preferred tweets, which are in the same *equivalence class* described by level 0. The next level 1 is represented be the tweets with the hashtag *pandemic* followed by messages including *lockdown* and *stayhome* on level 2. The maximum level 3 is reserved for tweets with *other* hashtags. The height of the BTG due to the $\texttt{LAYERED}_3$ preference is 4, which is also the number of nodes.

A BTG for a *complex Pareto preference*, which has only weak order preferences as its components, has also more complex structure (see e.g., [DP02, Pre09]). A complete distributed lattice is an ordered set S, so that for all subsets of S an *infimum* and *supremum* for any two of its elements is defined.

Some properties of BTG for Pareto preference are listed below:

Theorem 2 (Properties of BTG for Pareto Preferences)
Let $P = \otimes(P_1, ..., P_m)$ be a Pareto preference and $P_i, i = 1, ..., m$ be WOPs. Then:

a) The number of nodes in the BTG for P is:

$$nodes(BTG_P) := \prod_{i=1}^{m}(max(P_i) + 1)$$

b) The number of edges in the BTG for P is:

$$edges(BTG_P) := \prod_{i=1}^{m}(max(P_i) + 1) \cdot \sum_{i=1}^{m} \frac{max(P_i)}{max(P_i)+1}$$

c) The overall level value of a node $a = (a_1, ..., a_m)$ in the BTG of P is given by the level function

$$level_P(a - 1, ..., a_m) := \sum_{i=1}^{m} level_{P_i}(a_i)$$

d) The height of the BTG for P is:

$$height(BTG_P) := 1 + \sum_{i=1}^{m} max(P_i)$$

There is *more than one* single node in each level except the minimum and maximum levels (see Figure 2.7). The BTG for Pareto preference P always has a *hexagon shape* and is symmetric with respect to its middle axis. The *top node* has the combination of *best values* and *bottom node* the combination of *worst values* of the base preferences that form the current Pareto preference P.

Each node in an BTG of a Pareto preference can be provided with a unique integer number, the *node ID* (cp. [Pre09]). For this, the *edge weights* have to be defined first.

Definition 16 (Edge Weight)
Let $P = \otimes(P_1, ..., P_m)$ be a Pareto preference and $P_i, i = 1, ..., m$ be WOPs. The *weight of an edge* in the BTG, which expresses domination with respect to any $P_i \in P$ is defined as

$$weight(P_i) := \prod_{j=i+1}^{m}(max(P_j) + 1)$$

With the edge weights the *unique node identifiers* for the BTG nodes can be calculated. Note that in the BTGs of WOPs all edges have a weight of 1.

Theorem 3 (Unique Node ID)
Let $a = (a_1, ..., a_m)$ be a node in BTG_P and ID be mapping such that

$$ID(a) := \sum_{i=1}^{m}(weight(P_i) \cdot a_i)$$

Then the following properties hold:

- The ID is unique for every node in the BTG.
- Every value in the set $\{0, 1, 2, ..., |BTG| - 1\}$ is an ID of a node in the BTG.

Example 8 The BTG in the Figure 2.7 is build for some Pareto preference P, which consists of two base preferences P_1 with maximum level of 2 and P_2 with maximum level of 3. This BTG has 5 levels in total. The top node has base preference level values of $(0, 0)$, while the bottom node has the value combination of the base preferences' maximum levels $(2, 3)$.

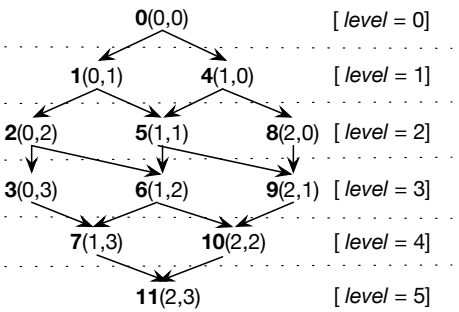


Figure 2.7: BTG for the Pareto preference.

Using Theorems 2 and 3 the following measures can be computed:

- The number of nodes:
  $nodes(BTG_P) = (max(P_1) + 1) \cdot (max(P_2) + 1) = (2 + 1) \cdot (3 + 1) = 12$
- The number of edges:
  $edges(BTG_P) = (max(P_1) + 1) \cdot (max(P_2) + 1) \cdot (max(P_1)/(max(P_1) + 1)$
  $+max(P_2)/(max(P_2) + 1)) = 12 \cdot (2/3 + 3/4) = 17$
- The height of the BTG:
  $height(BTG_P) = 1 + max(P_1) + max(P_2) = 1 + 2 + 3 = 6$

- The edge weights:
  $weight(P_1) = (max(P_2) + 1) = (3 + 1) = 4$
  $weight(P_2) = 1$
- The node ID:
  $ID(1,3) = (weight(P_1) \cdot 1) + (weight(P_2) \cdot 3) = 4 \cdot 1 + 1 \cdot 3 = 7$

The BMO Query Model

Preference query returns *best matches only* (BMO), so the result is called *BMO-set*. It contains objects that match user's wishes best possible: exact matches if such objects exist and best alternatives else. The BMO query result adapts to the quality of the data and is not necessarily perfect. This semantic of BMO query model was proposed by [Kie02] and used by [Cho03].

Definition 17 (Preference Selection, BMO-set)
For a preference $P = (A, <_P)$ on a dataset D, which contains attributes $D(A_1, ..., A_m)$ with $A \subseteq attr(D)$, the BMO set is given by the *preference selection* defined as:

$$\sigma[P](D) := \{t \in D | \neg \exists t' \in D : t[A] <_P t'[A]\}.$$

Preference selection retrieves all perfect values t from the dataset D. If such values do not exist, it deliveres *best matching alternatives* w.r.t. given preference P. All objects in the BMO-set are undominated by others [KEW11, End11].

Example 9 In this example two preferences will be combined, namely

- $\mathrm{P}_{hashtags}$:= POS/POS(*hashtags*, {#yog, #figureskating}, {#isu})
- $\mathrm{P}_{statuses\_count}$:= $\mathtt{AROUND}_{d=500}$(*statuses_ count*, 4000)

to a Pareto preference P:= $\mathrm{P}_{hashtags} \otimes \mathrm{P}_{statuses\_count}$.
Consider Table 2.1, which represents the sample Twitter data about the 2020 Youth Olympic Games (YOG). It shows only a few of the many tweet's attributes. *Id* is used to identify tweets, *hashtags* and *statuses_ count* (the number of posted tweets) are used in Pareto preference P, and the *text* of the tweet is the information the user is looking for. Evaluation the preference P on the given data sample, the BMO set is represented by the bold ids. The tweets with ids 752371 and 377652 are discarded, because the are dominated by another ones. E.g., tweet with id = 170513 dominates the one with id = 752371, because it is better concerning the *status_ count* and indifferent concerning the *hashtags*.

id	hashtags	statuses_count	text
170513	#yog #competitions #scores	1.723	Did Yuma deserve to win #YOG? Yes and by far. Are the scores horribly inflated? Hell yes. Are #competitions with inflated and unrealistic #scores enjoyable and a positive thing for the sport? No, not at all.
861395	#figureskating #teamjapan #iloveyog	1.917	@so_ta0110 impresses in the Mens SP, leads field with 73.07 points. #figureskating #teamjapan #ILoveYOG
752371	#yog	1.329	Watching training of figure skating(men) at #yog `http://yfrog.com/esimeyrj`.
377652	#lausanne2020	1.678	Good day for China in figure skating #lausanne2020: Han Yan leads the men, Yu/Jin leads pairs `http://exm.nr/yfnylq`.
591142	#isu	3.278	After @FSU_Figure started their campaign against ISU's unfair judging, @olympicchannel removed TES scoreboard from the figure skating live broadcast at the Youth Olympics. #ISU haven't got anything to hide, have you?
115214	#figureskating #yog #lausanne2020	6.418	"I did my twizzles better" - Russia's Irina Khavronina gets to the heart of her improved performance in the #figureskating free dance #lausanne2020 #YOG.

Table 2.1: Example of Twitter data about the YOG.

2.1.2 Preference SQL

To evaluate data with preferences, existing SQL database systems had to be extended towards Preference SQL. The University version of Preference SQL adopts a middleware approach as depicted in Figure 2.8.

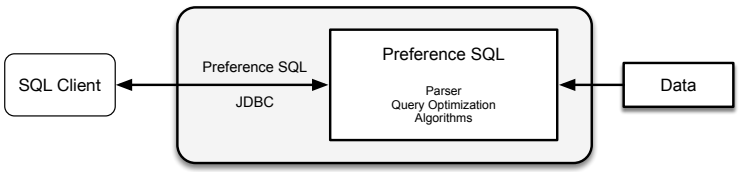


Figure 2.8: System architecture of Preference SQL.

For a seamless application integration standard JDBC technology was extended towards a Preference SQL / JDBC package. This enables the client application to submit Preference SQL queries through familiar SQL clients. The Preference SQL middleware parses the query and performs preference query optimization. The preference selection operator implemented by several evaluation algorithms computes the final result. This approach has proven its flexibility and performance in various pro-

totype applications conducted so far, see [KEW11] for example.

To express arbitrary preferences, the syntax of SQL - Standard Query Language (see [Mel93, AB15, Bea20]) - was augmented by an additional `PREFERRING` clause in [HK05, WK02], which supports soft constraints. The new *Preference SQL* query language supports SQL92 standard as well as base and complex preferences. A Preference SQL query block has the following schematic design:

```
SELECT        ...  <selection>
FROM          ...  <table_reference>
WHERE         ...  <hard_conditions>
 PREFERRING   ...  <soft_conditions>
 GROUPING     ...  <attribute_list>
 BUT ONLY     ...  <but_only_condition>
GROUP BY      ...  <attribute_list>
HAVING        ...  <hard_conditions>
ORDER BY      ...  <attribute_list>
```

Figure 2.9: Preference SQL query block.

A preference is evaluated in the `PREFERRING` clause on the date that remained after hard constraint evaluation in the `WHERE` clause. As a consequence of this, empty result set can only occur, if all tuples have been filtered out by *hard conditions*.

With knowledge of the preference constructors from Section 2.1.1 the syntax of the preference extensions can be found in Table 2.2.

Preference constructor	Preference SQL expression
BETWEEN$_d(A, low, up)$	A BETWEEN low AND up, d
AROUND$_d(A, z)$	A AROUND z, d
HIGHEST$_d(A)$	A HIGHEST sup, d
LOWEST$_d(A)$	A LOWEST inf, d
MORE THAN$_d(A, z)$	A MORE THAN z, sup, d
LESS THAN$_d(A, z)$	A LESS THAN z, inf, d
LAYERED$_m(A, L_1,..., L_{m+1})$	A LAYERED $(A, L_1,..., L_{m+1})$
POS/POS(A, S_1, S_2)	A IN S_1 ELSE S_2
POS(A, S)	A IN S
POS/NEG(A, S_1, S_2)	A IN S_1 NOT IN S_2
NEG(A, S)	A NOT IN S
$\otimes(P_1,...,P_m)$	P_1 AND ... AND P_m
$\&(P_1,...,P_m)$	P_1 PRIOR TO ... PRIOR TO P_m

Table 2.2: Preference SQL Syntax.

Example 10 Assume there are two relations: *tweets* (Table 2.3) and *authors* (Table 2.4) with tweets and authors of these tweets accordingly. Each tweet has an author and the tables are connected by a foreign key (*author*). For example, the tweet with id = 114432 was posted by the author @fs_gossip with 2938 followers (*followers_count*) and 2779 posted tweets (*statuses_count*).

The user is searching for tweets (*text*) of any *author*, only the messages from an author @fs_gossip he does not want to have. He also prefers the tweets that have been "liked" by at least 100 (d = 10) other users (*likes*), which is equally important to the author preference. Just as important as author and number of likes is the verification status of the author (*verified*), it should be *true* if possible. *Preference SQL* expression for this preference looks like this:

```
SELECT t.text FROM tweets t, authors a
WHERE t.author = a.author
PREFERRING a.author NOT IN ('@fs_gossip')
AND t.likes AROUND 100, 10 AND a.verified IN ('true')
```

The query returns the BMO result consisting of the tweet with id = 731556 posted by the verified author @ISU_Figure, which has 79 likes.

id	text	likes	retweets	*author*
114432	"I need to fight the cause, not the effect." Evgenia Medvedeva about withdrawal from the Russian Cup #figureskating #EvgeniaMedvedeva	13	2	@fs_gossip
133761	Maxim Trankov: "The champion position is very precarious. Today they love us, tomorrow they will crucify us." #figureskating	18	3	@fs_gossip
541129	Check out these cool photo captures! These shots are from the Junior Pairs Free Skating of #WorldJFigure 2019! You can watch or re-watch the stream of this event anytime at your leisure: `https://youtu.be/Xyu8kDpKHOs` #UpAgain #FigureSkating	57	8	@ISU_Figure
388644	In case you haven't heard: Italy will be hosting a grand prix series of figure skating - consisting of four events, starting this month. List of events & participants: `shorturl.at/bjyQY` #figureskating #italian #danielgrassl	4	0	@theskatingtimes
961127	Rika Kihira's updated ISU profile lists Lambiel and Hamada as her coaches. #figureskating #rikakihira #stephanelambiel	4	0	@theskatingtimes
731556	Streaming Now! On the "Skating ISU" YouTube channel you can watch the Junior Ice Dance Rhythm Dance from the World Junior Figure Skating Championships of 2019! `https://youtu.be/pZpmISmtF4I` #UpAgain #WorldJFigure #FigureSkating	79	25	@ISU_Figure

Table 2.3: Sample data set containing tweets.

author	verified	statuses_count	followers_count
@fs_gossip	no	2 779	2 938
@ISU_Figure	yes	9 549	91 700
@theskatingtimes	no	105	30

Table 2.4: Sample data set containing authors of tweets.

2.2 Data Streams

As of today, much of the data that we deal with, come as different streams. User interactions on websites and in mobile apps, placements of orders, server logs, sensor measurements - all of these are streams of some kind of events. In fact, it is difficult to find examples of finite, complete datasets that are generated all at once. Naturally, to process and analyze such kind of data you need applications that take into account streams' peculiarities: potential infinity and fast element changeability. Algorithms and approaches used to process static data cannot be used for stream processing without adaptation.

The *data stream* (Figure 2.10) is defined in computer science as a continuous stream of records or objects, the end of which cannot be predicted in advance, in some cases there are no discrete beginning or end at all. The stream records may or may not be related or correlated with each other. They are generally timestamped, in some cases geo-tagged and have a set of attributes specific for the certain stream. For example, data from a car traffic monitoring or messages send on Twitter are continuous and have no *finish*, they are geo-referenced and timestamped.

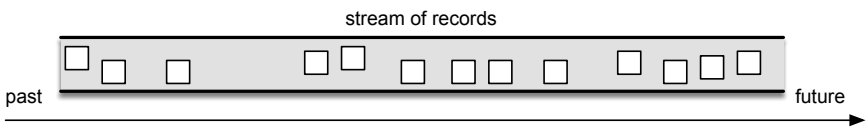


Figure 2.10: Stream of records.

Records in stream are *immutable*, they make a sort of *history*: the records are processed, the new ones are added to the stream and processed as well. It's impossible to go back to older records unless they have been purposefully stored. This new form of data existence is somewhat opposed to the familiar and well known *tables* in traditional database systems, which can be considered in such context as a current *state* of the system. The table is a snapshot of a stream in a certain sense and represents a collection of *mutable* records.

Traditional database systems but also batch data processing efficiently handle large

and seldom changing data that have been pre-loaded into the database or other store and process them only after user had queried explicitly. The results provided to the user reflect the current state of the data, which remain in the database after the query has been processed. If the data have not been changed or deleted from the database explicitly, the result of the repeated query will remain unchanged.

The new class of applications working with data streams, is able to process the incoming stream on the fly and update a query result, which allows to provide the most current answer that takes into account the latest data. As time passes and new records are added to the stream, the query result also changes. Some stream applications do not even wait for request from the end user, to give some response. They check the incoming data continuously and send notification when something remarkable happens. Examples of stream-based applications can be found in such areas as sensor networks, astronomy, infrastructure and traffic monitoring, security, electronic trading and of course social networks [GO03, DP18, CM12].

The individual elements in the data stream can be relational events, sensor readings, web pages visits, etc. They can be very variable, but all the records within a stream have a same type. Stream data are not available for reading from any storage source. To some extent it is possible to store the records gathered from different stream sources. However, the amount of data collected over a few months can be so huge that it will be a very work- and time-expensive task to extract information from them. And while the user waits for the answer to his query, the data change again, because new records are added constantly.

The stream items arrive live, there is no system control over the order of these elements. The data are potentially unlimited and are discarded after processing. Thus, the reanalysis of stream data is usually impossible. The queries used in traditional DBMS and batch processing are called *one-time queries*. They evaluate a snapshot of data once at a certain time point and return the response to the user. On the other hand, *continuous queries* should remain in the system longer and evaluate the stream data continuously as they arrive. The response to a continuous query is updated over time, taking into account all the data received so far [Krä07, BBD+02].

Today, there are two directions in working with data streams: the *stream processing model* [BBD+02] and the *complex event processing model* [Luc01]. The stream processing model considers the information flow processing problem as an evolution of traditional data processing: data from different stream sources are processed through a sequence of analytical operations, such as selection, aggregation, filtering, joining, etc. to get new data streams as a result. The complex event processing model on the other hand considers information flows as notifications of real world's events. The main focus of this model is to identify relationships and patterns among incoming low level events to derivate existing or potential high level events. Whenever something

special happens and will be detected, application raises a flag or sends some kind of notification. [GO03, DP18, CM12].

The operations for data stream transformation and analysis can be both provided by stream processing engine and implemented by the application programmer. There are *stateless* and *stateful* operations. During stateless operations processing of each particular stream record is independent of any other records (previous or following) and the is no need to keep some records (or intermediate result) in the memory. The arriving order of stream records does not affect the processing result as well, so such operations can be easy parallelized. Example of stateless operation can be, e.g., *filtering* (stream records are selected according to specific condition, see Figure 2.11) or simple *transformation* (extracting some attribute values from the record for future operation).

Figure 2.11: Stream of filtered records.

Stateful operations on the other hand, need the information about the records processed in the past or about previous result (state), because it can be changed by the new data. This kind of operation is more challenging and often requires some form of working memory to keep intermediate computation results. E.g., to calculate the total sum of all processed records, the already calculated value should be hold in the memory and will be updated, when next record arrives (see Figure 2.12). However, more often real life applications perform operations that include not all of the stream records seen so far, but only a part of them, e.g. when we want to compute a median temperature for a certain month. For such task it is not enough to hold one calculated value in the memory, we need all stream records, relate to the time period we are interested in, at once. This can be handled by using of *window* approach [KH19].

A window is a certain amount of data, to which we want apply some operations. The records in the window are kept in the working memory and are available for further processing. Window operations create finite sets of stream records called *buckets* or *chunks* and let the application perform computations on these finite sets. Records are usually assigned to the chunks based on their properties: depending on arrival *time* or *number* of objects, one wants to have in one chunk [KH19].

There are different kinds of windows, but I discuss here two most common ones, implemented in Flink[4] and described in [KH19] as well.

[4] Flink: flink.apache.org

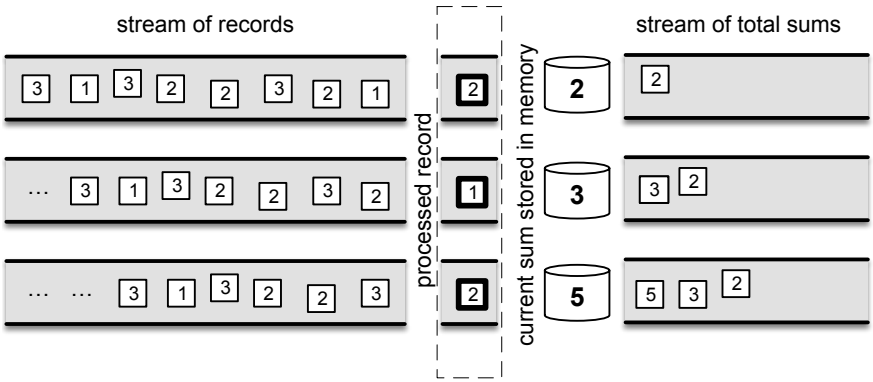


Figure 2.12: Total sum calculation across the stream records.

Tumbling window (also known as fixed window) assign stream records into *non-overlapping* chunks. They can be build *size* or *time* based (see Figure 2.13). When specifying a time, the chunks will be build based on the data retrieved within the given time slot. By specifying a size in the window function, it will determine, how many stream records belong to one chunk. The *full* chunk is then sent to an evaluation function for processing.

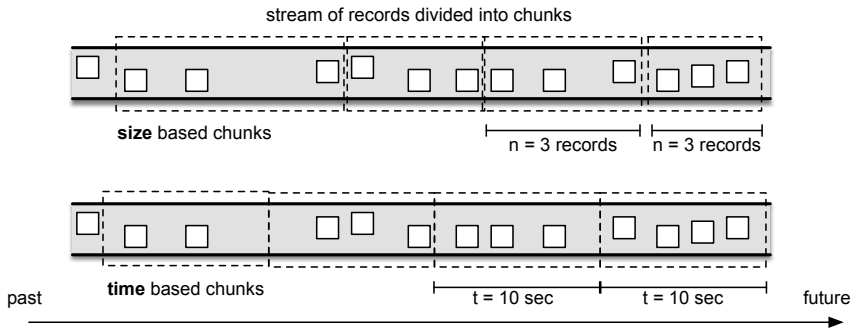


Figure 2.13: Size and time based tumbling windows on the stream.

Sliding Window (or hopping window) - window operation assigns events into overlapping buckets of fixed size (see Figure 2.14). That means, that some records can belong to two "neighbour" chunks. Sliding window has two characteristics: (i) how

many records are within one chunk (*length*) and after how many records new chunk will be created (*slide*) [KH19]. This kind of window allows to analyze the development of data over time.

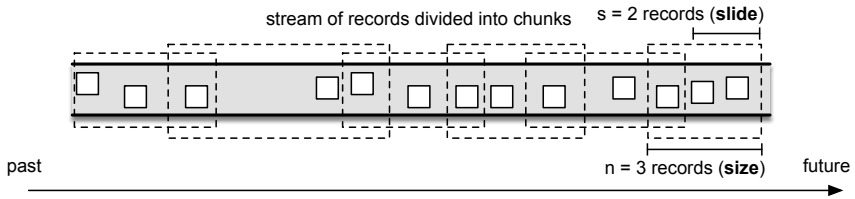


Figure 2.14: Sliding windows on the stream.

Streaming data processing system should be able to manage data on the fly, with ultra low latency, but the main goal - *querying and analysis of stream data in real time* - can not always be reached easily. Stream processing provides a lot of challenges and limits: a very large (sometimes endless) volume of data and only limited memory, impossibility to see the data twice and as result the need to use only one-pass algorithms. Besides, some common data operations make no sence or are impossible on the endless data (e.g. sorting or grouping all over the stream).

All these factors make a task to process and analyze stream data in real-time quite challenging but also very exciting.

2.3 Apache Flink - Framework for Stream Processing

Apache Flink is a top level Apache project and an open-source stream-processing platform for distributed stream and batch processing[5]. Flink was born at Berlin's Technical University, and it used to be called Stratosphere before it joined Apache's program pool in April 2014 and became top-level project in January 2015.

Flink can handle any kind of stream: *unbounded*, *bounded* (with fixed-sized data sets), *real time* streams, but also *recorded* data. Flink's core is a distributed streaming dataflow engine written in Java and Scala that provides data distribution, communication, and fault tolerance for low delay computations over data streams. A Flink setup has four different components that work together executing streaming applications: *JobManager*, *ResourceManager*, *TaskManager*, and *Dispatcher*. These components have the following responsibilities [KH19]:

[5] Apache Flink: flink.apache.org

- The *JobManager* is the master process and controls the execution of a single application. Each application has its own JobManager. The application consists of a logical data-flow graph (JobGraph), which will be converted by the JobManager into a physical dataflow graph (ExecutionGraph). The latter consists of tasks that can partially be performed in parallel.

- Flink has multiple *ResourceManagers* for different environments and resource providers. The ResourceManager manages TaskManager's slots - units of processing resources. The ResourceManager also finds and terminates idle TaskManagers to free compute resources.

- *TaskManagers* are the worker processes of Flink. Flink setup provides multiple TaskManagers, each has a certain number of slots. TaskManager has to register its slots to the ResourceManager, after it has been started. TaskManagers that run tasks of the same application can exchange data with each other during the execution.

- The *Dispatcher* provides a REST interface to submit applications for execution and starts a Jobmanager once the application is submitted. The Dispatcher also runs a web dashboard to provide information about job executions.

The building blocks of each Flink program are *streams* and *transformations*. Transformation takes one or more streams as input, and produces one or more streams as output. Flink programs are mapped to streaming data-flows. Each data-flow starts with one or more *sources* and ends in one or more *sinks*. The data-flows can be imagined as arbitrary directed acyclic graphs (DAGs), consisting of streams and transformation operators (see Figure 2.15).

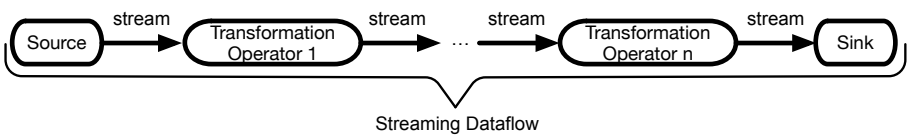


Figure 2.15: Schematic representation of the streaming data-flow.

Let's take a closer look at the dataflow components [Fli]:

- **Data source** is where the input for the program comes from. There are several predefined stream sources, which can be divided into *file-based* (data stream is obtained from a file), *socket-based* (program gets data stream from a socket),

collection-based (data stream will be created from some collection) and *custom* (data stream will be attached with a new source function).

Custom data sources are especially relevant for applications dealing with live data streams (such as Preference Based Stream Analyzer, which is topic of this work). Custom data source can be completely defined and implemented by the programmer and existing connectors can be used to connect to the third-party system. Flink provides connectors, which allow to use following systems: *Apache Kafka*, *Amazon Kinesis Streams*, *RabbitMQ*, *Google PubSub* and *Twitter Streaming API* that I use in my system.

- **Transformation operators** transform *one* or *more* data streams into a *new* data stream. Multiple transformations can be combined into complex data-flow topologies. There are many stream transformations, I will only mention a few of them:

  - **Map:** takes *one* element as input and produces *one* transformed element as output, e.g. number value can be doubled.

  - **FlatMap:** takes *one* element as input, generates *zero*, *one* or *more* elements as output, e.g., text line (input) can be split to the single words (output).

  - **Filter:** if an input element matches a *filter criterion*, it is a part of output, otherwise it is filtered out. For example all even numbers can be output, if a filter criterion returns the remainder equal to 0 when divided by two.

  - **Reduce:** the current input element will be *combined* with the *last reduced value* (previous input elements). So the output is always a "rolling" reduced value of the input elements that were seen so far, e.g., an output stream of partial sums can be produced for the input stream of number values.

  - **Window operators:** *group* the *input data* according to some characteristic. Windows can be time driven (e.g., all data that arrived within the last 10 seconds belongs to the same window) or data driven (e.g., 100 elements per window). One typically distinguishes different types of windows, such as tumbling windows (no overlap), sliding windows (with overlap), and session windows (restricted by session activity). General window concepts on the stream data are described in detail in Subsection 2.2.

  - **Aggregation operators:** (e.g., counts, sums) work on streams differently than in batch processing. It is impossible to count all elements in a stream, because streams are in general infinite. That's why the aggregation operators are applied to the elements inside windows, which are finite.

  Obviously, the named (as well as unmentioned here) transformation operators can perform much more complex actions on the stream data. As an example the framework I have developed and implemented for my doctoral work can be considered. It receives a stream of tweets as the input and must filter them taking user preferences into account (not to confuse with simple filter operator). Details of this system's conception and implementation will be discussed later in this work.

- **Data sink** consumes data streams and *prints* or *passes* them to *files*, *sockets* or *external systems*. Similar to DataSource, *user-specific* data sinks can be implemented. This allows to output the stream data within program or application in the best possible way. For example, in my framework the selected tweets are sent to the display of user's device after the input Twitter stream was evaluated w.r.t. user preferences.

The stream applications are mostly stateful. Only in rare cases, when the transformation operations are applied to a single stream records (without caring about the earlier or later ones), no state is required. So if the application has to calculate and return 60% of each input number, this is realized stateless. In some other cases an intermediate result (state) must be stored to make it accessible at a later point in time, for example when the next record is received or after a specific time period. So if you want to calculate the moving sum over all numeric stream records, the last calculated sum must always be accessible, because with every new stream record this value must be updated.

I chose Apache Flink as a stream delivery and stream processing tool for my framework because it allows real object-by-object stream processing. The alternatives (e.g., Spark), can now do that as well, but at the time when I decided to use Flink, Spark was only able to offer batch processing, which was disadvantageous. Apache Flink allows also to implement stateful stream processing applications. For my framework this is a necessary condition, because the preference evaluation is data dependent. This means that when the new data arrives with the stream, an intermediate result has to be evaluated again. But for this it must be accessible for the new calculation. I will provide the detailed description of preference evaluation on stream data with the appropriate algorithms in the later chapters of this doctoral thesis.

2.4 Twitter - Micro-blogging and Social Networking Service

"Twitter[6] is a micro-blogging and social networking service where users post and interact with messages known as *tweets*" - this definition can be found on Wikipedia[7]. "Twitter is what is happening in the world and what people are talking about right now" - that is what Twitter says about itself[8]. Both statements describe the same social service, but the second one is what distinguishes Twitter from other social platforms and made it special.

Twitter started in 2006 with an idea by Jack Dorsey (one of the inventors and co-founders) to inform small groups of people via SMS about *what they are doing*. The first tweet was posted on March 21th and read *"just setting up my twttr"*[9] (see Figure 2.16). The well known 140 character length restriction is associated with the original character of Twitter as an SMS-based platform. Even when Twitter was already grown into a web platform, the 140 character limit remained for branding and marketing reasons, until finally in November 2017 the maximum length of a tweet has been increased to 280 Unicode characters[10],[11].

Figure 2.16: A very first tweet posted by Jack Dorsey

Twitter is a communication platform, social network and even a publicly accessible online diary. Individuals, organizations, companies and mass media use Twitter as a platform for distributing their content on the Internet. Unlike Facebook, Twitter does not focus on contacts with friends. The registered users can post short massages

[6] Twitter: twitter.com
[7] Twitter: en.wikipedia.org/wiki/Twitter
[8] About Twitter: about.twitter.com
[9] First tweet: twitter.com/jack/status/20
[10] How Twitter Was Born: 140characters.com/2009/01/30/how-twitter-was-born
[11] Th Real History of Twitter, in Brief: lifewire.com/history-of-twitter-3288854

called *tweets*. Tweets are publicly visible by default, but the author can restrict access, so that only his followers will see his tweets.

Following is a central concept of Twitter as a social platform. To *follow* someone on Twitter means to be this account's *subscriber* and to see that user's tweets in own Twitter feed in reverse chronological order (see Figure 2.17). The subscribers are known as *followers*. The number of followers determines how *influential* a particular Twitter account is. Celebrities (actors, singers) or politicians typically have the most followers. E.g., president of the United States Donald Trump is a very active Twitter user with 57.1 thousand tweets and 87.2 million followers (status on 13th October, 2020 and the numbers are growing[12]).

Figure 2.17: A very small excerpt from Twitter account of Donald Trump

Users, especially the younger ones, use Twitter as a news source because of its *real-time* nature: there is a very low delay between an event in the real world and messages about it on Twitter. Twitter has completely changed the way the news spread, more than any other medium in recent history. This micro-blogging service allows people to keep up with developments live, e.g. when an air-plane crash-landed on the Hudson River in New York on January 15th, 2009 (see Figure 2.18), the first messages were

[12] Donald Trump's Twitter: twitter.com/realDonaldTrump

published through Twitter and Tumblr[13]. The generally more popular Facebook and Instagram are much less suitable for posting short text messages about the current situation or for sharing own thoughts than Twitter. Numerous tweets related to events recreate a picture of what happened through the eyes of one particular user. The more different users post about the same event, the more complete picture you get. It could be *social* events such as parties or Metallica show, *political* cases, *disastrous* events such as storms, fires, traffic jams or earthquakes and so on [Wal, SOM13].

Figure 2.18: The first tweet about "Miracle on the Hudson"

Twitter reports about 330 million monthly active users[14], who send 500 million tweets daily[15]. It is impossible to follow everyone, who tweets something. This is also completely pointless, considering that most tweets are interesting only for a (very) limited group of users. To follow particular user's account makes sense if you have a constant interest in the content posted there and want to know on the daily basis *What else will Donald Trump announce via Twitter?* or *Which bag did Katy Perry buy this time?* or even *What's new in Oracle 19c?* But there exist millions of other tweets and not all of them are valuable. It is very typical that an account has very few or no tweets and one day it posts multiple messages about an event that is very important for a certain user group. And the question is, how to find this content if you don't follow the account?

Twitter allows to browse posted tweets using *keywords* or *hashtags* (topics on Twitter written with a # symbol, e.g. #Brexit). The user can also enter a search term, term combination or more complex query and gets tweets containing the words he is looking for. The input query can be very simple (only one word or hashtag) or quite complex. On the Twitter developer website[16] one can find a lot of examples for

[13] Tumblr: www.tumblr.com
[14] Monthly active Twitter users: bit.ly/2FJLMyB
[15] Number of tweets daily: bit.ly/2NmLhPo
[16] Search Tweets: bit.ly/2FReVrv

search queries. Here are some examples, how to use keywords and hashtags in Twitter search:

- *"happy hour"* provides tweets with exactly the entered word combination
- *puppy -filter:retweets* is looking for tweets (but no retweets) with the term *puppy*
- *traffic ?* will return the tweets that includes question and term *traffic*

The formulated search queries can be very diverse. One searches for matches in the attributes *tweet text* and *username*. Further criteria can be specified with restrictions: Twitter allows to choose whether to search among *all existing accounts* or only those *whom the user follows*. It is also possible to choose between *everywhere* (arbitrary location) and *close to you* (there is no information about what Twitter defines as "close").

In addition, Twitter offers the possibility to extend the search with some more attributes (see Figure 2.19). The user can additionally specify the *language* and *accounts* that have posted tweets to which tweets are directed or which are mentioned in the tweets. The interactions can also be taken into account: number of *responses*, *likes* or *retweets* for desired tweets can be pointed out. The last attribute used by extended search specifies a *date* or *time interval* when searched tweets were posted. The date attribute allows the searching only in the past. If it is not specified, the result will be updated after some time has been passed and the tweets posted after query was sent will be also shown to the user.

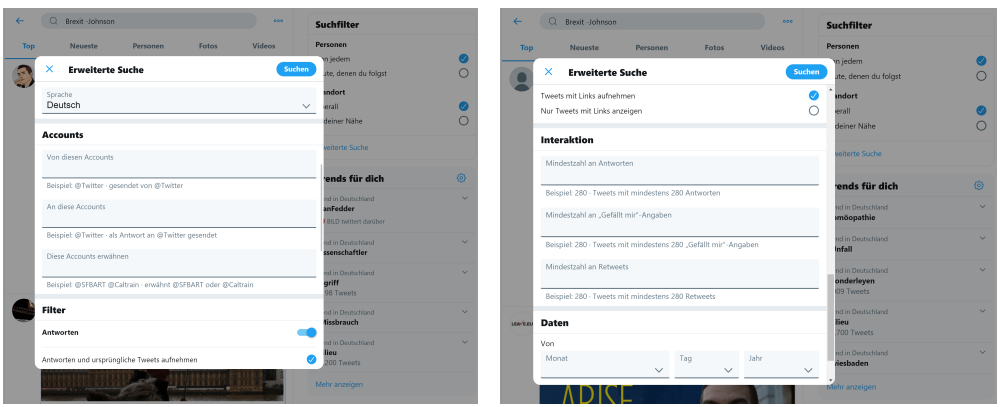


Figure 2.19: Some fields in extended search form

As one can see, the number of attributes for advanced search is relatively small. In return, a *tweet object* provided by Twitter API is much more, than only its content

(text, picture, video), author, language and posting date. Twitter provides the public interface for developers, which allows to get access to the tweets as JSON objects[17]. One tweet includes between 27 and 32 diverse attributes (depending on whether tweet is a retweet or not), which can be either *simple* or *complex*. The latter contain multiple attributes themselves (which can be simple and complex too). For example, **user** attribute in tweet object has 39 attributes itself, including such informative ones as *location*, *verified* (if set, the account of public interest is authentic), *followers_count*, *statuses_count*, *lang* (language) and much more.

Part of these attributes can be found on the user's profile page: e.g., *verified* or *followers_count*. The other part is only partially available: e.g., *created_at* attribute (specifies when the user created his account) stores time up to seconds in JSON object and shows only the month and the year on the profile page. There are attributes that are visible not on every platform: e.g., Twitter App for Android does not show the *statuses_count* - number of posted tweets. However, in the web version, this information is available. And finally, some attributes are hidden from the "normal" user, but exist freely accessible in JSON tweet object. An interesting example for such kind of attributes is *source*, which stores, which software was used to tweet this specific message (*"Twitter for Android"*, *"Twitter for iPhone"* and so on).

Most of the tweet attributes are well suited to be included in search query to filter the data, assuming the user can specify the desired values. Twitter allows this only to limited extent. One of the goals of this work is, among other things, to provide end users with the ability to use all the attributes provided by the JSON object in order to look for interesting tweets. In addition, a search should be conducted on preference basis (see Section 2.1) to achieve the best possible result.

In this chapter I summarized the most important background information: I discussed the preference theory, which I use in my doctoral thesis, including all important preference constructors that will be applied in my stream preference evaluation framework. Much attention has been paid to the subject of data streams and their properties. The differences to persistent data sets were explained. The third discussed topic was Apache Flink. I gave a short insight into framework that is used as provider of Twitter stream data in my system. And finally I discussed Twitter, because tweets received as a stream are the data that I analyze and evaluate with my framework.

[17] JSON: www.json.org

Chapter 3

A Framework for Preference-Based Stream Processing

This chapter describes the general design of my system.

I start with the idea of the developed framework and motivate the necessary steps. The individual parts of this framework are content of the following sections. I start with short description in Section 3.1, how to get tweets as a data stream. Then, the *ETL process* is described in Section 3.2, which explains how the incoming stream data have to be transformed for processing with Preference SQL. Much attention is paid to individual tweet attributes, especially those that may be useful for further data filtering. Section 3.3 addresses the challenges that arise during *evaluation of stream data* with Preference SQL. Since the stream data are continuous and, in principle, infinite over time, there is no final result at any given fix point in time. It must always be reevaluated with new data in mind. Finally, Section 3.4 deals with data filtered with Preference SQL. In order to make it easier for users to consume the delivered results, I want to post-process them and show to the users automatically *summarized text* messages without duplicates and irrelevant posts.

I want to analyze data streams exploiting the user's defined preferences. My approach is based on the Preference SQL prototype system [KEW11]. The system was developed to run queries against bounded data sets that are stored persistently in a relational database. However, to analyze streams (continuous, unbounded collections with new elements or events arriving over time) it is necessary to extend the Preference SQL system in order to process such kind of data. Hence, I need a framework that transforms the data from a stream into a Preference SQL usable format (cp. [RE17]).

I use Apache Flink (see Section 2.3), an open source platform for scalable stream and batch data processing, to transform continuous data into a Preference SQL compatible and processable format. Figure 3.1 depicts my stream processing framework for preference evaluation. The incoming data stream is processed by Apache Flink in an ETL (Extract, Transform, Load) process. Since streams are often encoded in various formats, data must be transformed into a Preference SQL readable form. For this, one has to implement the mapping of a stream object to a relational structure inside the *StreamProcessor*, which provides a corresponding interface implementation. The *DataAccumulator* builds (finite) chunks of objects, which then can be processed by Preference SQL to find the best-matches-only (BMO) set w.r.t. the preference specified by a user (cp. [RERK16, EKR18]).

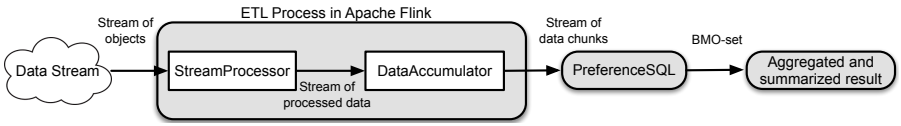


Figure 3.1: Streaming architecture for preference analytics.

In order to improve my framework and make it easier for users to deal with the results delivered by Preference SQL, I want to post-process them and show to the users automatically summarized text messages without duplicates and valueless posts.

3.1 Twitter as Stream Source

Twitter accounts and tweets are usually public and available to all internet users. For example, you can go to the Elon Musk account and read what he posted so far. But my framework for preference tweet analysis requires the data as incoming continuous stream of records. So I need a stream of tweets, which I will evaluate with preferences to give the user the best possible result.

As a stream processing framework, Flink comes with a few basic pre-implemented data

sources and sinks (cp. Section 2.3), but everyone can write his own one if required. Sources are where the application reads its input from. A source should be attached to the program by using `StreamExecutionEnvironment.addSource(sourceFunction)`.

There are several predefined stream sources accessible from the StreamExecutionEnvironment: *file-based*, *socket-based*, *collection-based* and *custom*. The last one is of most interesting one: Flink Streaming has a build-in TwitterSource class, which allows to connect it using the Twitter Streaming API. To use this connector, the following Maven dependency has to be added to the project (x and y stand for the corresponding version number):

```
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-connector-twitter_y</artifactId>
  <version>x</version>
</dependency>
```

Also one needs to attach a new (Twitter) source function:

```
DataStream<String> strSource = env.addSource(new TwitterSource(props));
```

For user authentication TwitterSource class requires as input properties by Twitter: *consumerKey*, *consumerSecret*, *token*, *secret*. To get access to these authentication properties the developer has to have Twitter account and register his application. The properties values look like this:

```
consumerKey=NQEk0KbczVbaAcjCWLksbodkN
consumerSecret=HDKxyp2REOHvuq19oKrZZsdAovItwG6upOGJuSNwbtr6npp2c3
token=2402192752-DQSeCtehr68SerQVyjHLzpHhMvtcwJQlbvwxnLi
secret=BAk0krCYq7X84p45UfyuAuNnpR3nrv9WofO9PNL46XFch
```

After successful connection, the TwitterSource provides a stream of strings containing a JSON-objects, representing tweets.

3.2 ETL Process

When processing data streams with Preference SQL, two fundamental tasks must be carried out:

1. The data must have a Preference SQL processable format, because Preference SQL works on *attribute based* data, but streams are encoded in various formats.

2. The result computation must be adapted to stream properties, since the dataflow is continuous. There is no "final" result after some data of the stream is processed. Hence, the result must be calculated and adjusted as soon as new data arrive.

3.2.1 Tweet Representation in a Preference SQL Processable Format

The objects delivered by a data stream are encoded and not compatible with Preference SQL. That means the data must be structured and needs an attribute based format like `attributeName` = *attributeValue*. The stream objects are transformed into a list of single attribute values by the `StreamProcessor`, cp. Figure 3.1. For this one has to implement the mapping of the object in the stream to a table structured format. The data types of the attributes can be extracted from the stream objects.

In my framework I use Twitter as a stream data source, therefore the delivered objects are tweets encoded in JSON format. JSON (JavaScript Object Notation) is an open-standard, lightweight, text-based, language-independent syntax for defining data interchange formats. JSON was first presented to the world at the JSON.org website in 2001. A definition of the JSON syntax was published as IETF RFC 4627 in July 2006 [ECM17].

A JSON text is a sequence of Unicode tokens that strictly conforms to the JSON grammar defined by the specification. The set of tokens includes six structural tokens ([- left square bracket,] - right square bracket, { - left curly bracket, } - right curly bracket, : - colon, , - comma), strings, numbers and three literal name tokens (`true`, `false`, `null`). JSON data format transmits objects, whose structure is represented as a pair of curly bracket tokens surrounding zero or more *attribute–value* pairs. A JSON **value** can be an *object*, *array*, *number*, *string*, `true`, `false`, `null`. An **attribute** is a string. A single colon token follows each attribute, separating it from the value. A single comma token separates a value from a following attribute [ECM17, ECM19].

A simple example of an JSON-object representing a person's data looks like this:

```
{
"persNr":22111786,
"name":"Lena Rudenko"
"married":false
"email":"lenar@mail.ua"
"phoneNumbers":[{"type":"home", "number":123},
                {"type":"mobile", "number":12345}]
}
```

Figure 3.2: A very simple JSON object example.

The object from the example above has 5 attribute-value pairs: the values are strings (*Lena Rudenko*, *lenar@mail.ua*), number (*22111786*), `false`-token and an array - an ordered list of values, each of which may be of any other JSON type. The values in our example are in turn JSON-objects too.

Tweets are the basic building blocks of Twitter. The tweets provided by Apache Flink as JSON objects are complex structures of attribute-value pairs: The simplest tweet (no retweet, no quote) has 27 such pairs. Some values may be complex objects themselves again, e.g., *user*. This JSON-object contains attributes describing the user who sent the current tweet.

Figure 3.4 shows the JSON representation behind the rendered display of the tweet in Figure 3.3. This example is a very simplified version of the original object to preserve readability. The complete JSON representation of a tweet can be found in Appendix A.1.

Figure 3.3: Tweet screenshot by user @endocrinez


```
{"created_at":"Thu Oct 31 15:28:16 +0000 2019","id":1189927073093181440,
"text":"a personal reminder that I've done it to myself again and no man is
worthy of my time","source":"Twitter for Android",...,"user":
{"id":1158758906295922688,"name":"allusion of form",
"screen_name":"endocrinez","location":null,"url":null,"description":"a dream
diary only it's my waking life","translator_type":"none","protected":false,
"verified":false,"followers_count":27,"friends_count":39,"listed_count":0,
"favourites_count":229,"statuses_count":119,
"created_at":"Tue Aug 06 15:17:06 +0000 2019",
"utc_offset":null,"time_zone":null,"geo_enabled":false,...},...,
"favorited":false,"retweeted":false,"filter_level":"low","lang":"en",
"timestamp_ms":"1572535696658"}
```


Figure 3.4: Simplified JSON object of a tweet.

As one seen even in the simplified example from Figure 3.4, the tweet attributes are very different. For example, this tweet was posted on Tuesday, October 31, 2019 at 15:28:16 UTC (***created_at:****"Thu Oct 31 15:28:16 +0000 2019"*) and has a specific identifier (***id:****1189927073093181440*). There are several different fields which describe a tweet in detail, e.g., *name*, *description*, *followers_count*, *statuses_count*, *lang* and many more. The StreamProcessor converts such objects into more structured data,

e.g., into `created_at` = *"Thu Oct 31 15:28:16 +0000 2019"* and defines the data type "VARCHAR".

Not all of tweet attributes have the same meaning for my framework: e.g., *created_at* tweet attribute will always have a "now"-value - the current date and time in relation to the moment the user uses the framework. That is because my framework should analyze the live stream data and receives them at the moment they appear. Also the attribute *retweeted* has always the value `false`. The current tweet was captured by my system at the time it was created. It is technically impossible that the message was already retweeted at that time. However, it is clear that for another system and another use cases that does not require live analysis, these attributes can be of great importance.

I focus on the attributes[18] that are of interest for my Preference Stream Analyzing Framework:

- `text` - actual text of the current tweet object, in UTF-8 encoding. This attribute contains the information, the user is looking for in the most use case scenarios. Can also be used for filtering the result set, if, for example, the occurrence of a certain word in the text is desired. A separate preference `CONTAINS` was developed and implemented for this purpose (cp. Chapter 4).
- `source` - application used to post the tweet. Messages from the website have a source value of *web*. Can be used as filter, if the user wants to have or not to have tweets from certain utilities. For example, one can assume that if the tweet was sent by *web*, the author was in front of the personal computer, maybe at home or in a cafe, but not somewhere in a big crowd.
- `user` - author of the tweet. A JSON-object with a large set of own attributes.
- `user.name`, `user.screen_name` - `user.name` is the identifier of the user, as he had defined it himself, so not always a real name. The `user.name` is not unique and can occur in different accounts. `user.screen_name` is unique and identifies a specific account, typically has a length of 15 characters.
- `user.location` - user-defined location for his account, not always a real location. It should be used carefully: The values lie in a range between e.g., *"London, UK"* and *"In Your Heart"*, can also often be `null`.
- `user.description` - description of own account entered by the user. The value can be `null`. Most users give a short summary with keywords about the topics they are interested in.

[18] Full Description of all Tweet Objects:
developer.twitter.com/en/docs/tweets/data-dictionary/overview/tweet-object

- `user.verified` - Indicates for accounts of public interest that the respective account is authentic, if the value is `true`. Can be very useful, when looking for reliable information that verifiably has been posted by a certain ("trusted") account.

- `user.followers_count` - number of this user's account followers. This attribute can be very helpful, if we want to filter accounts with exclusive information: The accounts with the most followers are those of politicians, actors, musicians, media, etc. Because most users seek for information from these sources, a high value of `user.followers_count` can indicate an increased relevance.

- `user.friends_count` - number of accounts this user is following. The more influential the user, the greater the difference between `followers_count` and `friends_count`. Twitter Account von Donald Trump (@realDonaldTrump) has more than 67 millions followers and follows only 50 other users.

- `user.favourites_count` - number of tweets the user liked.

- `user.statuses_count` - number of tweets (including retweets) posted by the user. The utility of this value depends on the actual user's interest: tweets about the current political situation from the accounts with small number of statuses are hardly worthy of attention. But some interesting details to certain local sport competition or music event can be posted by accounts with a high or low number of tweets.

- `user.created_at` - date and time in UTC[19] format the account was created. This attribute can be used to filter messages according to the "age" of the sender. Sometimes, news from rather "mature" sources are more desirable; "fresh" accounts might correlate to a "hotness" score of their tweets.

- `coordinates` - precise geographical position of the user at the moment when posting the tweet. If the value is set, `coordinates` is a very good filter attribute.

- `place` - this attribute indicates a place the tweet is associated to.

- `entities.hashtags` - a word or phrase preceded by a hash sign (#) to identify messages on a specific topic. `hashtags` are very useful (if not `null`) to filter the tweets with the certain subject.

- `lang` - indicates machine-detected language of the tweet text. Is useful to select the tweets, which the user can read an understand.

[19] Coordinate Universal Time (UTC)

The attributes `hashtags`, `place`, `coordinates`, `location` and `description` may (and often are) undefined (`null`).

The summary of the listed attributes with short description can be found in Appendix A.3, Table A.1.

3.2.2 Building of Chunks

Preference SQL was developed to evaluate *finite* data. So some preparations would have to be done to make the queries on the *endless* stream data possible.

After the tweet encoding and preparation the Twitter stream must be splitted into parts to be processed in Preference SQL. These parts are finite and therefore can be evaluated with Preference SQL. The splitting is done by grouping objects into chunks. This corresponds to the *tumbling window* concept described in Section 2.2: Stream of individual tweets is converted into a stream of the groups of tweets. Each group contains a certain (finite) amount of members. The grouping occurs in the `DataAccumulator`, see Figure 3.1. It takes a stream of processed objects as input and provides a stream of chunks as output. The grouping is based on *size* (how many objects are in one chunk) or *time* (the number of objects per chunk is determined by the time). Each chunk can then be evaluated with Preference SQL.

3.3 Preference SQL Evaluation on Data Streams

After the ETL has been completed by *StreamProcessor* and *DataAccumulator* (see Figure 3.1), the stream data can be evaluated with Preference SQL.

The first tweet evaluation tests have shown that the existing preference constructors are not sufficient for the analysis of the tweets' texts. The existing categorical preferences (cp. Section 2.1.1) work quite well if the domains values are finite. For example, book titles, country names, animal species or car colors can be searched and evaluated with existing categorical base preferences. But they obviously will not work with *free text* attributes, such as comment, note or tweet. So I was also faced with a task to *design a preference that is able to handle the tweets' plain text.*

Still, the evaluation of stream data is not a trivial task even after all preparations: the result of the individual finite chunks can be simple determined, but the overall result is not equivalent to simple combining the results of the individual chunks. To illustrate this, let us look at the following example:

Example 11 Assume, the tweets from Table 2.1 are the input data that shall be evaluated within my framework. We consider the following simple preference query:

Find all tweets with hashtag #isu. If such tweets do not exist, the messages with hashtags #yog or #figureskating are good alternatives. It is a POS/POS categorical base preference:

$$P_{hashtags} := \text{POS/POS}_{m=3}(hashtags, \{\#\text{isu}\}, \{\#\text{yog}, \#\text{figureskating}\})$$

If we consider the entire input stream at once, the BMO-set includes *one tweet* (it is also a *perfect match*): *"After FSU_Figure started their campaign against ISU's unfair judging, olympicchannel removed TES scoreboard from the figure skating live broadcast at the Youth Olympics. #ISU haven't got anything to hide, have you?"*. This tweet includes the *hashtag #ISU* while the other candidates do not.

But if the input data was split into chunks and the BMO-sets of each chunk was combined, the result is different.

Let us look at the input data again:

id	hashtags	statuses_count	text
		chunk 1	
170513	#yog #competitions #scores	1.723	Did Yuma deserve to win #YOG? Yes and by far. Are the scores horribly inflated? Hell yes. Are #competitions with inflated and unrealistic #scores enjoyable and a positive thing for the sport? No, not at all.
861395	#figureskating #teamjapan #iloveyog	1.917	@so_ta0110 impresses in the Mens SP, leads field with 73.07 points. #figureskating #teamjapan #ILoveYOG
752371	#yog	1.329	Watching training of figure skating(men) at #yog `http://yfrog.com/esimeyrj`.
		chunk 2	
377652	#lausanne2020	1.678	Good day for China in figure skating #lausanne2020: Han Yan leads the men, Yu/Jin leads pairs `http://exm.nr/yfnylq`.
591142	#isu	3.278	After @FSU_Figure started their campaign against ISU's unfair judging, @olympicchannel removed TES scoreboard from the figure skating live broadcast at the Youth Olympics. #ISU haven't got anything to hide, have you?
115214	#figureskating #yog #lausanne2020	6.418	"I did my twizzles better" - Russia's Irina Khavronina gets to the heart of her improved performance in the #figureskating free dance #lausanne2020 #YOG.

Table 3.1: YOG data from Table 2.1 splitted in two chunks.

In Table 3.1 one can see tweets split into two chunks. It is the *output* (stream of chunks) after *DataAccumulator* has done his job (cp. Figure 3.1) and the *input* for Preference SQL, which evaluate the incoming chunks one after another. The tweets in the first chunk do not include hashtag *#isu*, but all three of them have *#yog* or *#figureskating*. Therefore the BMO-set of chunk 1 consists of all three tweets

belonging to this chunk. After that, chunk 2 is evaluated. This chunk includes one tweet with hashtag *#isu*, which is also the best match. Obviously the end result is not correct in terms of Preference SQL if the BMO-set of chunk 1 and the BMO-set of chunk 2 will simple be combined. For the correct evaluation an other strategy should applied.

Dividing a data stream into chunks allows to compute chunk-wise correct results for the entire stream. With a continuous data flow, its sequencing continues producing more recent chunks that then may contain better objects w.r.t. the user preference that can clearly be seen in the Example 11. However, my goal is to provide the user with a final correct result at any given moment of time, regardless of how many chunks have been processed. Hence, each temporary BMO-set must be compared to the objects in the subsequent chunks provided by the ETL process.

In detail, let $c_1, c_2, ...$ be the chunks provided by the ETL processor. The Preference SQL system evaluates the user preference P on the first chunk, i.e., $\sigma[P](c_1)$. Since c_2 could contain better objects the new objects from c_2 have to be compared with the current BMO-set, i.e., it will be computed $\sigma[P](\sigma[P](c_1) \cup c_2)$, and so on. However, this leads to a computational overhead if c_2 is large. Therefore I apply a pre-filter to c_2, i.e., I first compute $\sigma[P](c_2)$ and afterwards apply the preference selection to the union of $\sigma[P](c_1)$ and $\sigma[P](c_2)$, since the following holds [HK05, CCM13]:

$$\sigma[P](c_1 \cup c_2) = \sigma[P](\sigma[P](c_1) \cup \sigma[P](c_2)) \tag{3.1}$$

This leads to a correct result. However, this method does not guarantee *efficient* behaviour. In particular, the rapid update of incoming tweets results in a frequent change of most recent chunks and, as a consequence, requires a constant recomputation of the most recent BMO-sets. Development and implementation of an *algorithm that can solve the task of evaluating stream data quickly and efficiently* was one of the core tasks of my doctoral thesis. For this I will present the Stream Lattice Skyline algorithm in Chapter 5.

3.4 Aggregation and Summarization of Tweets

After evaluating incoming tweet streams w.r.t. user's defined preference, the result set has to be presented to him. Due to the nature of Twitter, even the reduced set of information could be too large on the one hand or contain duplicates and incomplete snippets on the other hand. Thus, it is not enough to only provide personalized information to the user. It is equally important to present the result in a way that it can easily be consumed. Also, the number of best matching objects delivered to

a user after evaluating a stream with Preference SQL can be quite large, especially if these objects are not the perfect matches. In order to improve my framework and make it easier for users to consume the results delivered by Preference SQL, I want to post-process them and show the users automatically summarized text messages with duplicates and irrelevant posts removed.

Let us imagine a user wants to find some facts about the soccer team Germany at the World Cup 2018 in Russia. He knows that Germany just had their last group game and wants to find out everything about this topic. Our user formulates his preference query using the hashtags *#Germany* and *#WorldCup2018* and receives the following set of tweets as result:

1. *It's a sad day for all of Germany and the World Cup.*
2. *After a loss against South Korea, team Germany leaves the World Cup.*
3. *0:2 defeat in the last group match and team Germany leaves the World Cup in Russia.*
4. *RT @UserXyz It's a sad day for all of Germany and the World Cup.*

Each of these tweets reports about the same event - *soccer team Germany leaves the World Cup after group stage*. The first message describes some user's emotions and is less important for someone looking for factual information. The following two tweets provide facts (however, incomplete). The last one is a retweet of the first message with no additional information at all. We will get a full picture of what happened reading the both messages and summarizing the information from them: *„Team Germany loses its last group match against South Korea 0:2 and leaves the World Cup in Russia.“* The first and last tweets can be ignored because it does not contain any relevant information.

Automatic summarization of documents has been a topic of research work for a long time. In most cases, however, this term refers to a summarization of longer text documents. The result of such a summarization is a short summary of one or more longer documents to give users a brief overview of the combined content. The summary is always shorter than the original document and for the most part uses phrases and expressions from the original documents. Summarization of micro-blog posts, such as tweets, has different goals and tasks.

I want to collect and summarize the incomplete information from different tweets removing duplicates and messages, which don't contain any valuable facts for better perception by the user. The resulting message can be longer than individual source tweets and should contain more facts about the event. Details about Tweet aggregation can be found in Chapter 6.

In this chapter I have given a short overview of the architecture of my framework: I described the necessary transformations that should be done with the incoming data to make them evaluatable with Preference SQL. Furthermore I discussed the evaluation and identified a few open building issues. They include a) development and implementation of a *new preference* that is compatible with a short plain text, b) development and implementation of an *efficient algorithm* for preference evaluation on the data streams and c) *automatic summarization* of the filtered short messages for better user perception. This will be discussed in the next chapters of this work.

Chapter 4

Tweet Text Processing

This chapter describes the new CONTAINS preference.

In my doctoral thesis I deal with the preference based analysis of a very special kind of data streams - namely tweets. A person who is familiar with Twitter as a simple user usually does not realize how much information (apart from the actual text message) a tweet contains. Many attributes, such as *followers_ count* or *location*, can be filtered well with preferences to find the Twitter posts that are interesting and relevant to the user. Despite the variety of available preferences, there is no one, which can evaluate text messages. In this chapter I describe a *new preference* for evaluation of short plain texts. The idea and formal definition of this preference are the content of Section 4.1. The following Section 4.2 deals with *Natural Language Processing on tweets*: due to the informal natur of the tweets, some preprocessing steps have to be applied to these short messages before they can be evaluated with the new CONTAINS preference. In the last Section 4.3 of this chapter I describe results of some *experiments* done with the new preference.

Section 2.4 gives a brief overview of the micro-blogging service Twitter and shows that the core substance of this social network - *tweets* - are much more than simple text messages. Detailed description of the tweet's attributes, which are relevant for my framework, can be found in Section 3.2). The most of these attributes can be evaluated with the existing preference constructors (see Section 2.1). But despite the variety of available preferences I realized that there is no suitable preference constructor for evaluation of tweets (in the meaning of short free text messages). The existing categorical preferences (cp. Section 2.1.1) work quite well if the domains values are finite, which tweets are clearly not. Literally anything can appear in the text. A tweet is limited only by the number of characters and can include typing errors, abbreviations, various spelling forms and other special language constructs.

Thus, my task was to come up with a new preference constructor to search on full-text data and to handle natural language mistakes and other tweets' special language features if possible.

4.1 CONTAINS Preference

The existing categorical preference constructors are well suited to evaluate the attributes with the finite domains, such as *book titles*, *country names*, *animal species* or *car colors*. A preference query $\texttt{LAYERED}_{m=1}$(*color*, {'white'}, others) will return all objects that have a value *white* in the attribute *color*. If no such objects exist, objects of any color are returned. This approach will obviously not work with free text attributes, such as *comment*, *note* or *tweet*.

CONTAINS base preference has been developed in order to apply the preference queries to the texts, cp. [RH20, RHE20]. It has a similar structure as LAYERED (cp. Definition 11). The user defines some levels with a set of words or terms he would like to see in the result set. The general idea is as follows: If the text message *contains* some term from one of the levels, it gets a number value corresponding to the level number, the term belongs to. The smaller the value, the more preferred the text is. The texts with the smallest values belong to the evaluation result set after all.

Hence, the preference query *text* $\texttt{CONTAINS}(('figureskating'), ('isu'))$ means that the user is looking for all tweets from the current stream including the term *figure skating*. If no such texts exist, the messages including *isu* should be returned.

Definition 18 (CONTAINS Preference)
Let $L = (L_1, ..., L_{m+1})$ with $m > 0$ be an ordered list of $m + 1$ sets with terms t_i. For m sets a user defines the terms as those he wants to see in the result set. Additionally there is also another level with the set called *others*. All terms that will be found in tweets but are not listed in the sets of levels $1, ..., m$ belong to the *others*-set. All

sets are disjunct, each term t_i belongs to one set only. The terms within a set are indifferent. Then I define the CONTAINS preference as $\texttt{CONTAINS}_m(A, L_1, ..., L_{m+1})$. Assume x and y are two terms from a set of all terms that occur in tweets. Thus the following applies:

$$x_i <_P y_j \text{ iff } x_i \in L_i,\ y_j \in L_j,\ j < i \text{ for } i, j \in \{1, ...m+1\} \tag{4.1}$$

$$x_i \notin L_1, ..., L_m \Rightarrow x_i \in others \tag{4.2}$$

$$x \wedge y \in L_i \Leftrightarrow (x \cong_P y) \tag{4.3}$$

For each considered text message a *score value* is calculated. The *lower* the score value for a message, the *more preferred* it is. A text can contain terms belonging to *different levels*.

Theorem 4 (CONTAINS Preference Score Function)
Let $t_1, ..., t_n$, $(n > 0)$ be terms in some text message t that belong to some different levels $L_{1,...,m+1}$. The score function is defined as:

$$f(t) = \min(\{i - 1 | i = 1, ..., m + 1\})$$

It is necessary to prove that $\texttt{CONTAINS}_m(A, L_1, ..., L_{m+1})$ with the scoring function from Theorem 4 is a *preference* w.r.t. Definition 1:

Proof.

- **irreflexive**: $\neg(x <_P x)$
  Let $x \in dom(A)$. Let also assume k is the smallest level of x such that:
  $f(x) = min(i - 1 | i = 1, ..., m + 1)$
  $f(x) = (k - 1) \Rightarrow (x <_P x) \Leftrightarrow f(x) > f(x)$
  $(x <_P x) \Leftrightarrow (k - 1) > (k - 1) \Rightarrow false$

- **transitive**: $(x <_P y) \wedge (y <_P z) \Rightarrow (x <_P z)$
  Let $x, y, z \in dom(A)$, then: $x <_P y \wedge y <_P z \Rightarrow f(x) > f(y) \wedge f(y) > f(z) \Rightarrow f(x) > f(z) \Rightarrow x <_P z$, since $f(x)$ is a monotone function.

□

Note that the other score function, e.g., average or weighted average over level values would also be conceivable.

Example 12 The score value for the following tweet text t := *"After @FSUfigure started their campaign against ISU's unfair judging, @olympicchannel removed TES*

scoreboard from the figure skating live broadcast at the Youth Olympics" should be calculated. Regarding the preference P := `CONTAINS`$_{m=2}$(t, {'figure skating'}, {'isu'}, others), t has the score value $f(t) = 0$, even though the text includes terms from levels L_1, L_2 and from *others* (here L_3).

4.2 Natural Language Processing of Tweets

Tweets are rather short messages with the maximum length of 280 characters, which can contain not only words but also *links*, *special characters* (e.g., emojis), *references* to other user accounts and *hashtags*. Tweets also have a high percentage of typing errors, abbreviations and other language features, which is not surprising when you consider that 83%[20] of all users are those who use the mobile Twitter application and post fast without distracting from other activities.

Not only the content diversity of the tweets, but also various forms of their terms (*its*, *it is* or *it's* as variants of the same expression) refer to the challenges one faces when evaluating tweets with `CONTAINS` preference. To map different spelled or misstyped terms to the terms defined by this preference, some techniques from the field of Natural Language Processing have been incorporated into a preprocessing step.

Natural Language Processing (NLP) is a subfield of linguistics, computer science, information engineering, and artificial intelligence that involves a wide collection of techniques for language analysis and language development with the aim to achieve the most natural result as possible [Lid01].

The collection of NLP techniques can be divided into *syntactic* and *semantic* methods, depending from information level they handle (cp. [CWB+11]). *Semantic* methods refer to the *meaning* of input (cp. [LM11]): Answering user questions automatically, summarizing the topic of a document, Named Entity Recognition (finding personal names and locations in texts) are some examples [Lid01, CWB+11]. *Syntactic* methods, on the other hand, deal with the *structure* of a word, sentence or even complex text [LM11]. They include, for example, lemmatization, stemming, parsing, part-of-speech (POS) tagging and the segmentation of a text into sentences or words [CWB+11, MS98]. Parsing describes the segmentation of words in a sentence into subject, predicate and object. POS tagging assigns each word to its part of speech, while lemmatization and stemming reduce a word to a basic form.

The new `CONTAINS` preference should be able to find the tweets where the searched terms are written incorrectly or different. For this I use some of the previously mentioned *syntactic methods* from NLP. To keep the effort of the work within reasonable

[20] Statista: de.statista.com/statistik/daten/studie/541918/umfrage/anteil-der-mobilen-monatlich-aktive-nutzer-von-twitter-weltweit/

limits, I have concentrated on the tweets in English.

I have concentrated on *nouns*, which are naming words and identify persons, places, things, animals, feeling, ideas, things, events, substances, or qualities. The proposed approach can be extended with little effort to a few more parts of speech, such as verbs, adjectives, adverbs etc. Note that the nouns are also expected as input terms for `CONTAINS` preference.

4.2.1 Preprocessing of Tweets

The following steps are applied to a text before a score value can be calculated with regard to the preference: the tweet text is *normalized*, *segmented* into *sentences*, which are further divided into *single words*. For these words, a *part of speech* is determined and the *nouns* are sorted out of the whole set. The nouns will be stemmed in the last step. A graphic sequence of this process as a pipeline can be seen in Figure 4.1. For some preprocessing steps the existing tools and libraries were used (e.g., *determination of POS tags* is done with the help of Penn Treebank Tagset [MSM93] and for *stemming* Porter Stemmer [Por80] is used). The other steps were implemented specifically for this work to better cover the special character of tweets. This is the case especially with *normalization*, which has to handle, e.g., *@username references* in the tweets, which do not appear in other texts.

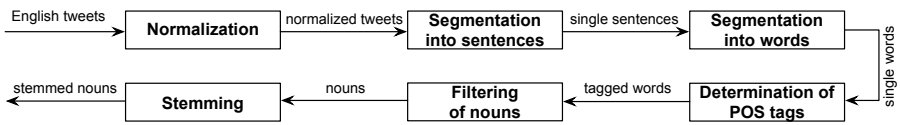


Figure 4.1: Pipeline of tweets preprocessing steps.

Let us take a closer look at the individual steps:

1. **Normalization.** In this step, *emojis* and *links* are removed from the tweet text, as they are an obstacle to part-of-speech definition and obviously cannot be specified as terms in the preference. If the tweet is a retweet, the *retweet-shortcut* (RT at the beginning of the message) and the *@username references* are deleted. The *@username mentions* at the beginning of the tweet (if the tweet is a reply to one or more users) are also removed. In addition, each hashtag term is cleared of the hash character #. Figure 4.2 shows a tweet before and after normalization.

2. **Segmentation into sentences.** After normalization, the text is divided into individual sentences. Although tweets usually contain very few sentences, this

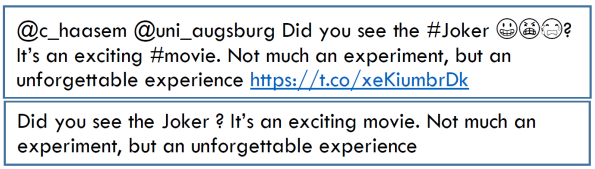


Figure 4.2: Tweet before and after normalization.

segmentation is still important because the POS Tagger determines the word types of the individual words based on the sentence structure.

3. **Segmentation into words.** In this step the sentences are segmented into individual words and passed on to part-of-speech tagger.

4. **Determination of POS tags.** Each input word is assigned to its part of speech with a part-of-speech tagger. Most such POS taggers use statistical methods to determine a part of speech. They are trained on the data where each word has already been tagged. The trained tagger then assigns a new input word the tag that has the highest probability in the context (cp. [LM11]). In Table 4.1, a sentence *"Gustav likes sunny days to go swimming"* is tagged with a Penn Treebank Tagset. This tagger for English language is also used in our implementation.

Gustav	likes	sunny	days	to	go	swimming
NNP	VBZ	JJ	NNS	TO	VB	VBG

Table 4.1: A tagged example sentence.

The tags determined by the tagger can be understood as follows: *NNP* is a proper noun in singular, *VBZ* is a verb in the present tense and in the third person singular, *JJ* stands for an adjective, *NNS* for a noun in the plural, *TO* is a word "to", *VB* corresponds to a verb in infinitive and *VBG* to a verb in the gerund.

5. **Filtering of nouns.** Penn Treebank Tagger distinguishes general nouns and proper nouns in singular and plural. We need all four groups, because, for example, both *"Trump"* and *"president"* can be used in the `CONTAINS` preference. In this step the filtered nouns are additionally cleared of unnecessary characters. *Points* and *apostrophes* remaining in words are removed, as they make stemming problematic and complicate comparision with the terms from the preference during evaluation.

6. **Stemming.** After cleaning the words from special characters, at first it is ensured that the word is in lower case and then a stemming is performed. Stemming shortens a word back to a word stem common to all morphed forms [SL18]. Stemming

should not be confused with *lemmatization*. These two processes reduce words to a certain basic form, but while lemmatization returns the word to its dictionary form, stemming often creates an unnatural common form. Lemmatizers are often used when the produced lemma has to be used in the area of word meaning or generally a natural real basic form is needed. I use stemming precisely because it takes no account of the meaning of the word. Since stemming create the unnatural common form anyway, it is possible to stem the misspelled words too.

The most common stemmer for the English language, which I also use in my work, was developed by M. Porter and published in [Por80] in 1980. A schematic representation of the form $[C](VC)^m[V]$, $m \geq 0$, where C is a consonant, V - a vowel, is calculated for each word. Then the word stem is calculated by changing or eliminating the current word ending in several steps according to the appropriate rules.

4.2.2 Edit Distance

The stemmed nouns from the tweets should now be compared with the nouns specified by the user in the `CONTAINS` preference. Note that the preference nouns are also stemmed by the same procedure to enable a fair comparison. Ideally, the word (stem) from a tweet is exactly the same as the word (stem) from a preference. But we know that tweets are rich with spelling mistakes. In order to allow comparison of the misspelled nouns, I calculate the distance between a stemmed input noun and a stemmed noun from the tweet. The terms with the distance below a certain value are regarded as equal.

For the comparison of words that include spelling mistakes there is a wide range of techniques, which are based on syntactic and statistical methods (cp. [Kuk92]). In this work I use the *Damerau-Levenshtein* algorithm that works over an *edit distance*. Edit distance between two words is defined as number of changes in one word that have to be made to get from that word to another one. Changes include *inserting* and *deleting* of a character, *replacing* of a character with another one, and *swapping* the positions of two characters that cover the most common spelling mistakes.

Damerau [Dam64] and Peterson [Pet86] independently found out that about 80% of all spelling mistakes belong to one of the four groups shown in Table 4.2.

The extended variant of Damerau-Levenshtein algorithm has a *threshold value*. The calculation is aborted if the distance between two words is too big and the threshold value is exceeded. Aborting the distance calculation is possible because I am not interested in the distance between two words, but in very similar words. This results in runtime advantages.

error	incorrectly	correctly
swap of two adjacent characters	huose	house
one letter false	dok	dog
one letter missing	aple	apple
one letter to many	informaition	information

Table 4.2: The four most common spelling mistakes [Dam64].

Example 13 Let us calculate the *edit distance* between two seemingly very different words *"housing"* and *"mouse"*. First, both words should be stemmed: *housing* → *hous* and *mouse* → *mous*. It is now possible to calculate how many changes it takes to turn stem *"hous"* into stem *"mous"*. In this case, only *one* change is needed, namely the *replacement* of the character *h* by *m*. As one can see, very different words can have a close *edit distance* between their stems.

I therefore need to define the term *similar words*. I need to determine if, for example, *"poll"* and *"pole"* - two different correctly spelled words - are similar. I have also to regulate if *"skater"* and *"skating"* - two different nouns with the same stem - are similar words in this approach. Finally, I need to determine whether misspelled words are similar with the correctly spelled ones, e.g., *"schedule"* and *"shedule"*. In addition, one should consider whether the degree of similarity is related to the length of the word. After all, a long word offers more room for possible errors.

I have decided to consider three possible scenarios:

1. The input word t from the `CONTAINS` preference is **short**: $|t| \leq 4$. This length was taken for several reasons: In [Kuk92] the author defines short words as word with a maximum length of 4 characters. Moreover, the distance is determined not for the original words, but for their stemmed variants, which in many cases are shorter. The allowed distance d between the short input word and the word from tweets to consider them as equal has to be $d = 0$, so they must be *identical*. If the distance is greater, one will identify the pairs *"cat"* and *"cap"* or *"poll"* and *"pole"* as identical, which is obviously a mistake.

2. For words of **medium** length the distance of $d = 1$ is allowed. Medium-length words include words with more than 4 but less than 8 characters: $4 < |t| < 8$. This decision is supported by the work of Norvig [Nor13]. He analyzed the number and length of the words that appear in books scanned by Google Books. He found that 80% of written words are between 2 and 7 characters long and the average length is about 4.7 characters per word.

   Distance 1 means that *one change* is enough to make the input word and tweet word exactly equal. The possible changes correspond to the 4 most common spelling

mistakes listed in Table 4.2. In the words of medium length (note that the length is always determined for a stemmed word) the probability of making a mistake is much higher than in the short words. At the same time, it is rather improbable to form an existing correct word by making only one change: inserting, deleting, replacing of a character or swapping the two characters positions.

The described approach will define equality for a pair of words *"schedule"* and *"shedule"*. The *correctly* spelled word *schedule* will be reduced by he stemmer to the form *schedul* with the length of 7 characters. The *incorrectly* spelled word *shedule* will be stemmed to *shedul*. The edit distance between these stems is $d = 1$ (inserting of c into *shedul*), hence the words are *"equal"* and the tweets containing *"shedule"* will also belong to the result set, if the preferred term in `CONTAINS` is *"schedule"*.

3. The last scenario applies to **long** words with at least 8 ($|t| \geq 8$) characters after stemming. This group is relatively small: According to Norvig (cp. [Nor13]), only 20% of all words reach the length of 8 characters or more. But he examined the words in the form they appeared in the books. Stemming usually shortens the words, so the percentage for this group is clearly below 20%. This is also supported by the fact that tweets are limited in length to 280 characters and users try to express their thoughts briefly and clearly. The short and precise words are a part of it.

   For the small part of the long words I allow the distance of $d = 2$. It is not difficult to make a mistake in a long (and often complicated) word. Especially because tweets are usually written casually, parallel to another activity. Because of the generally low number of long words, the danger of finding two of them within the distance of $d = 2$ is also low.

4.3 Experiments

In this section I describe my test results. These tests should give an impression of how *efficient* the developed approach is (runtime) and what *quality* the delivered results have (do the results correspond to the query and whether something remains undiscovered).

For comparison another method (further called *simpleC*) of score calculation for `CONTAINS` preference was implemented. For this, there is no complex natural language tweet preprocessing. I only remove the initial signature of retweets *"RT @name"*, the rest of the text remains unchanged. I use a pre-implemented case insensitive func-

tion *contains* from the Apache Commons Lang 3.9[21] library that checks if one term is contained in another one. For each term in each of preference sets it is checked whether it occurs in the tweet and if this is the case, the number of the term's level is assigned to the tweet. The total tweet's score is then a *minimum* of all assigned level numbers (cp. Definition 4).

Although I do not preprocess tweets using *simpleC*, stemming for the preference terms is done anyway. Thus *prefC* (my approach with complex preprocessing and spelling correction) compares stemmed preference terms with the stemmed tweet's nouns chosen while preprocessing of the text. *simpleC*, which is used as a reference point, compares the stemmed preference terms to all original written words (nouns or no) from the tweet's text. The reason for this decision is due to the way the plural of some nouns are formed in English. The word ending *-y* in singular changes into *-ies* in plural, e.g., *study* and *studies*. As a consequence, when searching for the word *study* in the tweet text, *studies* will not be considered as a hit and vice versa. Spelling mistakes are not taken into account in this method and only the correctly spelled words are regarded as hits.

4.3.1 Quality Tests

To test the quality of the returned results, 10.000 completely random tweet objects were saved to a text file. As input terms, I used both words that changed their form after stemming and those that remained unchanged. In addition, I selected words of different lengths to be able to test whether wrong spelled words will be found (cp. Section 4.2.2).

The two methods are applied to the tweets that I have stored. Table 4.3 shows the *number* of hits (tweets containing the corresponding term). In Table 4.4, I see the same results expressed in terms of ***Precision (P)*** - the percentage of retrieved documents that are relevant to the query - and ***Recall (R)*** - the percentage of the relevant documents that are successfully retrieved.

One can see that the number of hits in Table 4.3 for the *simpleC* is for most terms higher than that of the *prefC*. There is only one exception for the term *skating*, which I explain later. *simpleC* returns on average more tweets. Moreover, *recall* of *simpleC* is almost everywhere 100%, i.e. *all relevant tweets* have been found. The only exception I observe for the term *connection*. One of the relevant tweets includes this misspelled term (*konnection*) and will not be found by *simpleC*, but will be present in the results of *prefC*. *prefC* shows slightly worse recall results, but the difference doesn't exceed 15%, except for one term - *sandwich*. *simpleC* returns twice as many relevant tweets.

[21] Apache Commons Lang 3.9 API:
commons.apache.org/proper/commons-lang/javadocs/api-3.9/overview-summary.html

term	stemmed term	prefC	simpleC
sandwich	sandwich	1	2
housing	hous	35	93
skating	skate	25	1
decision	decis	7	7
connection	connect	8	20
replacement	replac	10	15
development	develop	13	19
independence	independ	6	8
forest	forest	6	7
bear	bear	6	20

Table 4.3: Number of hits using two different methods.

term	***P*** prefC	***P*** simpleC	***R*** prefC	***R*** simpleC
sandwich	100%	100%	50%	100%
housing	100%	39%	97%	100%
skating	4%	50%	100%	100%
decision	100%	100%	100%	100%
connection	87.5%	40%	78%	89%
replacement	40%	27%	100%	100%
development	100%	74%	93%	100%
independence	50%	37.5%	100%	100%
forest	100%	100%	85.7%	100%
bear	100%	30%	100%	100%

Table 4.4: Precision (P) and Recall (R) for two different methods.

But if you look at Table 4.3 with the number of hits, you can see that twice as many means 2 tweets, instead of 1, which *simpleC* finds.

The reason for this is that my approach takes into account only a very small number of *hashtags*. They can be almost any - complex, non-existent, invented: *#BeginnersMind*, *#BuildingConnections*, *#ManagersDay2016* are only some examples. Obviously, that stem *sandwich* is found in the hashtag *#tunasandwich* by *simpleC* but not by *prefC*, which can not split this complex term into its components. This is also the case for stem *forest* (cp. Table 4.3): the seventh hit of *simpleC* is a tweet with the hashtag *#UgandasForests*, which remains undetected by *prefC*.

Besides the "good" results, *simpleC* also detects the "bad" ones in this way: The tweets with hashtags *#clubhousegh* and *#DaquansPlayHouse* are e.g. returned when the user is looking for *housing*. Let's take a closer look at this term. Since the length of the stemmed term is $|hous| = 4$, no different spelling or mistakes are allowed. The tweets delivered by *prefC* (my approach) contain the words: *house*, *housing*, *House*, *house's*, which all make sense. In the tweets delivered with *simpleC*, one finds other terms besides those already mentioned, e.g. ***hous**wife*, *pent**house***, *t**hous**and*. Also the proper names (Whitney ***Hous**ton*, Amy *Wine**house***), hashtags (*#uk**hous**ing*, *#club**hous**egh*, *#DaquansPlay**House***) and account names (*@RachaelL**Hous***, *@Alert-**Hous**ton*, *@6Vis**house*** etc.) are included. The result is understandable, because all listed words contain stem *hous*. But it is not what I wanted.

Another reason for the larger result of *simpleC* are *verbs*, *adjectives* or even other *nouns* with the corresponding stem (e.g. to ***connect*** and ***connected*** while searching for ***connection***; ***beary***, to ***bear*** or ***beard*** - for ***bear***).

One should not forget words whose meaning changes to the opposite just by adding a prefix too: *dependence* and *independence*, *spelling* and *misspelling*. While looking for the first term of each pair, *simpleC* will find also the second one. And since their meanings are completely opposite, the presence of the latter terms (with prefix) in the result set is undesirable if the user wants the former ones.

The returned non-relevant results have a great effect on the *precision* values, which one can see in Table 4.4 for the *simpleC* very well. Precision values of *simpleC* are almost always worse, sometimes very clearly than those of *prefC*. The only exception is the term *skating*. This is the only term for which the number of hits of *prefC* is higher and precision is lower than for *simpleC* (see Tables 4.3 and 4.4). If I look more closely at the resulted tweets, I do not see in the texts *skating*, *skate*, *skater*, etc. that were expected. Instead one finds *state* and/or *Kate*. The reason is the allowed deviation of the correct spelling / error correction. The original term *skating* is stemmed to *skate*. It has a length of 5, so there is a distance of 1 between *skate* and the stemmed nouns from the tweet allowed (cp. Section 4.2.2). *Skate* and *state* have one different letter, *skate* has one letter more than *Kate* (the approach is case insensitive), so the result is quite logical.

My approach has also found a tweet with a misspelled term *konnection* (instead of *connection*). Unfortunately, the number of incorrectly returned tweets is too high with this correction. So I clearly need to revise this step of my approach. One can increase the length of the words that are allowed to be corrected. The implementation would be very simple, but one cannot guarantee that even with the longer words, a very similar but completely different term will not appear at the distance of one change.

Another option is to create some kind of lexicon where the most common (spelling) variations of the words are collected. A big advantage of this approach is that I do not have to limit ourself to 4 most common spelling mistakes (see Table 4.2), but can also include the very popular abbreviations (e.g. *prof* for *professor*). A big disadvantage in return is the effort required to create this lexicon.

In general, one can say that the quality of results delivered with *prefC* is higher compared to *simpleC*. But spelling mistake correction / different spelling part must be enhance.

4.3.2 Runtime Tests

I also did some runtime tests: I compared the time *prefC* and *simpleC* take to evaluate *one* tweet. The tests were performed on Intel® Xeon® Scalable Processor "Skylake"

Silver 4110 with 2.10 GHz, 192GB DDR4 and 2x 4TB SATA3-HDD running Ubuntu Linux 18.04 LTS 64 bit.

For the tests I collected real tweets. Each file contains a different number of tweets (cp. Table 4.5). The tweets were collected live and are different in each file. All tweets in each file were evaluated with respect to two preferences:

- P_1:=CONTAINS$_{m=2}$(t, {'bear'},{'forest'}, others)
- P_2:=CONTAINS$_{m=3}$(t, {'housing'},{'decision', 'statement'},{'connection'}, others)

using *prefC* and *simpleC*. The results can be found in Table 4.5 for P_1 and Table 4.6 for P_2. The measured time includes both the tweets *preprocessing* and the actual *score calculation*.

file	tweets number	time prefC	time simpleC
f1	120 277	374.09	108.07
f2	133 278	446.79	116.8
f3	268 231	438.76	115.39
f4	317 816	443.53	116.13
f5	399 929	369.35	106.42
f6	599 275	365.1	105.3
f7	931 027	431.88	113.99
f8	1 160 599	435.99	114.41
f9	1 307 764	362.31	104.46
f10	3 592 899	348.25	104.3

Table 4.5: Runtime in microseconds (μs) of two different methods per tweet for P_1.

file	time prefC	time simpleC
f1	374.4	110.65
f2	449.62	119.62
f3	441.18	118.32
f4	447.05	118.79
f5	370.8	109.17
f6	365.71	107.85
f7	434.53	116.06
f8	438.24	117.31
f9	362.58	106.48
f10	349.26	105.66

Table 4.6: Runtime in microseconds (μs) of two different methods per tweet for P_2.

As one can see, the runtime of *simpleC* is always significantly better compared to *prefC*. But let us not forget that my approach (*prefC*) requires much more complex preprocessing and score calculation. The other thing that stands out in the results is that regardless of a preference, the runtime for both methods remains very stable and do not depend on the file size (number of tweets).

It seemed interesting to us that the results can be clearly divided into two groups: the files *f1*, *f5*, *f6*, *f9* and *f10* belong to *Group* 1 with a *lower* runtime per tweet,

the files *f2*, *f3*, *f4*, *f7* and *f8* belong to *Group* 2 with the *higher* runtime. And this for both preferences and both methods. The only reasonable explanation is that the tweets in the first group are on average shorter. This means that fewer comparisons are required (no matter which method is used), which shortens the runtime.

In Table 4.7 I have summarized the runtime for complete tweet evaluation (preprocessing and score calculation) for both preferences P_1 and P_2 and the runtime for separate score calculation. Score calculation is about 60-65% of the total runtime.

file	**full** P_1	**getScore** P_1	**full** P_2	**getScore** P_2
f1	374.09	235.3	374.4	235.3
f2	446.79	292.59	449.62	295.19
f3	438.76	288.5	441.18	292.2
f4	443.53	295.57	447.05	295.26
f5	369.35	233.22	370.8	235.59
f6	365.1	231.51	365.71	233.83
f7	431.88	284.79	434.53	286.8
f8	435.99	288.1	438.24	289.83
f9	362.31	229.64	362.58	230.72
f10	348.25	220.82	349.26	222.97

Table 4.7: Full and getScore runtime in in microseconds (μs) for P_1 and P_2.

In this chapter I have described tweet text processing for preference-based search on tweets. A new `CONTAINS` preference was developed and implemented. It allows the user to list the terms that he would like to see in the tweets. A certain deviation in spelling or form, as well as possible errors should be taken into account as much as possible, because spelling mistakes are very typical for tweets. The texts from the tweets are preprocessed using some methods from Natural Language Processing. Then, both the input terms and the terms from the tweet are stemmed. Finally, the build stems are compared, and if they are equal (which is defined differently for terms of different lengths), the tweet belongs to the result set.

The first experiments have shown that this approach gives very good *precision* results in most cases. The misrecognized tweets are the result of the misspelling correction. This aspect clearly needs to be revised. *Recall* numbers are fine, but a few steps can be taken to improve them. For example, the *hashtags*, require special treatment, since in most cases they are not real words existing in the language and therefore cannot be treated like normal nouns. Although I already had some thoughts on improving the runtime while implementing this approach, the experiments have shown that there is still a lot of work to be done in this direction.

Chapter 5

Preference Algorithms on Data Streams

This chapter deals with a new algorithm for evaluating Pareto queries on data streams.

Stream query processing and especially preference query processing require modified algorithms, since there is no end result, the new data arrive again and again. I start with the description of the general *problematic of preference query computation on data streams* in Section 5.1. Also I describe here how a Block-Nested Loop algorithm can be changed and adapted to calculate the correct result on the stream data and what its weaknesses are. The next Section 5.2 is dedicated to the *new Stream Lattice Skyline Algorithm* (SLS) for efficient Pareto computation on unbounded data. I describe the general idea and then go into the details using pseudo code description of my SLS algorithm. In the last Section 5.3 I describe very detailed *experiments*, which were made with the new algorithm.

Today, data processed by humans as well as computers is very large, rapidly increasing and often in form of data streams. Many modern applications such as network monitoring, financial analysis, infrastructure manufacturing, sensor networks, meteorological observations, or social networks require query processing over data streams, e.g., [CDTW00, BGS01, ABB+03, SST+09].

Users want to analyze this data to extract personalized and customized information in order to learn from this ever-growing amount of data, e.g., [SKE11, GR09, KEW11, DPE08]. Due to the continuous and potentially unlimited character of stream data, it needs to be processed sequentially and incrementally. However, queries on streams run continuously over a period of time and return different results as new data arrive. Although many approaches exist for effective processing of data streams, learning from streams requires new algorithms and methods to be able to learn under the evolving and unbounded data.

On the other hand, preference queries [EP17, MKEK15, Kie02, Cho03, KEW11, REMK12, REH+12] have received considerable attention in the past, due to their use in selecting the most preferred items, especially when the filtering criteria are contradictory. In particular, *Pareto preference queries*, also known as *Skyline queries* [BKS01, CGGL03, CCM13], a subset of preference queries, have been thoroughly studied by the database community to filter high relevant information from a dataset. A *Pareto preference query* selects those objects from a dataset D that are not dominated by any others. An object p having d attributes (dimensions) dominates an object q, if p is better than q in at least one dimension and not worse than q in all other dimensions, for a defined comparison function. This dominance criterion defines a partial order and therefore transitivity holds. The Pareto-optimal set is the set of *best matches only (BMO)* objects, which are not dominated by any other objects of D (cp. Definition 17).

Algorithms proposed for traditional database Pareto computation, e.g., [ER18, BKS01, GSG05, CCM13], are not appropriate for continuous data and therefore new techniques should be developed to fulfil the requirements posed by the data stream model.

The most important property of data streams is that new objects are continuously appended, and therefore, efficient storage and processing techniques are required to cope with high update rates. Hence, a stream-oriented algorithm should satisfy the following requirements (cp. [KPM10]): 1) fast response time, 2) incremental evaluation, 3) limited number of data access, and 4) in memory storage to avoid expensive disk accesses.

In this chapter, I present a novel algorithm called SLS for evaluating Pareto queries over continuous settings, and empirically demonstrate the advantage of this algorithm on artificial and real data. My algorithm continuously monitors the incoming data and therefore is able to maintain the BMO-set incrementally. It is based on the lattice

structure representing the better-than relationships that must be built only once for efficient Pareto computation on continuous data. The proposed algorithm satisfies all four requirements on stream-oriented algorithms as mentioned before.

5.1 Pareto Queries on Data Streams

Preference query processing on data streams require modified algorithms, since a stream is a continuous dataflow and there is no "final" result after some data of the stream is processed. The result must be calculated and adjusted as soon as new data arrive, since new stream objects received later can build a new BMO-set compared with objects already recognized in previously computed (temporary) BMO-set (cp. Section 3.3.

Pareto queries are a special case of preference queries (cp. Definition 13). These queries are very popular by users because they allow them to specify desired values for several attributes at once. To the best of my knowledge, only *Block-Nested-Loop* (BNL) style algorithms (cp. [BKS01, XYWH05, JZMG15, MPG07] or [TP06]) can be adapted to Pareto evaluation on continuous data. These algorithms follow an *object-to-object* comparison approach, an expensive operation with a worst-case runtime of $O(n^2)$, where n is the number of objects.

For analysis of a continuous, unbounded data stream, it is necessary to divide it into a series of (non-overlapping) chunks $c_1, c_2, \ldots$, as explained in Section 3.2.2. A BNL-style algorithm would evaluate the BMO-set on the first chunk. Since c_2 could contain better objects w.r.t., one also has to compare the new objects from c_2 to the current BMO-set, i.e., compute and so on (cp. Equation 3.1). However, this leads to a computational overhead if c_2 is large.

Let us consider the following example:

Example 14 A user wants to find the tweets with *hashtag* #isu or alternatively with hashtags #yog or #figureskating. He is also more interested in tweets that were posted by authors with the total number of posts (*statuses_count*) between 3000 and 6000, the deviation of 500 tweets does not matter. Both attributes - *hashtag* and *statuses_count* - are equally important for the user.

The evaluation of this preference with BNL on the data from Table 3.1 is schematically represented in Figure 5.1.

In the first step *tweet1* from *chunk 1* is added to the current BMO-set. In the second step *tweet2* is compared to *tweet1*. They are *indifferent* and both are in the temporary BMO-set. Note that indifference is given because d-parameter is set to 500. This means that the values of the attribute *statuses_count* for *tweet1* (1723) and *tweet2*

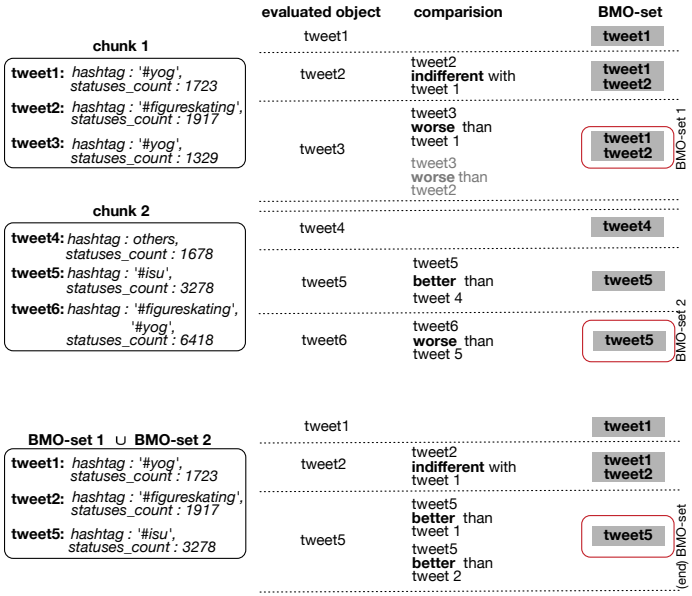


Figure 5.1: Example of stream data evaluation with BNL.

(1917) end up in the same level (cp. Definition 7). The next candidate from *chunk 1* is *tweet3*, it has be compared to the objects from the current BMO-set. It is *worse* than *tweet1*, the comparison to *tweet2* is not necessary in this case, *tweet3* it will be discarded. To the current temporary BMO-set belong still *tweet1* and *tweet2*. *Tweet3* was the last one in the *chunk 1*. The algorithm continues with the next *chunk 2* and calculates the BMO-set for this new chunk.

First *tweet4* is added to the current BMO-set. Afterwards *tweet5* is compared to *tweet4*. It belongs to the new current temporary BMO-set for the *chunk 2* because it is *better* than *tweet4*. The last tweet in this chunk *tweet6* is *worse* than *tweet5* and is discarded after comparison to it. *Tweet5* is the only tweet in the BMO-set of the *chunk 2*.

Now I have two partially BMO-sets for two chunks. In the next step the total BMO-set can be calculated according to Equation 3.1. *Tweet1* belongs to the BMO-set in the first step, following by *tweet2* in the second step, which is *indifferent* compared to *tweet1*. An finally *tweet5* has to be compared to the tweets in the current BMO-set. It is *better* than *tweet1*, but it is not enough to add *tweet5* to the BMO-set. It will

be compared also to other objects in the temporary result set. Since *tweet5* is also *better* that *tweet2*, it is the only one object in the end BMO-set of two chunks.

The result is *correct*, however, it is very inefficient. A large number of comparison leads to a quadratic worst-case run time. To make the response time better, I developed and implemented an algorithm that can evaluate queries in linear time. I was inspired by the idea described in [MPJ07, PK07, EK14] and wanted to adapt this algorithm to stream data. That is how ***Stream Lattice Skyline*** algorithm came about.

5.2 Stream Lattice Skyline Algorithm (SLS)

I have developed *Stream Lattice Skyline* (SLS) algorithm for efficient real-time Pareto (Skyline) computation on unbounded streams [RE18]. It does not depend on object comparisons as BNL algorithms do (cp. Section 5.1), but on the lattice structure constructed by a Pareto query over low-cardinality domains (either inherently small, such as language of a user on Twitter – or mapped to low-cardinality domains, such as number of followers in Twitter).

5.2.1 Concept of SLS

Before I describe my SLS algorithm for real-time stream processing, I revisit the basics of *Hexagon* [PK07] and *Lattice Skyline* [MPJ07] this algorithm is based on.

A Pareto (Skyline) query over discrete domains constructs a *lattice*, a structure where two objects o_1 and o_2 of a dataset have a least upper bound and a greatest lower bound. Visualization of such lattices is often done using *Better-Than-Graphs* (BTG) (see Section 2.1.1). An example of a BTG is shown in Figure 5.2.

The nodes in the BTG represent *equivalence classes*. The bold numbers next to the nodes are the *unique IDs* (see Theorem 3), which play an important role in the implementation of the algorithm. The idea is to map objects from the stream to these equivalence classes using some kind of feature function. All values in the same class are considered *substitutable*.

I write [2,4] to describe a two-dimensional domain as well as the maximal possible values of the feature vector representing objects. For example, the BTG in Figure 5.2 could present a Pareto on the tweets *language* of a user ({english, german, unknown}) (values 0, 1, and 2) and the *hashtag* which might be an element of {#yog, #lausanne2020, #figurescating, #isu, #others} (values $0, \ldots, 4$). The arrows in the BTG show dominance relationships between nodes. The node $(0, 0)$ presents the *best node*, whereas $(2, 4)$ is the *worst node*. The bold numbers next to each node are *unique identifiers* (ID) for each node in the BTG. Nodes having the same level are

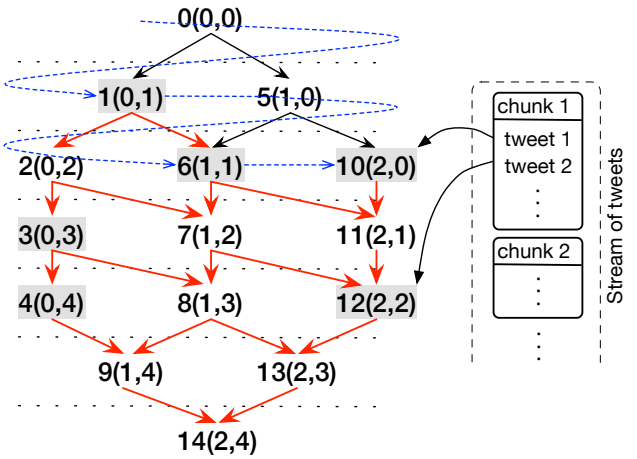


Figure 5.2: Stream data processing with SLS.

indifferent. That means for example, that neither the objects in the node $(0,4)$ are better than the objects in $(2,2)$ nor vice versa. They have the same overall level 4. A dataset does not necessarily contain representatives for each BTG node. All gray nodes are occupied with an element of the data and therefore *non-empty*, white nodes are *empty*.

The elements of the data that compose the BMO-set are those in the BTG that have *no path leading to them from another non-empty node*. In Figure 5.2 these are the nodes (0,1) and (2,0). All other nodes have direct or transitive edges from these both nodes, and therefore are dominated.

My approach in general works as follows: after constructing the BTG, which only must be done once, all tweets of a chunk are mapped to the corresponding nodes in a consecutive way, e.g., *tweet1* is mapped to $(2,0)$, *tweet2* to $(2,2)$, and so on. Assume all gray nodes in Figure 5.2 are occupied with data from the first chunk. Afterwards, a *breadth-first traversal* (BFT) runs to find the non-empty nodes (blue dashed line in Figure 5.2). For the first non-empty node (here $(0,1)$) I start a *depth-first traversal* (DFT) (red arrows) to mark all transitive dominated nodes as *dominated*. If the DFT reaches the bottom node $(2,4)$ (or an already dominated node) it will recursively follow all other edges. Thereafter, the BFT continues with node $(1,0)$, which is empty. The next non-empty node is $(1,1)$, but dominated. Continue with $(2,0)$. Since all other nodes are marked as dominated, the remaining nodes, $(0,1)$ and $(2,0)$, present the BMO-set.

Now consider the next chunk (chunk 2). Read all objects form chunk 2, add them to the lattice and perform a BFT and DFT. Note, that no new objects will be added to the nodes marked as "dominated" during processing of previous chunks, because they can not be part of the BMO-set. After BFT and DFT, the remaining nodes (maybe including the objects from the previous computation) contain the temporary best tweets. Continue with chunk 3, etc.

Note that lattice-based algorithms are developed for Pareto computation over low-cardinality domains only. An attribute domain $dom(S)$ is said to be low-cardinality if its value is drawn from a set $S = s_1, ..., s_m$, such that the set cardinality m is small. For a low-cardinality domain and $s_i \in \mathbb{R}$, a one-to-one mapping function $f : dom(\mathbb{R}) \rightarrow \mathbb{N}_0$, $f(s_i) = i - 1$, can be defined to get discrete values as required in our algorithm.

However, since there are many attributes with high-cardinality domains (e.g., Twitter account of Katy Parry has about 100 millions *followers*), they can be mapped to a small range of numbers. This can be done by a function like: $f : dom(\mathbb{R}) \rightarrow \mathbb{N}_0$, $f(s_i) = \lfloor i/d \rfloor$, $d > 0$, where d represents a discretization of $f(s_i)$ and can be specified by the user (cp. d-parameter in Definition 6).

5.2.2 The SLS Algorithm

SLS is based on a series of finite chunks as described in Section 3.3. The algorithm itself is divided into three phases:

1. The **Construction Phase** (see pseudo code Phase 1) *initializes the BTG which depends on the Pareto query (see [PK07, MPJ07] for details). This has to be done only once for the first chunk (line 1). For evaluation of the following chunks, the existing BTG will be reused.*

   The BTG is represented by an array (line 2) of NODES *in main memory. The mapping from a BTG to an array is possible through the node IDs (see Theorem 3), which are unique and allow linear storage. The size of the array equals the number of nodes in the BTG (see Theorem 2). A* NODE *is a data structure representing an equivalence class in the BTG, which may contain objects from the stream in main memory having the size of the lattice, i.e., the number of nodes. Each position in the array stands for one node in the lattice.* NODES *are identified by their IDs, which correspond to their position in the BTG array. A* NODE *also contains its status* empty *(initial status),* non-empty*, or* dominated.

Phase 1 Construction Phase

```
1: if algorithmFirstRun then
2:     BTG := array of empty Nodes (equivalence classes)
3: end if
```

2. In the **Adding Phase** (see pseudo code Phase 2) I process the input data:

   a. *Read the next chunk c_i from the data stream S.*

   b. *Iterate through the objects o_j of chunk c_i (line 3). Each object will be mapped to one node in the BTG. For this I compute the* ID *of the current object o_j (line 4) and store it in the BTG array, if the* NODE *is not dominated (line 5 and 6).*

Phase 2 Adding Phase

```
1: Input: Next chunk c_i from data stream S
2: ▷ Add objects o_j from chunk c_i to the BTG
3: for o_j in c_i do
4:     ID := compute ID for o_j
5:     if ¬BTG[ID].isDominated() then //NODE is not dominated
6:         BTG[ID].add(o_j)
7:         BTG[ID].setStatus = non-empty
8:     end if
9: end for
```

3. **Removal Phase** (see pseudo code Phase 3): *After all objects in the chunk have been processed, the nodes of the BTG that are marked as* non-empty *and are not reachable by the transitive dominance relationship from any other* non-empty *node of the BTG represent the (temporary) pareto-optimal result set. From an algorithmic point of view this is done by a combination of* breadth-first traversal (BFT) *and* depth-first traversal *(DFT).*

   I start a BFT at the top of BTG (line 1) and search for the first non-empty *node, which is not dominated. From this node on, I start a DFT to mark dominated nodes (line 4 and procedure DFT in line 9) recursively. When processing objects from the next chunk, this ensures that there is no need to add objects to already dominated nodes in the BTG. This reduces memory requirements and enhances performance. After processing all nodes in the DFT, I continue with the BFT until all nodes are visited. The remaining nodes contain the temporary Pareto-optimal set and can be presented to the user.*

Phase 3 Removal Phase

```
1: Start a BFT beginning at the top of the BTG
2: for each ID in BFT do
3:     if ¬BTG[ID].isDominated() ∧ ¬BTG[ID].isEmpty() then
4:         DFT(ID) //Start depth-first traversal to mark dominated nodes
5:     end if
6: end for
7:
8: ▷ Start DFT
9: DFT(ID) =
10: for each successor sID of ID do
11:     if BTG[sID].isDominated() then return //Node is already dominated
12:     end if
13:     ▷ Otherwise remove dominated nodes and continue DFT
14:     BTG[sID].setStatus = dominated
15:     BTG[sID].clear() //Set objects to null
16:     DFT(sID) //Recursion, continue with successor of sID
17: end for
```

4. Since there is a continuous data stream, Phase 2 and Phase 3 have to be repeated for the next chunks.

The Pareto-optimal set computation in Phase 3 can be done after an arbitrary number of processed chunks or after a pre-defined time. Therefore, my algorithm can be used for real-time Pareto evaluation. It is also possible to parallelize this approach in the sense of [EK14, EK16]: parallelize the adding of the objects in the chunk, and, after adding an object, directly start a DFT to mark nodes as dominated.

Since SLS follows the idea of Hexagon and Lattice Skyline, the linear runtime complexity of $\mathcal{O}(dn + dV)$ remains for this algorithm. Thereby, n is the number of input objects, d number of dimensions, and V the size of the lattice, i.e., the product of the cardinalities of the d low-cardinality domains from which the attributes are drawn. Note that although a data stream has potentially infinite number of input objects, SLS algorithm gets the data chunk by chunk. And the chunks are finite, so the complexity of the algorithm can still be estimated.

5.3 Experiments

In this section I present comprehensive experiments on my SLS algorithm.

5.3.1 Benchmark Framework

In these benchmarks I wanted to explore the behavior of SLS on synthetic and real-world data, depending on the data size, chunk size, and different domain size. For runtime evaluation I compared SLS to the stream variant of BNL, cp. Section 5.1, because to the best of my knowledge, this is the only other stream-based Skyline algorithm.

For the experiments on artificial data I generated correlated (corr), anti-correlated (anti), and independent (ind) data streams as described in [BKS01] and varied three parameters: (1) the data cardinality (n), (2) the data dimensionality (d), and (3) the number of distinct values for each attribute domain.

For real data, I used Twitter records collected over a specific period of time. These objects (tweets) include various attributes such as *name*, *description*, *created_at*, *followers_count*, *status_count*, *lang*, and many more. I mapped all attribute values of theses short messages to a numerical domain according to a mapping function described in Section 5.2.1.

For analysis of a data stream, it is necessary to divide it into a series of chunks c_i as described in Section 3.2. For this I used Apache Flink, an open source platform for scalable stream and batch data processing, which is also able to process real-world data like Twitter (cp. Section 2.3).

My algorithms have been implemented using Java 8. All experiments are performed on a single node running Debian Linux 7.1. The machine is equipped with two Intel Xeon 2.53 GHz quad-core processors.

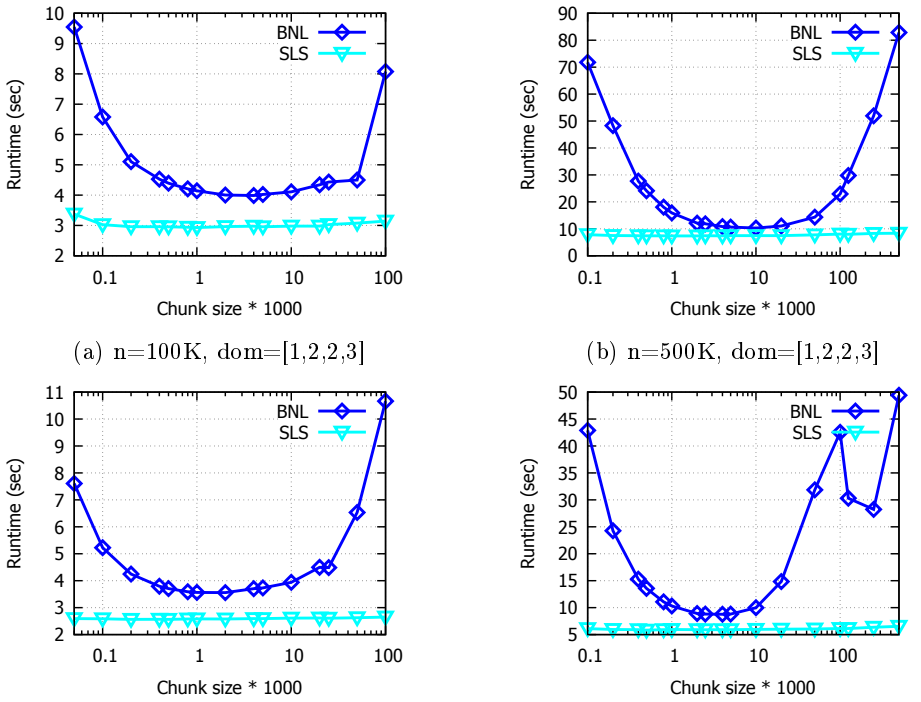
5.3.2 Influence of the Chunk Size

In the first experiment I varied the chunk size to find out the optimal number of objects per chunk. I also compared BNL to SLS w.r.t. their runtime depending on the chunk size. I used datasets with $100K$ and $500K$ objects and considered the algorithms behaviour for anti-correlated, independent and correlated data distribution.

For a more reliable result I considered different domains: $[1, 2, 2, 3]$, $[1, 5, 10]$, $[1, 2, 2, 2, 2, 2, 2, 2, 3]$, $[2, 3, 7, 8, 4, 10]$, $[1, 5, 2560]$ and $[13, 35, 70]$. Remember, each number corresponds to the maximal possible values of the single domains.

Figure 5.3 (pages 75 – 77) shows the results for *anti-correlated* data. In all experi-

ments SLS performs significant better than BNL independent of the chunk size. For small chunks (up to 500 objects) and for very large chunks (more than $50K$ objects), BNL is substantial worse than SLS. For small chunk sizes this can be explained by a higher number of unions which has to be carried out after each chunk evaluation, cp. Equation 3.1. For larger chunks the object comparison process takes more time.



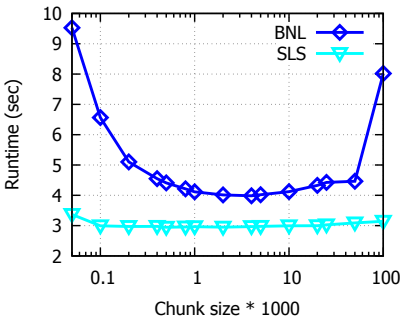
(a) n=100K, dom=[1,2,2,3]

(b) n=500K, dom=[1,2,2,3]

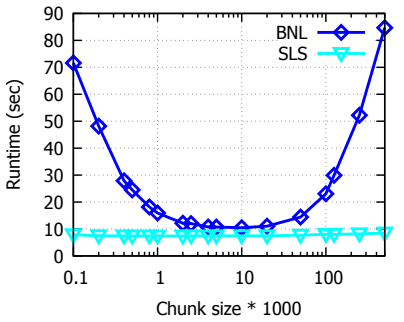
(c) n=100K, dom=[1,5,10]

(d) n=500K, dom=[1,5,10]

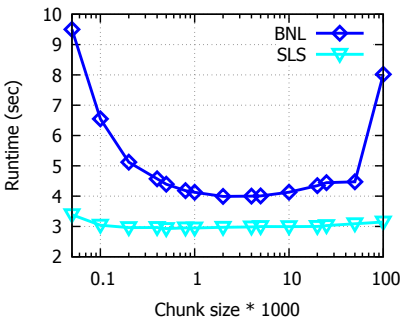
Figure 5.3: Influence of the chunk size. SLS vs BNL, anti-correlated data distribution.



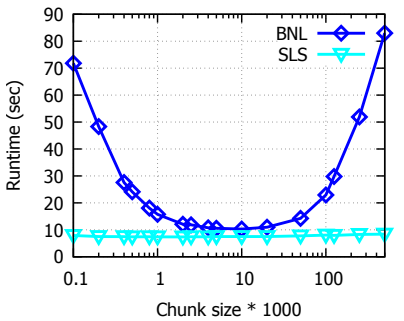
(e) n=100K, dom=[1,2,2,2,2,2,2,2,3]



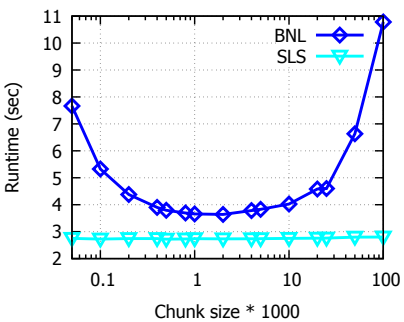
(f) n=500K, dom=[1,2,2,2,2,2,2,2,3]



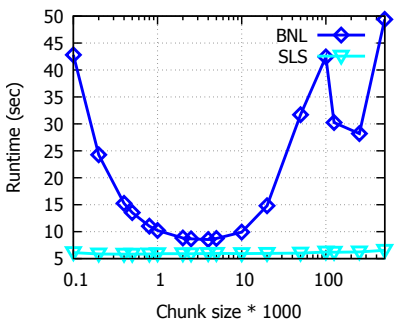
(g) n=100K, dom=[2,3,7,8,4,10]



(h) n=500K, dom=[2,3,7,8,4,10]

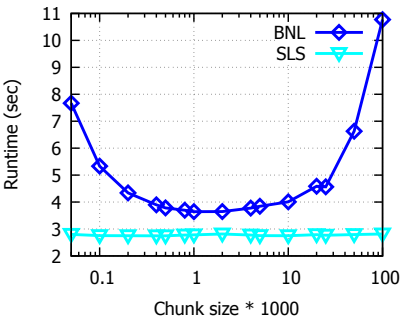


(i) n=100K, dom=[1,5,2560]

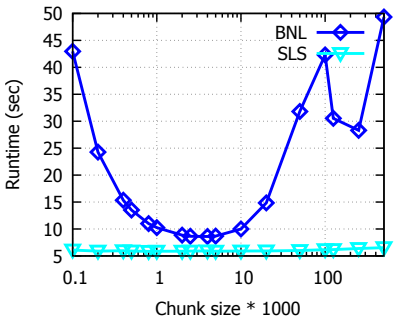


(j) n=500K, dom=[1,5,2560]

Figure 5.3: Influence of the chunk size. SLS vs BNL, anti-correlated data distribution.



(k) n=100K, dom=[13,35,70]

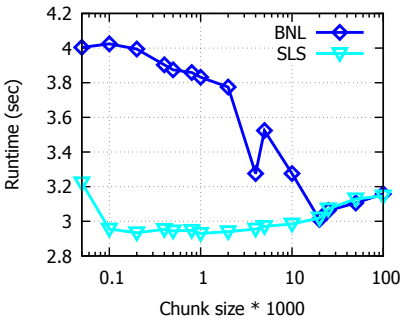


(l) n=500K, dom=[13,35,70]

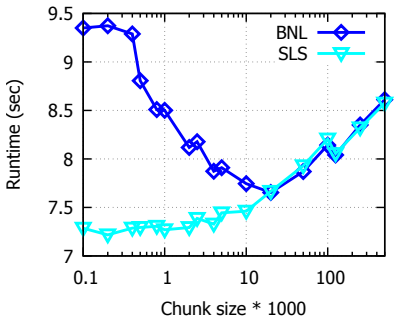
Figure 5.3: Influence of the chunk size. SLS vs BNL, anti-correlated data distribution.

SLS seems to be nearly constant w.r.t. the runtime. However, a closer look to the runtimes of SLS spawns that SLS is slower for small chunks (up to 200 objects) than for chunks with more than 200 objects, cp. **Figures 5.6a** to **5.6d** on pages 82 – 83. This can be explained by the frequent repeating of the BFT and DFT in SLS, which have to be carried out for each chunk. For the chunk size over $20K$ objects the runtime of SLS increases again, because the adding of new objects to the BTG (Phase 2) in SLS is more expensive. In summary, one claim that the optimal chunk size for the best runtime with *anti-correlated* data distribution is between 200 and $20K$ objects.

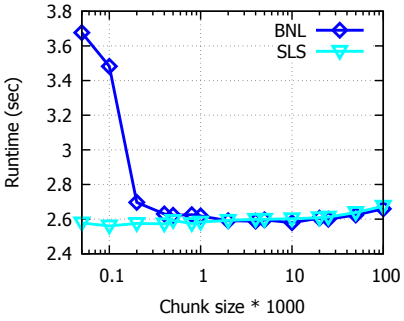
Figure 5.4 (pages 78 – 79) presents the comparison result of SLS and BNL for *independent* data distribution. One can see, that SLS has better runtime for small and medium chunks (up to $10K-20K$ depending on the domains), but for large chunks both algorithms have very similar runtime. For some domains, e.g. [1,5,2560] and [13,35,70] (cp. **Figures 5.4i** to **5.4l**) BNL outperforms SLS. In the case of these two domains we are dealing with the high cardinality domains, which produce *deep* BTGs in the sense of the *height*. The required time for the depth search (DFT) in the deep BTG (cp. Removing Phase 3 in Section 5.2.2) for such domains is significant longer than for low cardinality domains. This enables better runtime of BNL compared to SLS for chunk sizes from $1K$ objects for domains with high cardinality.



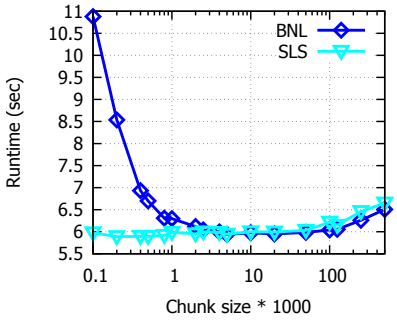
(a) n=100K, dom=[1,2,2,3]



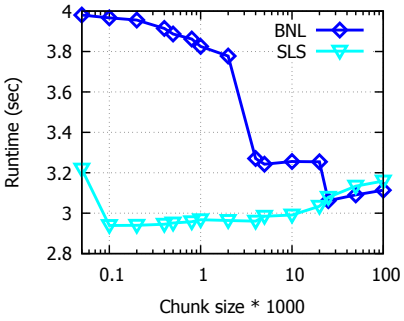
(b) n=500K, dom=[1,2,2,3]



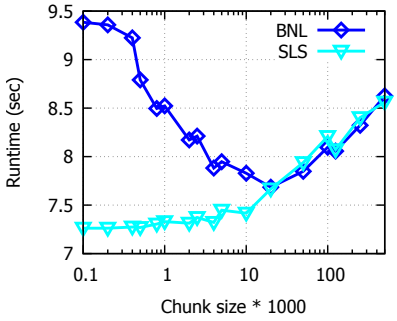
(c) n=100K, dom=[1,5,10]



(d) n=500K, dom=[1,5,10]

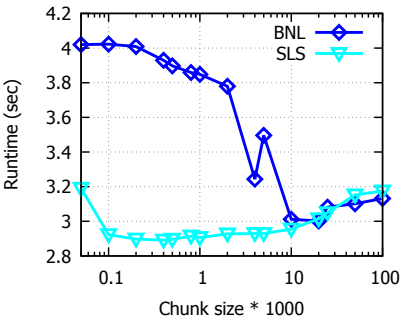


(e) n=100K, dom=[1,2,2,2,2,2,2,2,3]

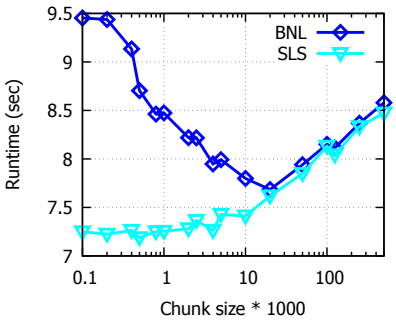


(f) n=500K, dom=[1,2,2,2,2,2,2,2,3]

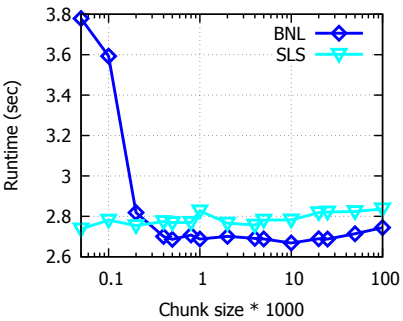
Figure 5.4: Influence of the chunk size. SLS vs BNL, independent data distribution.



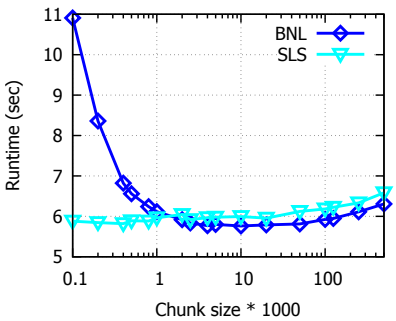
(g) n=100K, dom=[2,3,7,8,4,10]



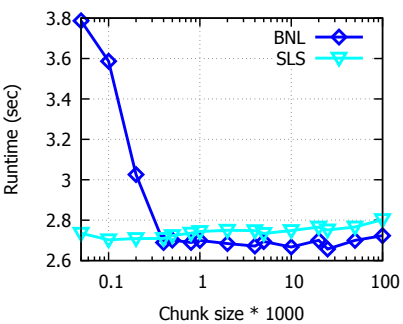
(h) n=500K, dom=[2,3,7,8,4,10]



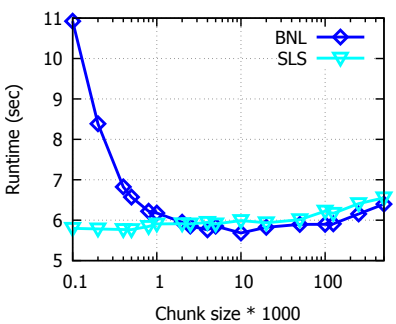
(i) n=100K, dom=[1,5,2560]



(j) n=500K, dom=[1,5,2560]



(k) n=100K, dom=[13,35,70]

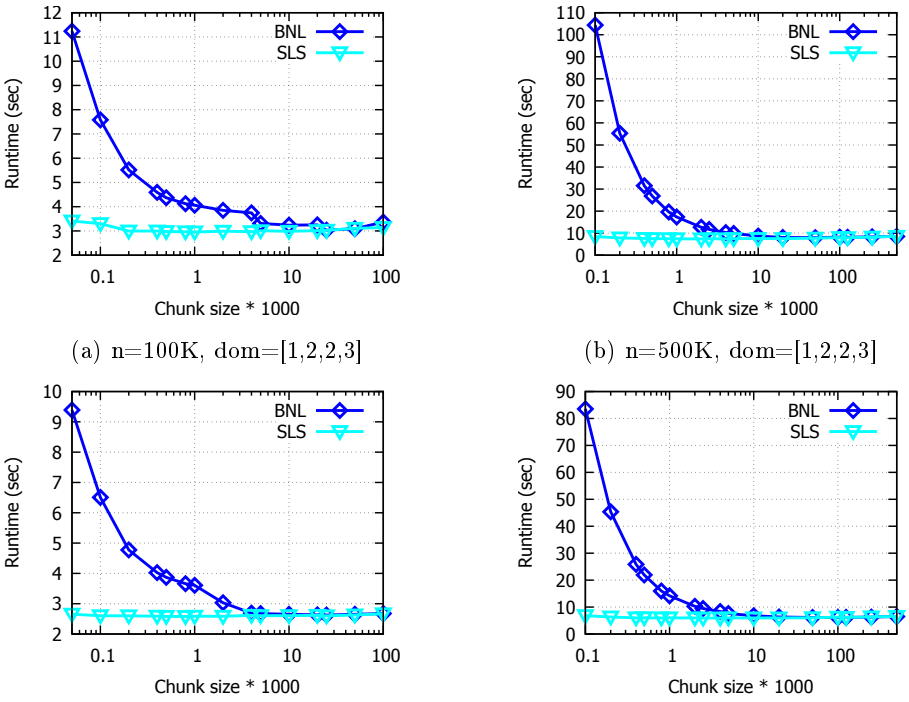


(l) n=500K, dom=[13,35,70]

Figure 5.4: Influence of the chunk size. SLS vs BNL, independent data distribution.

For a better estimation of the optimal chunk size for *independent* data, I take a closer look to the runtime of SLS. **Figures 5.6e** to **5.6h** on page 83 demonstrate the best results for the chunk size between $1K$ and $10K$ objects.

In **Figure 5.5** (pages 80 – 82) one can observe the runtime comparison of BNL and SLS for *correlated* data. For a chunk size up to $2K$ objects SLS is much better than BNL. For larger chunks both algorithms show very similar runtime.



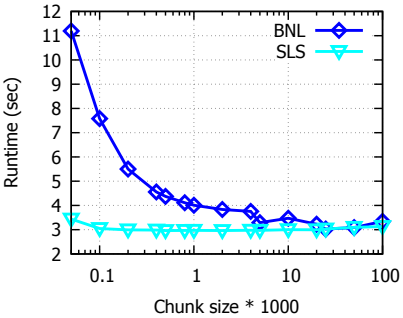
(a) n=100K, dom=[1,2,2,3]

(b) n=500K, dom=[1,2,2,3]

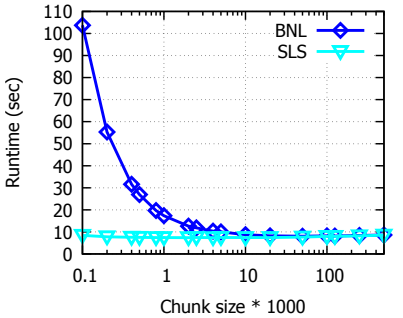
(c) n=100K, dom=[1,5,10]

(d) n=500K, dom=[1,5,10]

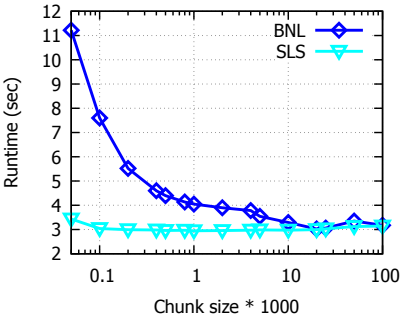
Figure 5.5: Influence of the chunk size. SLS vs BNL, correlated data distribution.



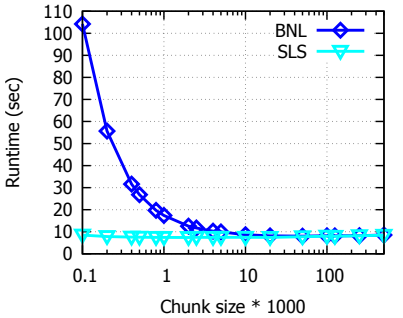
(e) n=100K, dom=[1,2,2,2,2,2,2,2,3]



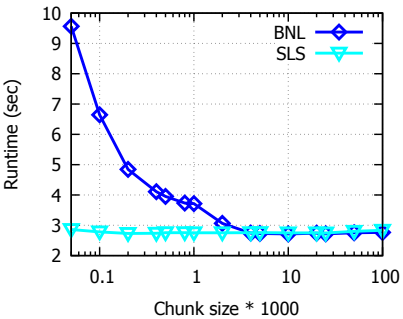
(f) n=500K, dom=[1,2,2,2,2,2,2,2,3]



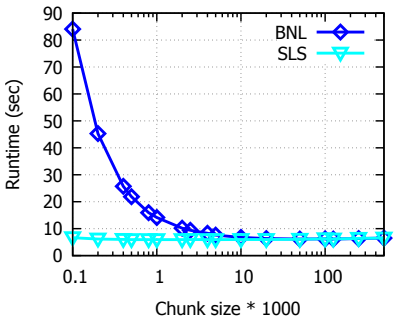
(g) n=100K, dom=[2,3,7,8,4,10]



(h) n=500K, dom=[2,3,7,8,4,10]

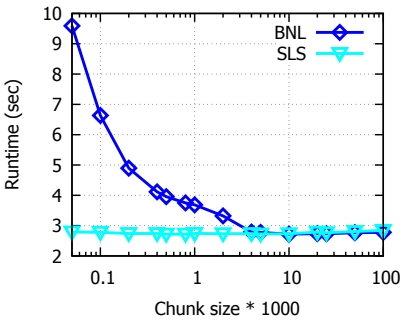


(i) n=100K, dom=[1,5,2560]

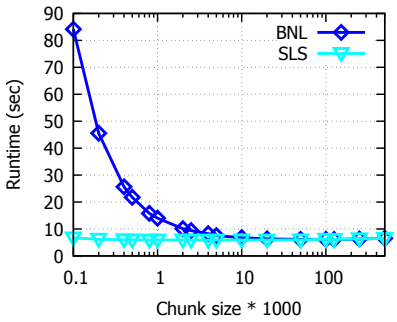


(j) n=500K, dom=[1,5,2560]

Figure 5.5: Influence of the chunk size. SLS vs BNL, correlated data distribution.



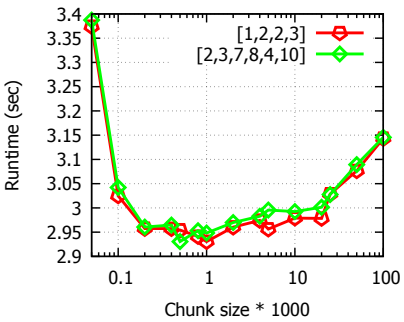
(k) n=100K, dom=[13,35,70]



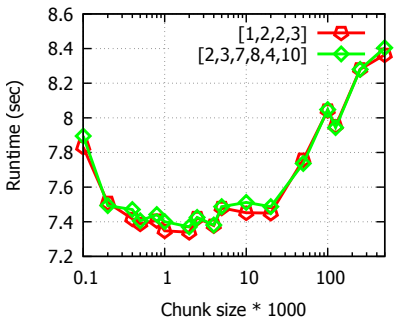
(l) n=500K, dom=[13,35,70]

Figure 5.5: Influence of the chunk size. SLS vs BNL, correlated data distribution.

As one can see in **Figures 5.6i** to **5.6l** on page 84, the SLS achieves its best runtime results for *correlated* data where the chunk size is between 200 and $20K$ objects.

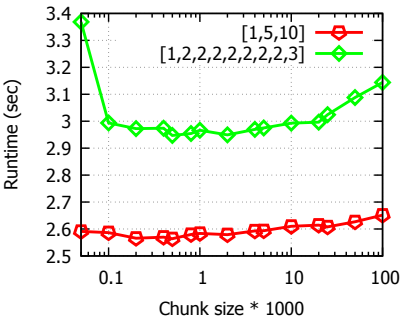


(a) n=100K, anti

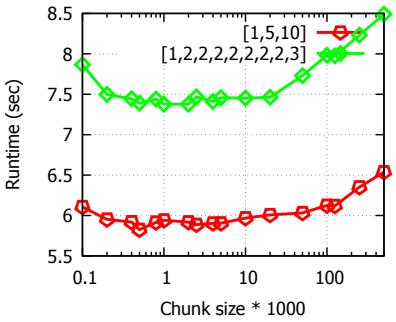


(b) n=500K, anti

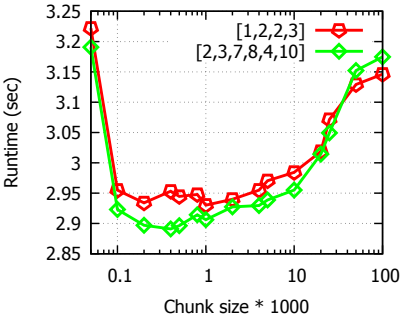
Figure 5.6: Influence of the chunk size on SLS.



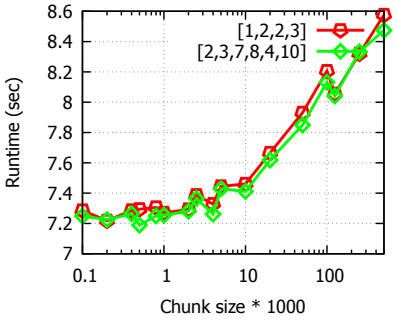
(c) n=100K, anti



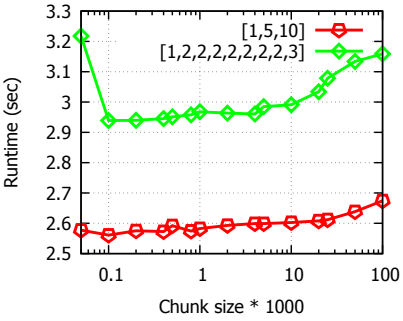
(d) n=500K, anti



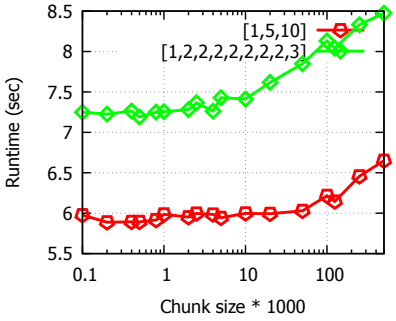
(e) n=100K, ind



(f) n=500K, ind

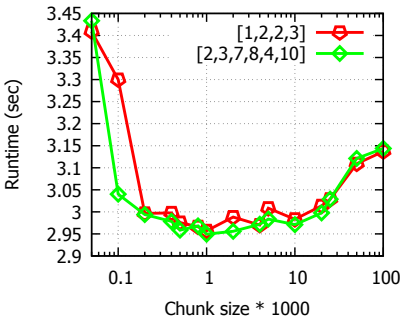


(g) n=100K, ind

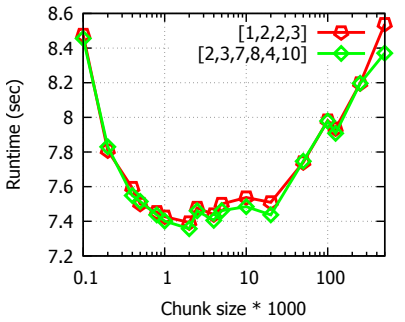


(h) n=500K, ind

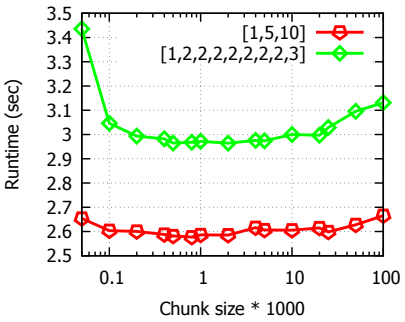
Figure 5.6: Influence of the chunk size on SLS.



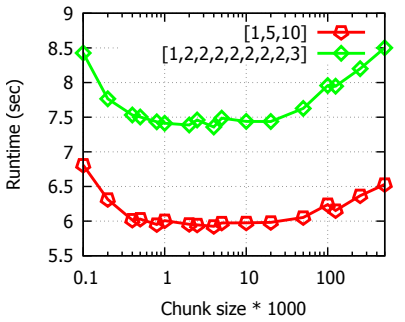
(i) n=100K, corr



(j) n=500K, corr



(k) n=100K, corr



(l) n=500K, corr

Figure 5.6: Influence of the chunk size on SLS.

5.3.3 Influence of Different Domains

In this experiment, I explored the influence of different domains on SLS. I used data with $10K$ and $500K$ objects.

The results on *anti-correlated* data are shown in **Figure 5.6** (page 85). In **Figures 5.6a** and **5.6b** I varied the number of attributes from 4 to 9, while the domain values remain within the *low-cardinality* range $\{0, ..., 10\}$. The runtime behavior is similar for all domains, because low-cardinality domains produce *flat* BTGs and therefore the runtime for the DFT search in the BTG is nearly constant. Note that the [2,3,7,8,4,10] domain has some $47.5K$ nodes, [1,2,2,2,2,2,2,2,3] produces a BTG with $17.5K$ nodes and [1,2,2,3] has only 72 nodes.

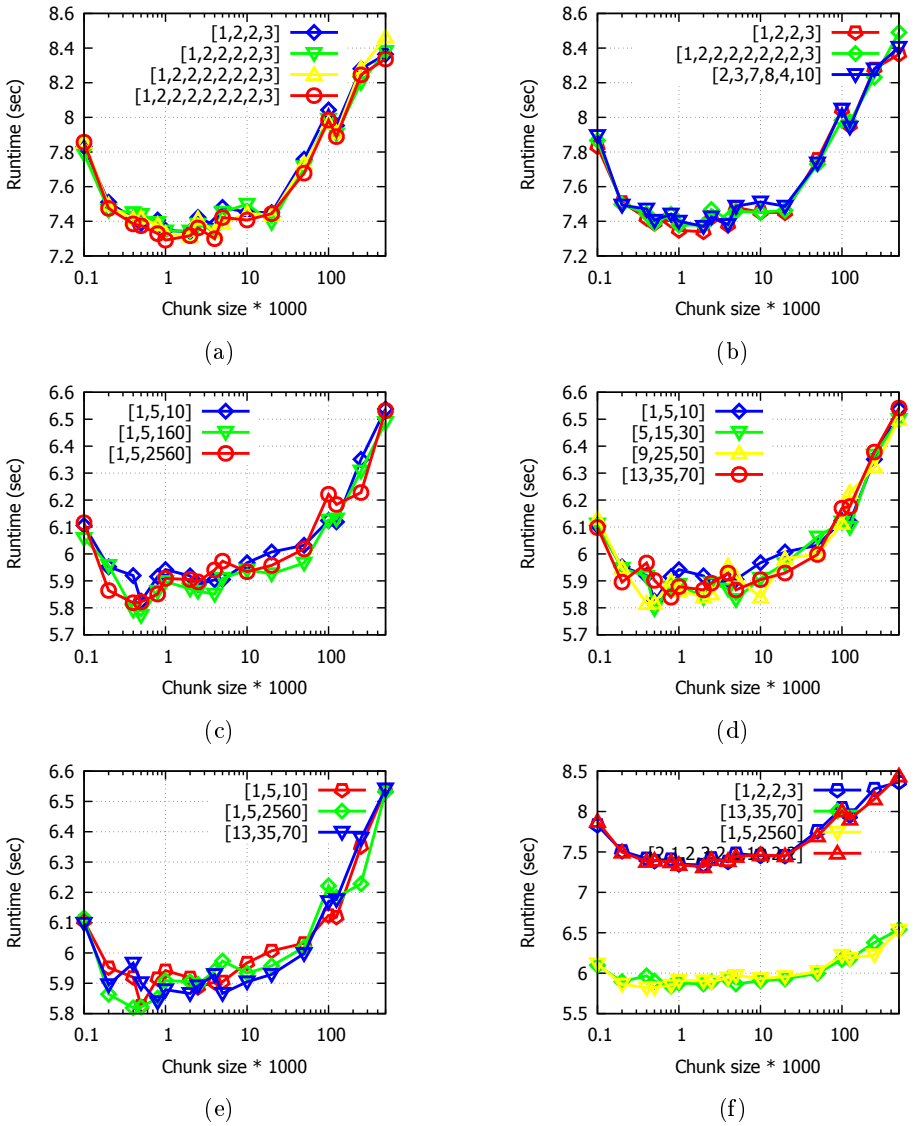


Figure 5.6: Influence of different domains, n=500K, anti-correlated data distribution.

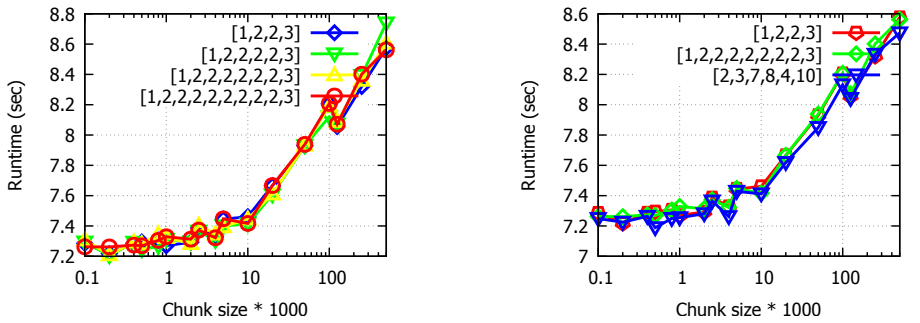
In **Figures 5.6c**, **5.6d** and **5.6e** I compared domains with the same number of attributes, but varied strongly the number of distinct values for each attribute and generated the *high-cardinality* domains. These domains produce *deeper* BTGs in the sense of the *height*, but observe a similar behavior as in **Figure 5.6a** and **5.6b**. There are some $36K$ nodes in the largest BTG and only 132 nodes in the smallest.

In **Figure 5.6f** I compared two domains ([1,2,2,3] and [2,1,2,3,2,1,10,2,2]) producing *flat* BTGs and two domains building *deep* BTGs ([13, 35, 70] and [1,5,2560]). As one can see in this figure, SLS needs more time for *deep* BTGs, because the required time for depth search (DFT) (cp. Removing Phase 3 in Section 5.2.2) for *high-cardinality* domains is significant longer than for *low cardinality* domains.

In summary, the runtimes of SLS are nearly independent from the number of attributes and the size of the domain, as long as we have *low-cardinality* domains. The best chunk size for *anti-correlated* data distribution is between 200 and $20K$ objects for all tested domains.

The influence of different domains on SLS for *independent* data is shown in **Figure 5.7** (pages 86 – 87). Similar to the experiments on *anti-correlated* data, I compared *low-cardinality* domains producing *flat* BTGs in **Figures 5.7a** and **5.7b**, *high-cardinality* domains creating *deep* BTGs in **Figures 5.7c**, **5.7d** and **5.7e** and mixed these domains in **Figure 5.7f**.

The SLS behaviour is very similar for different *low-cardinality* domains and for various *high-cardinality* domains. Furthermore, the runtime of SLS is higher for *deep* BTGs, due to the higher runtime of the DFT.



(a) (b)

Figure 5.7: Influence of different domains, n=500K, independent data distribution.

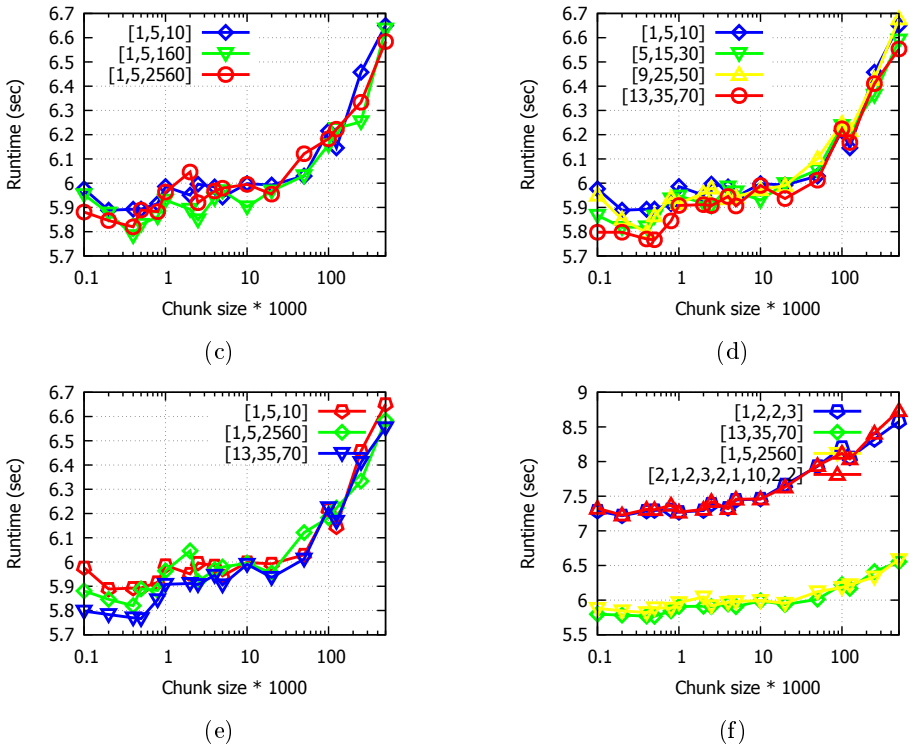


Figure 5.7: Influence of different domains, n=500K, independent data distribution.

The best runtime results SLS shows for the chunk sizes up to $10K$ objects. Compared to *anti-correlated* data, the results for *independent* data have no troubles with small chunks. The very large chunks (more than $10K$ objects) require more runtime because the adding of new objects to the BTG (Phase 2 Section 5.2.2) in SLS is more expensive.

Figure 5.8 (on page 88) shows the experimental results for *correlated* data. I used the same domains as for *anti-correlated* and *independent* data and compared *flat* and *deep* BTGs. These results also confirm my assumption: *runtimes of SLS are nearly independent from the number of attributes and the size of the domain, as long as we have low-cardinality domains. Deep* BTGs produced by *high-cardinality* domains are processed by SLS more slowly. The best runtimes were shown for chunk sizes between 200 and $20K$ objects.

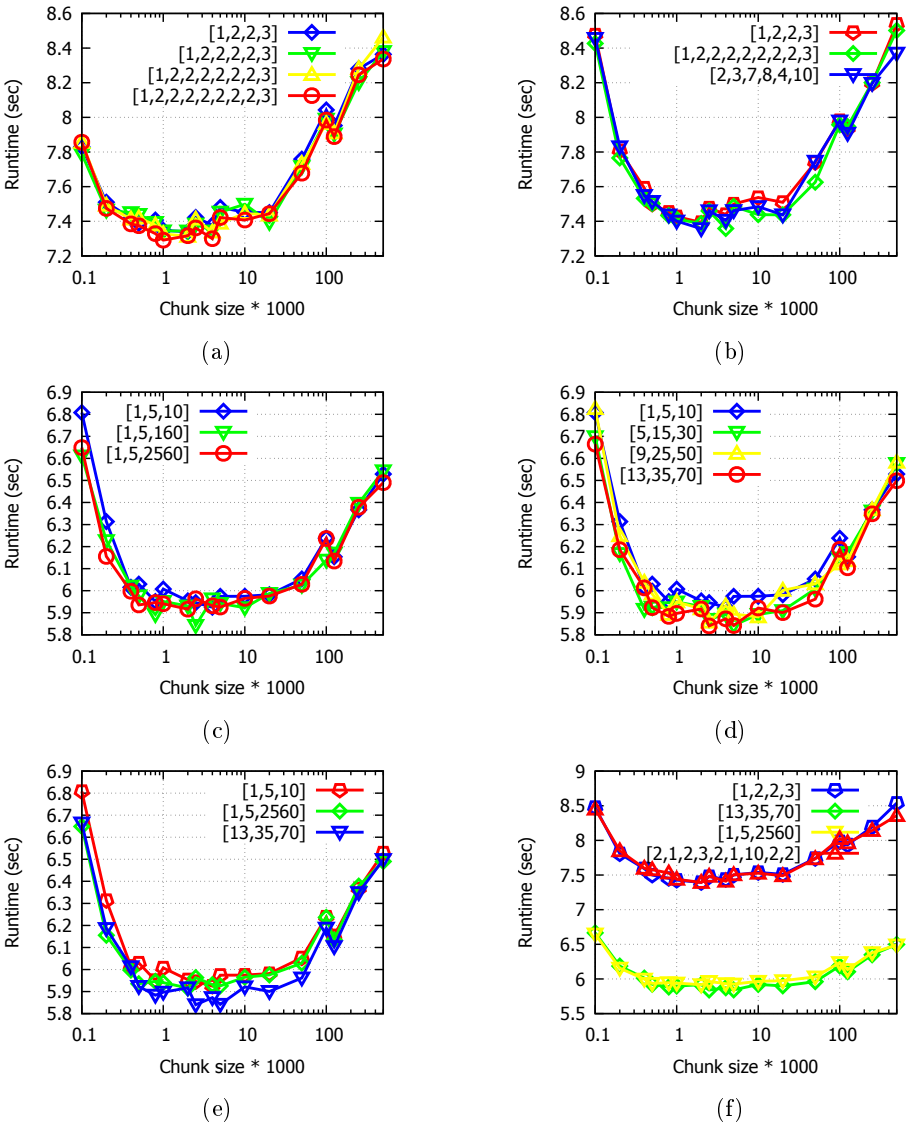
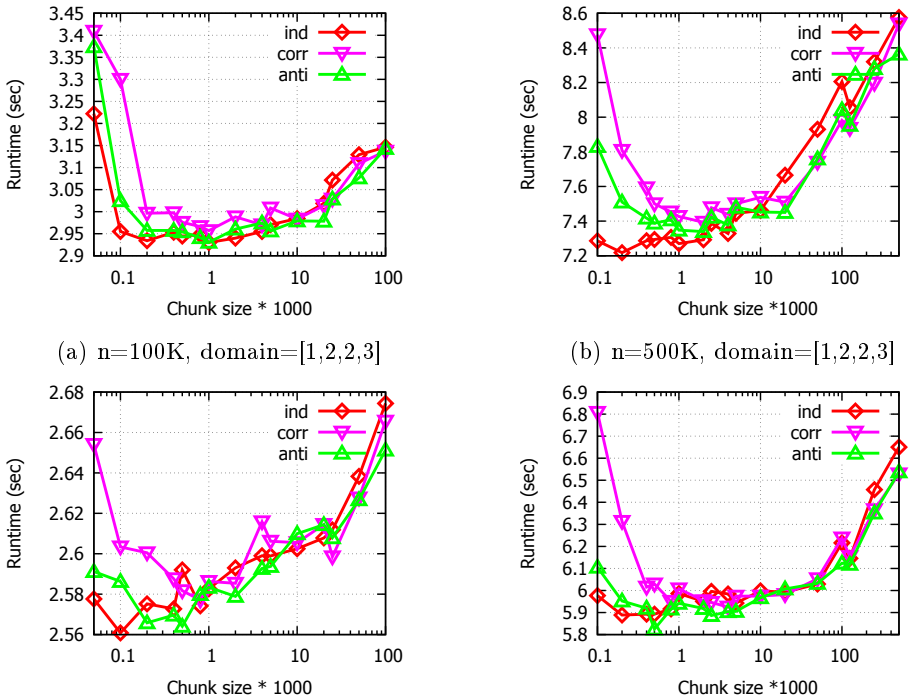


Figure 5.8: Influence of different domains, n=500K, correlated data distribution.

5.3.4 Influence of the Data Distribution

In this experiment I wanted to investigate the impact of different data distributions on SLS. I used independent (ind), correlated (cor), and anti-correlated (anti) data. I varied the size of the dataset ($100K$ and $500K$ objects), and the domains. In **Figure 5.10** (page 89) I compared *low-cardinality* domains producing *flat* BTGs and in **Figure 5.10** on page 90 I demonstrated the SLS behaviour for different data distributions with *high-cardinality* domains generating *deep* BTGs.



(a) n=100K, domain=[1,2,2,3]

(b) n=500K, domain=[1,2,2,3]

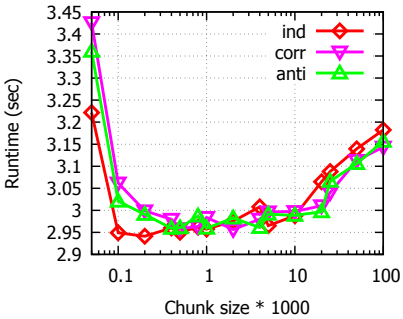
(c) n=100K, domain=[1,5,10]

(d) n=500K, domain=[1,5,10]

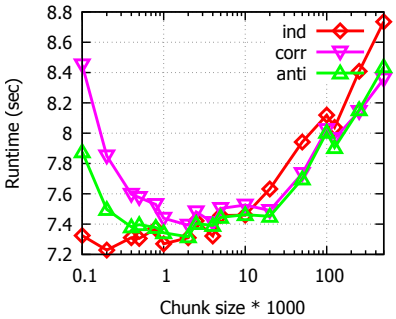
Figure 5.10: Influence of the data distribution on SLS.

SLS relies on the lattice structure a Skyline query constructs and does not depend on any object-to-object comparisons. Therefore, I expected that the runtime of SLS is nearly the same for any kind of data distribution. This expectation was completely fulfilled for *low-cardinality* domains as confirmed by **Figure 5.10**. For *high-cardinality* domains, SLS takes more time to process very small (up to 200 objects) and very large

(over $20K$ objects) chunks. I observed the best runtime for the chunks with $[200; 20K]$ objects, as already shown in Section 5.3.2.



(e) n=100K, domain=[2,1,2,3,2,1,10,2,2]



(f) n=500K, domain=[2,1,2,3,2,1,10,2,2]

Figure 5.10: Influence of the data distribution on SLS.

5.3.5 Runtime Comparison of SLS, Hexagon and BNL

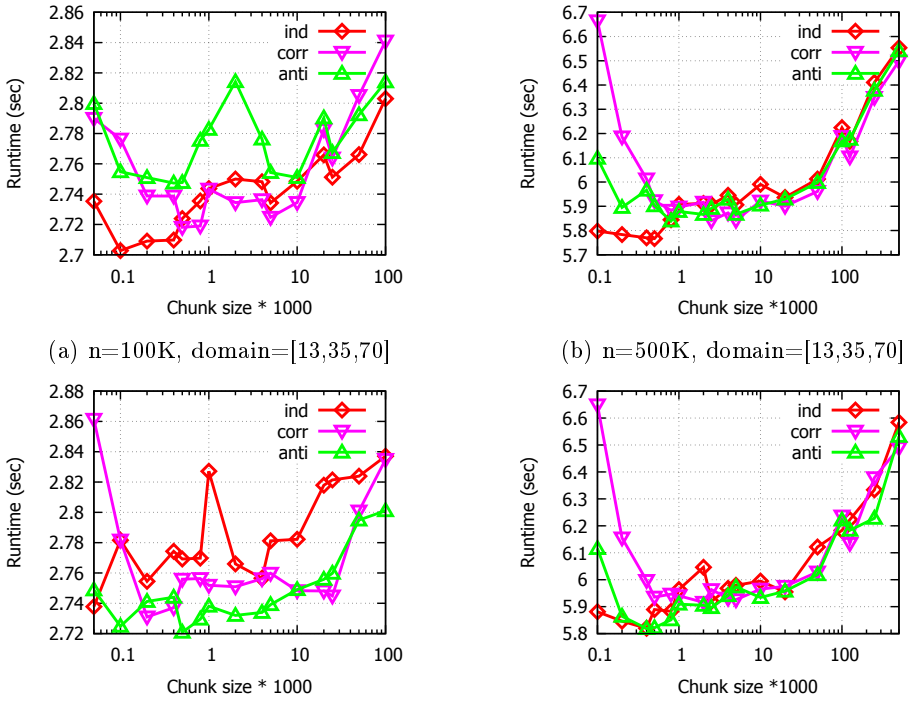
In this section I describe experiments in which I compared SLS with BNL and Hexagon. Since Hexagon is not able to process streams, I analysed the runtimes of Hexagon and compared them to the runtimes of SLS and the stream-based version of BNL with only one big chunk which is equal to the dataset size. I varied the data cardinality, data distribution and domains.

Figure 5.12 (page 92) presents the result for *anti-correlated* data. In this case SLS clearly outperforms BNL: The larger the dataset, the greater the difference.

The results for *independent data* are presented in **Figure 5.13** (page 93). SLS shows often shows the best runtime, but sometimes BNL is better (for example with $50K$ objects for [1,1,2,2,2,2,2,2,2,3] domain or with $500K$ objects for [2856,1,5] domain).

For *correlated* data SLS demonstrates better results for all chunk sizes except $1000K$ objects. For this size BNL is clearly faster (see **Figure 5.14** on page 94).

I tested also *Hexagon* in order to find out if our SLS implementation (based on Hexagon) shows very similar runtime results. And indeed, the two algorithms show very similar runtimes how one can see in **Figures 5.12**, **5.13** and **5.14**.



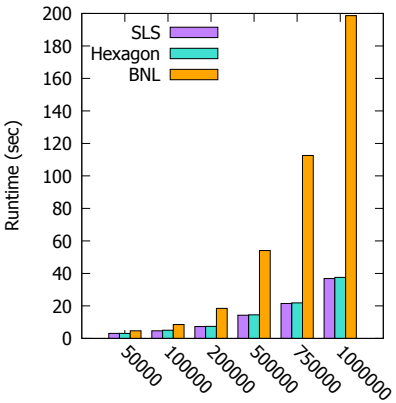
(a) n=100K, domain=[13,35,70]

(b) n=500K, domain=[13,35,70]

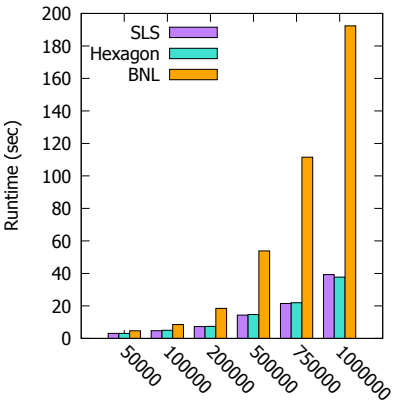
(c) n=100K, domain=[1,5,2560]

(d) n=500K, domain=[1,5,2560]

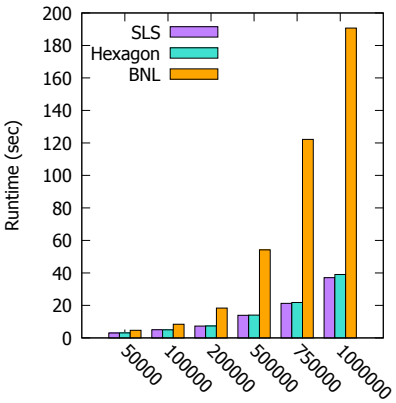
Figure 5.11: Influence of the data distribution on SLS.



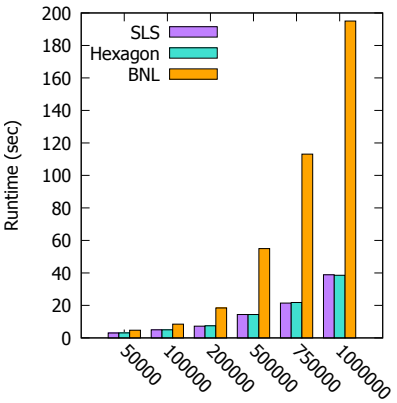
(a) anti, domain=[1,1,2,2,2,2,2,2,2,3]



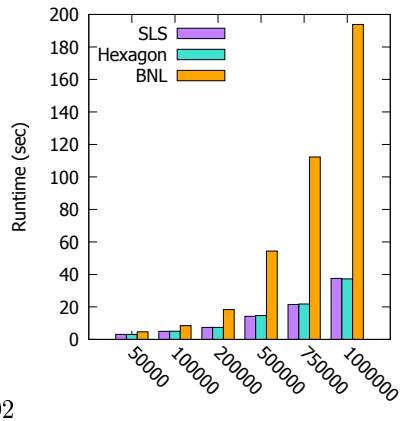
(b) anti, domain=[1,2,2,2,2,3]



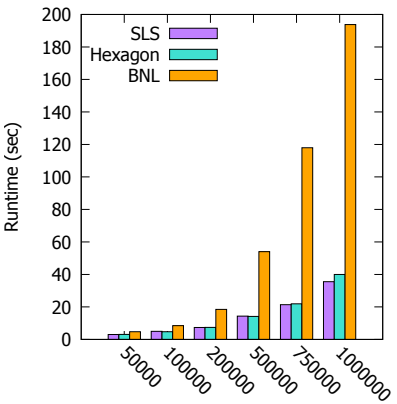
(c) anti, domain=[2,2,100]



(d) anti, domain=[2,3,5,10,100]

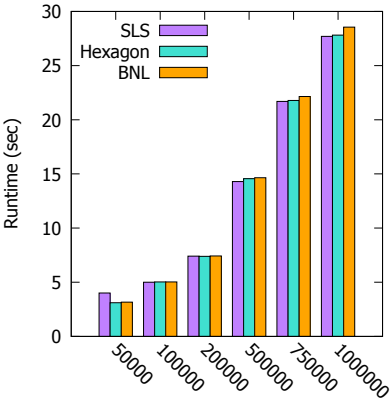


(e) anti, domain=[2856,1,5]

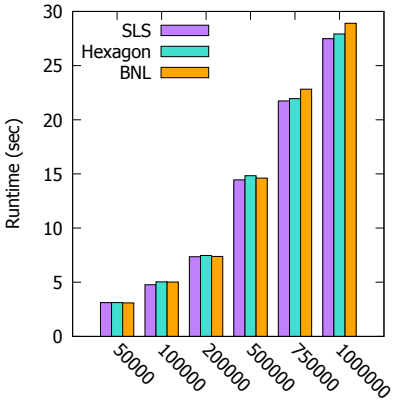


(f) anti, domain=[4,126,77]

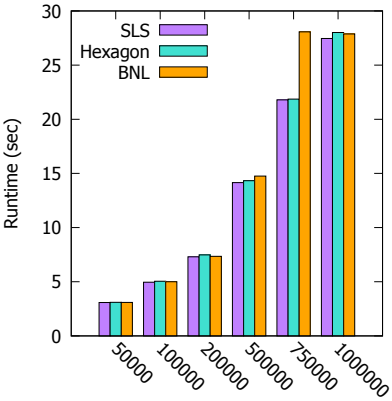
Figure 5.12: SLS vs Hexagon vs BNL, anti-correlated data distribution.



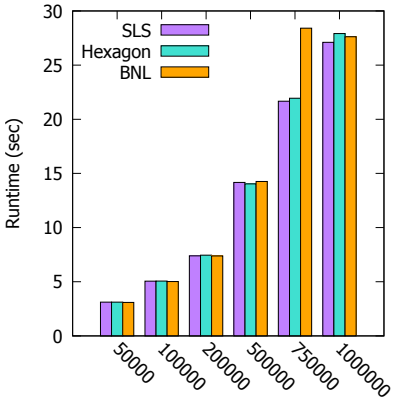
(m) ind, domain=[1,1,2,2,2,2,2,2,2,3]



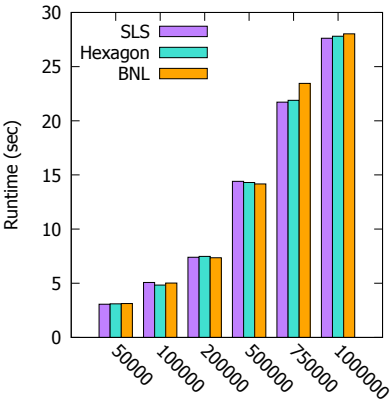
(n) ind, domain=[1,2,2,2,2,3]



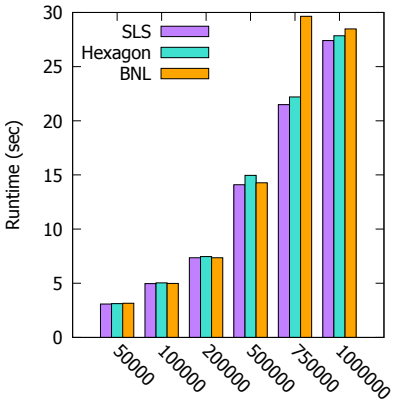
(o) ind, domain=[2,2,100]



(p) ind, domain=[2,3,5,10,100]

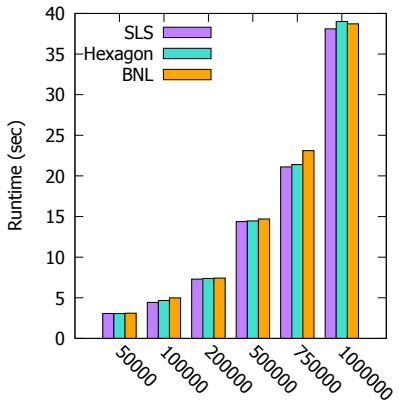


(q) ind, domain=[2856,1,5]

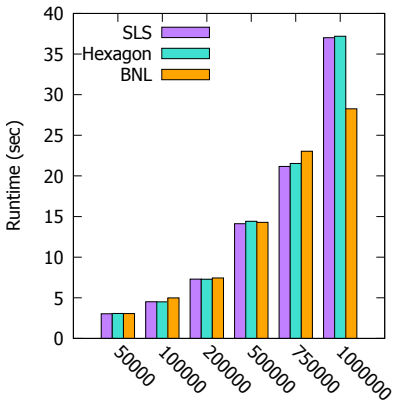


(r) ind, domain=[4,126,77]

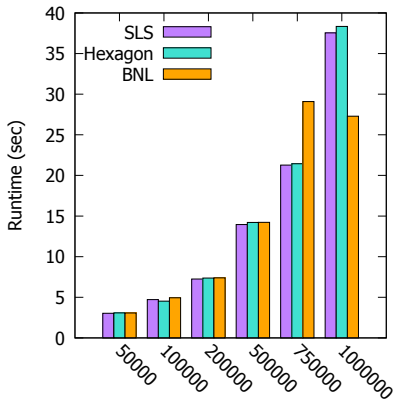
Figure 5.13: SLS vs Hexagon vs BNL, independent data distribution.



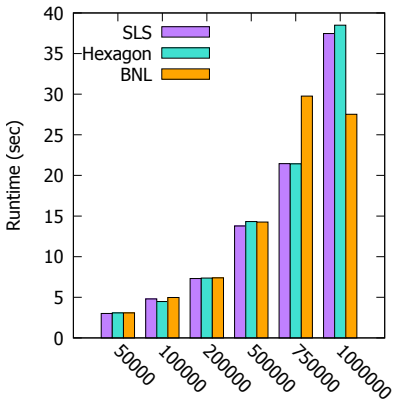
(a) corr, domain=[1,1,2,2,2,2,2,2,2,3]



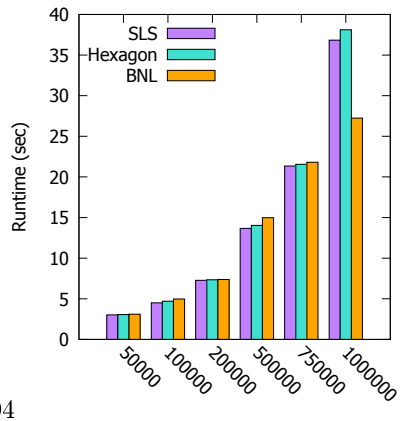
(b) corr, domain=[1,2,2,2,2,3]



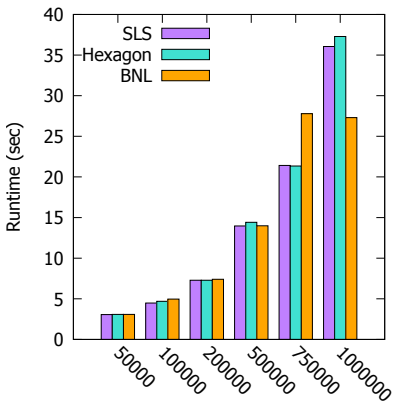
(c) corr, domain=[2,2,100]



(d) corr, domain=[2,3,5,10,100]



(e) corr, domain=[2856,1,5]



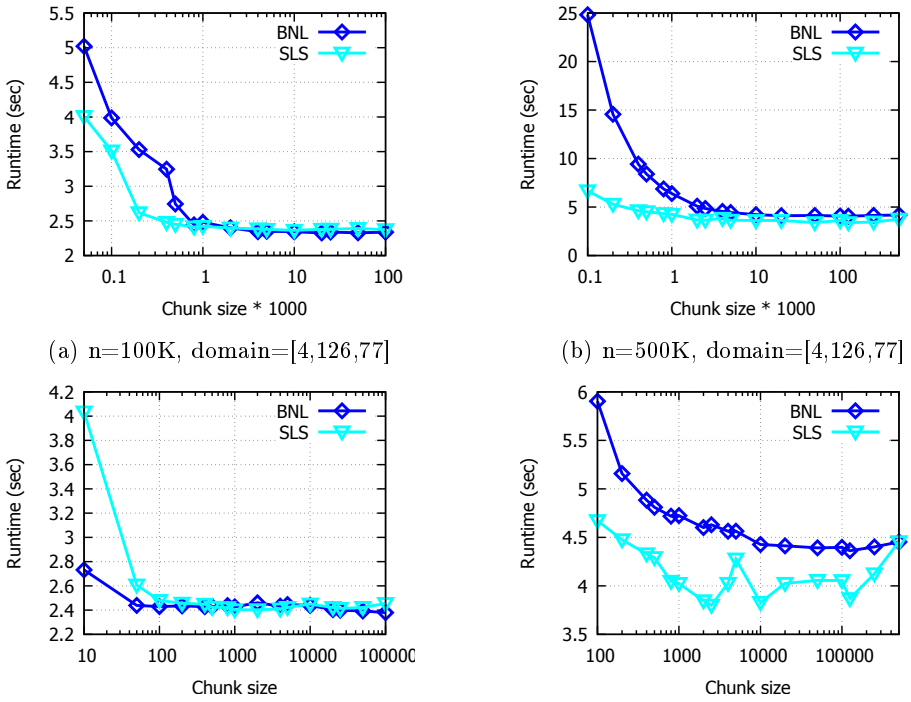
(f) corr, domain=[4,126,77]

Figure 5.14: SLS vs Hexagon vs BNL, correlated data distribution.

5.3.6 Real-World Data

For real data experiments I used tweets collected from Twitter over a specific period of time (for repeatable experiments). I used (disjunct) datasets of $100K$ and $500K$ objects. I mapped all attributes to a numerical domain according to a mapping function as described in Section 5.2.1. For example, I mapped *status_count* (number of posted tweets), *followers_number* and *hashtag* to the numerical domain [2856,5,1].

In my first experiment I compared the runtime of SLS and BNL on different data sizes, but the same domain, cp. **Figure 5.15**.



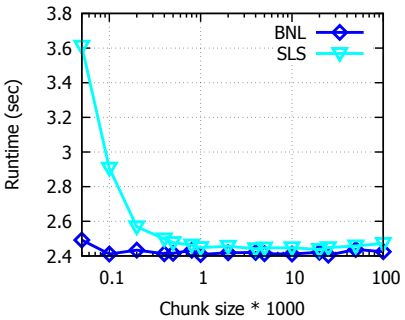
(a) n=100K, domain=[4,126,77]

(b) n=500K, domain=[4,126,77]

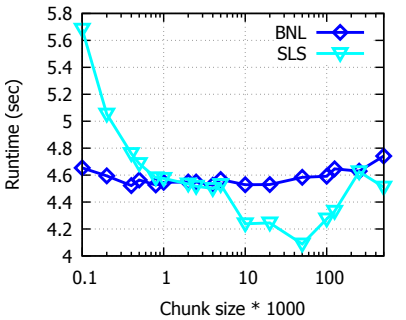
(c) n=100K, domain=[4,77,2,10]

(d) n=500K, domain=[4,77,2,10]

Figure 5.15: Performance of SLS vs BNL on real Twitter data.



(e) n=100K, domain=[2856,5,1]

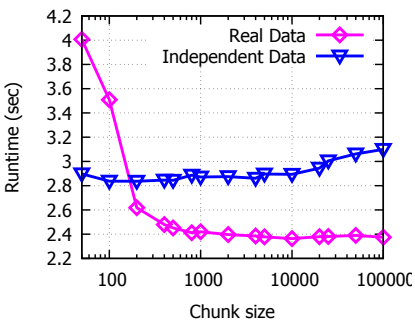


(f) n=500K, domain=[2856,5,1]

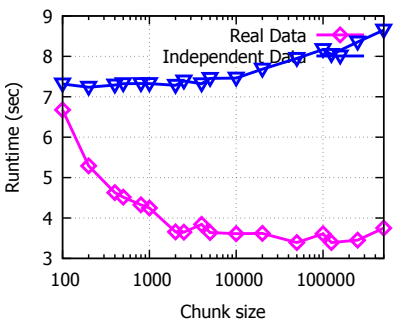
Figure 5.15: Performance of SLS vs BNL on real Twitter data.

Again, for BNL the worst runtime is for small chunks due to the frequent repetition of tuple-comparisons in each single chunk. However, for larger chunks, BNL becomes better. I assume that there are some *killer objects* (objects better than most of the other objects in the dataset), which can be accessed earlier by BNL through the larger chunk sizes and therefore speed-ups performance. Nevertheless, SLS is still better than BNL, in particular for the larger dataset.

In my second experiment I explored the runtime of SLS for real Twitter data in comparison to generated independent data having the same domains ([4,126,77], [4,77,2,10] and [2856,5,1]). I found it interesting to compare real data to independent generated data, since real data is often assumed to be *independent distributed*. **Figure 5.16** presents the results.

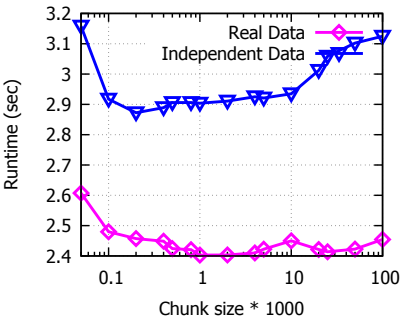


(a) SLS, n=100K, domain=[4,126,77]

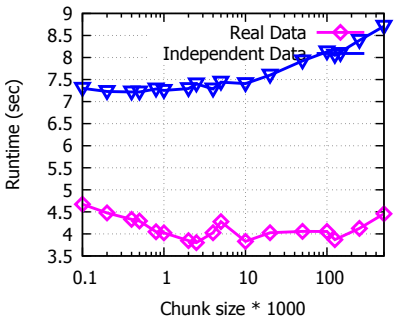


(b) n=500K, domain=[4,126,77]

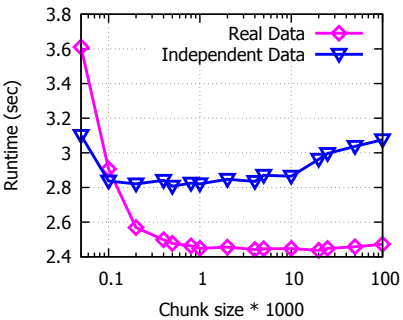
Figure 5.16: Performance of SLS on real and generated independent data.



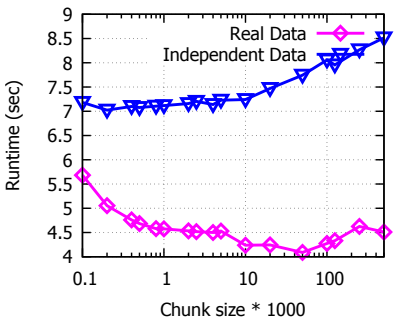
(c) SLS, n=100K, domain=[4,77,2,10]



(d) n=500K, domain=[4,77,2,10]



(e) SLS, n=100K, domain=[2856,5,1]



(f) SLS, n=500K, domain=[2856,5,1]

Figure 5.16: Performance of SLS on real and generated independent data.

Since real data is not always perfectly independent distributed and their are some *killer objects* the results show the expected behavior: SLS performs significant better on real data than on artificial data. In addition, one can see that also for real data chunk sizes between 200 and $20K$ objects are a good choice.

In summary, I conclude that SLS has an optimal runtime for a chunk size between 200 and $20K$ objects. For this range, SLS outperforms BNL for different domains, dataset sizes, and data distributions for real and synthetic data.

This chapter describes a new developed and implemented Stream Lattice Skyline algorithm for efficient Pareto computation on data streams. The preference evaluation on the data streams is not a trivial task, since one has to deal with infinite data that arrive continuously. The new algorithms are required. The first steps in this direction were made with the stream variant of the Block-Nested Loop algorithm. Of the

existing algorithms, to my knowledge, this is the only one that could be adapted to the data streams features. But its runtime was not good, so Stream Lattice Skyline was developed. In this chapter I described the general idea of the algorithm and explained why it should have much better runtime compared to Stream BNL. This hypothesis was then confirmed by various detailed experiments.

Chapter 6

Summarization and Aggregation of Tweets

This chapter describes the summarization and aggregation of tweets, which were filtered from the stream with regard to user's preferences.

The filtered result set can often reach a number of text messages that is annoying to read. For example, it is no fun to read several (almost) identical tweets about COVID-19 disease from British Prime Minister Boris Johnson. The procedure described in this chapter is designed to avoid this situation and presents the filtered set of tweets as a compact summary to the user. The aggregated message should *not contain any duplicates* and should provide *as complete information as possible*. The developed procedure is divided into three major steps: (1) *data preprocessing*, (2) *data clustering* and (3) *data aggregation*. In the first two sections these steps are described first as an *idea* (Section 6.1) and then as an *implemented product* (Section 6.2). Some differences between idea and realisation are also discussed here. Section 6.3 is dedicated to some experiments. Some *performance tests* have been done to estimate the efficiency of this approach, but also the *users' opinion* regarding the created summaries was evaluated.

6.1 General Concept

After evaluating incoming stream of tweets w.r.t. user's preference, the result set can be presented to him. It is important to remember that there is *no final result* while analyzing streams. As new data arrives, the result also changes. The user can receive the result of his query either as new data arrives (and, accordingly, as the result changes) or at equal intervals. The first approach is preferable if the new data and the new result do not appear too often. The second one makes more sense if there is too much incoming data, it arrives very quickly and the result changes every minute. In this case, a constant update of the output results may lead to the user not being able to read and become aware of them. But besides the data output mode, I would like to pay special attention to the form in which the user receives filtered tweets.

Every second, about 9.1 thousand tweets are posted, which means that even the amount of filtered results can still be very large. Not every tweet in this set is as informative as the other. Many users do not produce new and original content, they just retweet. If these *retweets* appear in large numbers in the result set, the user basically has many copies of the same information. Another group of tweets may formally match the search criteria, but may not contain any information, but only author's emotions. In order to reduce the disadvantages mentioned above, I want to aggregate the amount of tweets that were generated after filtering with user-defined preferences and present them to the user as a relatively short message. This message should not contain redundant information, but should include everything that is important.

Example 15 Let us look again at a small amount of example tweets from Section 3.4. The user formulated his preference query with the hashtags *#Germany* and *#WorldCup2018* and has received these four tweets:

1. *It's a sad day for all of Germany and the World Cup.*
2. *After a loss against South Korea, team Germany leaves the World Cup.*
3. *0:2 defeat in the last group match and team Germany leaves the World Cup in Russia.*
4. *RT @UserXyz It's a sad day for all of Germany and the World Cup.*

One of these tweets is a *retweet* (*"RT @UserXyz It's a sad day for all of Germany and the World Cup"*), the other one contains the *emotions* of the user (*"It's a sad day for all of Germany and the World Cup"*). And the remaining two should be aggregated to the message like follows using the approach described in this chapter:

„Team Germany loses its last group match against South Korea 0:2 and leaves the World Cup in Russia.“

To achieve this goal and to present well readable messages with all available facts about the event to a user, the following steps have to be carried out:

1. *data preprocessing*
2. *data clustering*
3. *data aggregation*

6.1.1 Data Preprocessing

Tweets often include abbreviations, errors (intentional or accidental), internet slang, emojis, URLs, etc. Therefore, the *preprocessing* of tweets is a very important step if I want to have a good summarization of text messages, which were filtered out from the stream. It will include standard text processing procedures (tokenizing, stop words removal), but also Twitter specific steps, like handling of retweets, URLs, etc.

A lot of tweets are *retweets* - messages which do not contain any new information and only quote the other author's post. If the original message was posted by an author with a large number of followers, the number of retweets can also be large and they occur frequently in the result set obtained by evaluating of user's preference query. All retweets have to be deleted if the original message is in the result set, too. If this is not the case, one of the retweets remains in the query output, the other ones have to be removed. For the tweets in Example 15 that means that the last tweet of the collection, which is retweet, (*"RT @UserXyz It's a sad day for all of Germany and the World Cup"*) will be deleted and the original message (the first one in the collection) remains in the set.

The next step in tweets preprocessing is *text tokenizing*. The authors use in their messages often very specific language, so it is not enough to remove punctuation and to divide the text into character sequences separated by blank spaces. Twitter tokenizing has to include handling of emojis, URLs and in a limited way - abbreviations. Slang and abbreviations are difficult to handle because of their diversity even within a single language. Subsequently the *stop words* have to be removed. In addition, all letters in the words are converted to lower case in order to facilitate further clustering and aggregation. In this way, after tokenizing and stop words removing the tweet *"After a loss against South Korea, team Germany leaves the World Cup"* from Example 15 looks like this: *"after loss against south korea team germany leaves world cup"*.

At this point I want to inspect the content of tweets for the first time. A simple and informative way is to look for the most common *N-grams* in the set of text messages.

An N-gram is a contiguous collection of n items, which can be phonemes, letters, words, etc. depending on the application. In this approach, text tokenizing delivers sequences of words such that N-grams are build from n words each. My idea is to use the calculated $N-$grams as centroids for clustering. This step can also be a part of the clustering. But the $N-$grams can also be used in other ways, e.g., if one looks at the most common $N-$grams, one could get an impression of the content of the tweets. Because of the limited length of tweets it seems to make sense to build $2-$, $3-$ or $4-$grams and to use the most common k of them. The $3-$grams for the already partly preprocessed tweet *"after loss against south korea team germany leaves world cup"* are *"after loss against"*, *"loss against south"*, *"against south korea"*, *"south korea team"*, *"korea team germany"*, *"team germany leaves"*, *"germany leaves world"* and *"leaves world cup"*.

6.1.2 Data Clustering

I want to summarize the tweets, which were filtered out from Twitter stream with Preference SQL. That means that these data include either *perfect matches* w.r.t. the user preferences or (much more often) contain *best possible alternatives*. The provided results can describe a single event as well as discuss several topics. My goal is to summarize the messages that do fit together in terms of content. Therefore, I want to perform clustering and collect the tweets describing the same topic or event in the same group. After that the messages in each group can be summarized.

If the data set includes *perfect matches*, the number of tweets in the set is not very large in most cases and they can be mapped well to a single topic. However, the clustering is very important step for the case involving result set of tweets with *best alternatives*. Looking for tweets with hashtags *#Ukraine* and *#WorldCup2018*, for example, the user gets a result set, which can be well separated in at least two groups: one with hashtag *#Ukraine* and the other with *#WorldCup2018*, because the soccer team of Ukraine did not qualify for the World Cup 2018.

There exist many different cluster approaches, e.g., well known and popular *k-means* clustering. But I wanted to avoid the costs of calculating the similarity between the individual tweets and came up with the idea to use the most common k $N-$grams (calculated during preprocessing) as cluster centroids. The tweets that have the corresponding $N-$grams are then simply assigned to "their" clusters. That means of course that after the first round I will have a large amount of clusters, some of them are *exactly the same* (but with different $N-$grams as centroids), the others *overlap to a certain degree*. This will be resolved by merging the clusters that are "too similar" or overlap by a certain percentage. This value can be a fixed, but a better option is to make the value variable and leave this decision to the user, because it can depend on

many factors and varies from application to application. I will also have the tweets that do not belong to any of the k most common clusters. One can either ignore them completely or calculate a similarity to the existing clusters for these few unassigned tweets in order to classify them.

6.1.3 Data Aggregation

After *data preprocessing* and *clustering* I *aggregate* the data within each cluster.

Automatic summarization of documents is a topic of research work for a long time. In most cases, however, this term refers to a summarization of longer documents. The result in this case is a short summary of one or more longer documents to give users a brief overview of the content. The summary is always shorter than the original document and for the most part uses phrases and expressions from the original documents. Summarization of micro-blog posts, such as tweets, has different goals and tasks. There exist multiple possibilities to summarize the text messages. They can be roughly divided into three groups:

(1) In the first approach the summary is a kind of label or keyword sequence for the cluster, which titles the topic of every message in the group. This approach is used, e.g., by O'connor, Krieger and Ahn in [OKA10]. Such kind of summary is very short and does not go into details. For the set of tweets from Example 15 the summary could be *Germany in the World Cup*. Reading such a summary, the user figures out that the tweets are about team Germany in the World Cup, but for more information he has to read the original messages.

(2) The second possibility is to choose only one text message as representative of the cluster. This representative should include the most important information for this cluster. Takamura and Okumura in [TO09] call the searching for such message „budgeted median problem“ and consider the summary as good if every message in the document cluster is assigned to a selected one and can be inferred from the latter. For Example 15 such a representative tweet should be the second: *„After a loss against South Korea, team Germany leaves the World Cup“* or the third one: *„0:2 defeat in the last group match and team Germany leaves the World Cup in Russia“*. This approach gives the user more details, but some information (score or opponent) is lost.

(3) To overcome the disadvantages of the previous approaches, I want to automatically generate the not yet existing text message using as much accessible information from objects in the cluster as possible. Sharifi, Inouye and Kalita in [SIK14] developed the *Phrase Reinforcement* algorithm that largely solves this problem.

The main idea is to display the words from text messages as nodes in a graph. These nodes have certain weights, depending on how often they appear in the text collection. Walking through the paths of this graph, it is possible to restore every text message. The path with the largest total weight is the result message, which will be presented to the user.

Let us take a closer look at this last approach using set of tweets from Example 15. To build the graph we need a root node. The most common word or word sequence in the text collection can be used for this purpose. The root node is already identified during preprocessing by searching for most common N-grams. Starting from the root, the graph is constructed. Figure 6.1 shows the resulting graph for our example set of tweets.

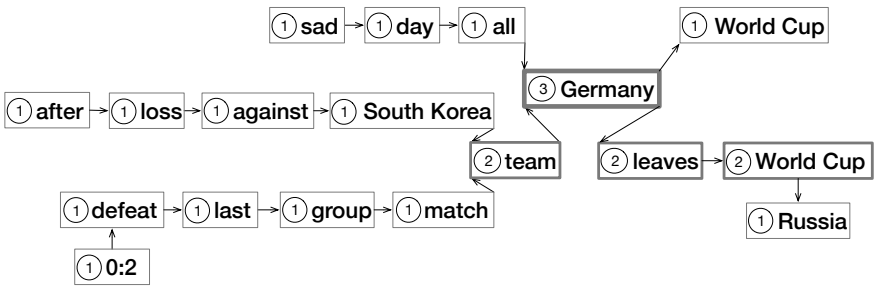


Figure 6.1: Text graph.

In this example, the nodes are weighted according to how often they occur in the text collection. For more details and extensions cp. [SIK14]. One can see that the most common word in this collection is *Germany*, which occurs 3 times. *World Cup* also can be found 3 times, but it follows after the word *leaves* twice and comes right after *Germany* once. Together with the word *team* that is located directly before *Germany*, the most important key information from our collection is: *team Germany leaves World Cup*.

Looking at the sum of the node weights for every path in the graph from the beginning of the sentence to the end, one can see that some of them have very close values, e.g., the path that corresponds to the *after loss against South Korea team Germany leaves World Cup Russia* message has the total weight of 14, *0:2 defeat last group match team Germany leaves World Cup Russia* comes up to 15, and so on. In the original *Phrase Reinforcement* algorithm the authors take only the path with the highest total weight as a result. I want to expand this approach

and consider as the result all paths with the total weight above a certain threshold value or those that deviate from the maximum only by a certain value. If I set the threshold value, e.g., to 14, we have to consider three paths with the common snippet *team Germany leaves World Cup*. In two paths I find the word *Russia* right after this snippet, which will then be included into the result message. Full path with total weight over 13 can start either with the *after loss against South Korea* fragment or with *0:2 defeat last group match* one. Both alternatives can be included into the final result, e.g., by combining them with a comma or the conjunction „and". The resulting message then looks as follows: *after loss against South Korea and 0:2 defeat in last group match team Germany leaves World Cup in Russia*. Since I still have the original messages with all stop words, I will use them to build the result message, which is as close as possible to human language. This text has all important information from the tweet collection and can be presented to the user.

6.2 Implementation

During the implementation of the concept described in Chapter 6.1 the individual steps were extended and adapted to the challenges presented by the real Twitter data. Nevertheless, the three global steps have remained unchanged: (1) *preprocessing*, (2) *clustering* and (3) *aggregation*. The details of the implementation are described in the following subchapters.

6.2.1 Preprocessing

The main goal of preprocessing is to transform the tweets into a *normalized*, *generalized* form. Note that the complete preprocessing was developed for very specific short texts. This takes into account a special character of tweets (e.g., a lot of *links* and *emojis*, *user references* and *hashtags*) and cannot be used for other texts without adaptation. The included actions are therefore mainly aimed to eliminate "unwanted" characters and components. The implementation was made for the tweets in English language. Accordingly, such tweets should be selected beforehand. At first, each tweet that was identified as a retweet, is substituted by its original message. If duplicates occur, they will be sorted out during clustering. Example 16 illustrates the complete tweet preprocessing step by step:

Example 16 Extended preprocessing including building of N-grams (with $n = 3$). Transformed places are marked blue:

Input: a set of tweets (retweets are already replaced by original messages):

a) *Winner of five GRAMMY Awards including 'Best New Artist' & 'Song of the Year', ⋆ @billieeilish ⋆ attended the 62nd in custom #Gucci by #AlessandroMichele. #GucciJewelry #GRAMMYs @RecordingAcad #GucciEyewear*

b) *#AvengersEndgame hit $2.6 billion in ticket sales worldwide, inching toward #Avatar and its $2.78 billion global box office record!! http://bit.ly/2HPFbn5 Señor Feige is a genius @Disney #Avengers #Marvel*

Step 1: transformation of the upper case letters to the lower case letters:

a) *winner of five grammy awards including 'best new artist' & 'song of the year', ⋆ @billieeilish ⋆ attended the 62nd in custom #gucci by #alessandromichele. #guccijewelry #grammys @recordingacad #guccieyewear*

b) *#avengersendgame hit $2.6 billion in ticket sales worldwide, inching toward #avatar and its $2.78 billion global box office record!! http://bit.ly/2HPFbn5 señor feige is a genius @disney #avengers #marvel*

First, all uppercase letters in tweets are converted to lowercase. Since capitalization in English is only used in rare cases (e.g., at the beginning of a sentence, proper names, brands, etc.), thus the loss of information is negligible. However, the distinction between the same words written in lower or in upper case is saved for further processing.

Step 2: translation of any existing HTML entities (here: &)

a) *winner of five grammy awards including 'best new artist' & 'song of the year', ⋆ @billieeilish ⋆ attended the 62nd in custom #gucci by #alessandromichele. #guccijewelry #grammys @recordingacad #guccieyewear*

b) *#avengersendgame hit $2.6 billion in ticket sales worldwide, inching toward #avatar and its $2.78 billion global box office record!! http://bit.ly/2HPFbn5 señor feige is a genius @disney #avengers #marvel*

Next, any existing HTML entities[22] are translated into their correct special character representation.

[22] HTML entities: www.key-shortcut.com/html-entities

Step 3: normalization with compatible decomposition:

a) winner of five grammy awards including 'best new artist' & 'song of the year', ★ @billieeilish ★ attended the 62nd in custom #gucci by #alessandromichele. #guccijewelry #grammys @recordingacad #guccieyewear

b) #avengersendgame hit $2.6 billion in ticket sales worldwide, inching toward #avatar and its $2.78 billion global box office record!! http://bit.ly/2HPFbn5 señor feige is a genius @disney #avengers #marvel

Special characters are separated into their components[23]. This step has to prevent misunderstanding of the unformatted HTML remnants as words and treatment of special characters different from their normal counterparts.

Step 4: removal of URLs and e-mail addresses:

a) winner of five grammy awards including 'best new artist' & 'song of the year', ★ @billieeilish ★ attended the 62nd in custom #gucci by #alessandromichele. #guccijewelry #grammys @recordingacad #guccieyewear

b) #avengersendgame hit $2.6 billion in ticket sales worldwide, inching toward #avatar and its $2.78 billion global box office record!! señor feige is a genius @disney #avengers #marvel

The URLs, which appear frequently, and the e-mail addresses that are less common in the tweets, are removed. These components usually do not have any special meaning for a human user, but will appear as disturbing elements during clustering.

Step 5: removal of "unwanted" characters (here: ★ ' ' , . ! ˜ ' '):

a) winner of five grammy awards including best new artist & song of the year @billieeilish attended the 62nd in custom #gucci by #alessandromichele #guccijewelry #grammys @recordingacad #guccieyewear

b) #avengersendgame hit $2.6 billion in ticket sales worldwide inching toward #avatar and its $2.78 billion global box office record senor feige is a genius @disney #avengers #marvel

All special characters that may negatively affect later clustering and are not important for the content of a tweet are deleted. Characters that have an added value with regard to the information content remain in the text. This includes *lower case letters* and *arabic numerals*, *spaces* and *apostrophes* commonly used in English, as well as *accidentals* (+, −), *points*, *colons* and *commas*, if they appear in connection with

[23] Unicode Normalization: de.wikipedia.org/wiki/Normalisierung_(Unicode)

numbers (e.g., in $-2.100,55$). The *glyphs* for ampersand (&), paragraph (§), percent (%) and for the most common currencies (dollar, euro, pound, yen/renminbi) are also preserved. While paragraph, percent and currency symbols are essential for the meaning of an associated number, the ampersand is often used as an abbreviation instead of the conjunction "and" and should be retained. The *hash* (#) and the *at sign* (@) also play a special role. They are used from Twitter as a marker for *hashtags* (#...) and *references* to user accounts (@...) and will be retained at first.

Step 6: removal of additional hashtags and user references:

a) winner of five grammy awards including best new artist & song of the year @billieeilish attended the 62nd in custom #gucci by #alessandromichele

b) #avengersendgame hit $2.6 billion in ticket sales worldwide inching toward #avatar and its $2.78 billion global box office record senor feige is a genius @disney

If *hashtags* or *user references* are embedded in the middle of a tweet, they will not be removed, as they most likely fulfil the role of normal words. If groups of said components are located directly at the beginning or at the end of a tweet, they are removed with the exception of the innermost element. It is very hard to define a method that reliably picks the most informative hashtags or user references from a sequence because there is no grammar or otherwise "semantic" structure in such sequences. In many cases it appears that a tag or reference taken from somewhere near the middle of a sequence is not to bad a choice, but it remains a rather simple ad-hoc-solution. Therefore a possibility to change the settings was provided in the implementation: you can additionally delete the remaining hashtags, the user references, both (hashtags and user references) or neither of them. In the current example the latter option is used.

Step 7: lemmatization of the remaining text elements:

a) winner of five grammy award include best new artist & song of the year @ billieeilish attend the 62nd in custom # gucci by # alessandromichele

b) # avengersendgame hit $ 2.6 billion in ticket sale worldwide inch toward # avatar and its $ 2.78 billion global box office record senor feige be a genius @ disney

In this step the words are reduced to their stems. One should distinguish between *stemming* and *lemmatization*. *Stemming* shortens the morphological forms of a word to its stem by truncating the suffix. The variants of a word are reduced to a common stem, sometimes resulting in forms that do not exist in the language. In the context of *lemmatization*, it is also the goal to reduce the variations of a word to its basic form (stem). Hence, in contrast to stemming, lemmatization provides linguistically correct expressions. This is done by using linguistic analyses and a large vocabulary, which

often produces better results but only comes by the price of higher computational effort [DPH09]. Since the final goal of this approach is primarily intended to deliver an information-rich and comprehensible final message to the user, it is necessary to maintain linguistic comprehensibility to the best of the ability. Therefore, the stem form reduction is carried out as *lemmatization*.

Hashtags and *user references* that still are in the tweet are treated in a special way here: they remain unchanged, since hashtags are often complete units like proper names or because they form the non-existing complex "words" and lemmatization cannot reduce them to their stems. However, their identifiers (# or @) are separated. This way it is ensured that the "simple" hashtags (e.g., *#trump*, *#usa*, *#oscar*) are matched with the same words that are not marked as hashtags during the clustering. For long compound and often invented hashtags (e.g., *#BlackLivesMatter*) no partners will be found despite separation of #.

Step 8: tokenisation of lemmatized messages:

a) (winner, of, five, grammy, award, include, best, new, artist, &, song, of, the, year, @, billieeilish, attend, the, 62nd, in, custom, #, gucci, by, #, alessandromichele)

b) (#, avengersendgame, hit, $, 2.6, billion, in, ticket, sale, worldwide, inch, toward, #, avatar, and, its, $, 2.78, billion, global, box, office, record, senor, feige, be, a, genius, @, disney)

In this step, the lemmatized tweets are tokenized: it is ensured that each word or free-standing character forms a separate token.

Step 9: removal of the stop words (here: of, the, in, by, and, its, be, a, &, #, @):

a) (winner, five, grammy, award, include, best, new, artist, song, year, billieeilish, attend, 62nd, custom, gucci, alessandromichele)

b) (avengersendgame, hit, $, 2.6, billion, ticket, sale, worldwide, inch, toward, avatar, $, 2.78, billion, global, box, office, record, senor, feige, genius, disney)

Now the stop words must be removed from the token sequences, since they are basically not relevant with regard to content and thus clustering. On the other hand, they are important for the comprehensibility of a text, which is why they are essential for the building of the final message. To avoid sorting out the stop words first and then inserting them again afterwards, the token lists are saved before. In addition, the amount of stop words varies depending on the application area and institution. In social networks, for example, frequently discussed topics use often slang or common shortcuts, while scientific or political tweets are usually more formally written. A single unchangeable list of stop words is therefore suboptimal.

Thus, here again the user is offered the possibility to select his preferred stop words. The default list of stop words, which is used if the user does not define his own ones, can be found in Appendix A.2. This also includes implementation specific stop words, such as the identification symbols of hashtags (#) and user references (@). They are separated during lemmatization in order to allow higher matches during clustering. However, since the user should recognize hashtags and user references in the final message, the characters (# and @) are not completely deleted, but recorded as stop words.

Step 10: building of the (in itself alphabetically sorted) $N-$grams ($n = 3$):

a) {(five grammy winner), (award five grammy), (award grammy include), (award best include), (best include new), (artist best new), (artist new song), (artist song year), (billieeilish song year), (attend billieeilish year), (62nd attend billieeilish), (62nd attend custom), (62nd custom gucci), (alessandromichele custom gucci)}

b) {($ avengersendgame hit), ($ 2.6 hit), ($ 2.6 billion), (2.6 billion ticket), (billion sale ticket), (sale ticket worldwide), (inch sale worldwide), (inch, toward worldwide), (avatar inch toward), ($ avatar toward), ($ 2.78 avatar), ($ 2.78 billion), (2.78 billion global), (billion box global), (box global office), (box office record), (office record senor), (feige record senor), (feige genius senor), (disney feige genius)}

The final step of the preprocessing is the generation of $N-$grams. These consist of n successive tokens each and are used to compare the tweets. An appropriate length depends on the expected similarity of the texts, so the choice is up to the user. As default value $n = 3$ is recommended, but for tweets with very similar content $n = 4$ can also be useful. Otherwise, the comparison of $N-$grams with $n \geq 4$ is too strict, resulting in vast numbers of very small clusters. On the other hand $N-$grams with $n < 3$ are usually not meaningful enough. In addition, lexicographic sorting of the tokens within an $N-$gram is performed. $N-$grams, which are similar in content, but differ due to the sequence of tokens, should be considered identical (in clustering). So, e.g., the following 3$-$grams, which are literally different, but of the same meaning < *cost* 300 $ > and < *cost* $ 300 > can be rearranged to < $ 300 *cost* > whereby they match.

6.2.2 Clustering

After the complex preprocessing, clustering takes place. I use my own cluster approach, which uses the $N-$grams calculated during preprocessing as centroids. Only *one iteration* through the incoming data set is necessary. No similarity between the

tweets is calculated, because the affiliation to a cluster is determined with the help of the N−gram centroids. This makes this cluster approach quite efficient. I explain it in detail in the following Example 17.

Example 17 Clustering process to determine the k largest clusters (with k = 5). Tweets are shown as sequences of their tokens and tokens are simplified as letters. Centroids are marked blue:

Input: a quantity of tweets including generated N−grams (here: n = 3):

ID	text (tokens)	set of N−gramms
11	*(a b c d e f g e)*	*(a b c), (b c d), (c d e), (d e f), (e f g), ~~(e f g)~~*
12	*(a b c d e f h g)*	*(a b c), (b c d), (c d e), (d e f), (e f h), (f g h)*
13	*(a b c d e f g h)*	*(a b c), (b c d), (c d e), (d e f), (e f g), (f g h)*
14	*(b c d e f g h)*	*(b c d), (c d e), (d e f), (e f g), (f g h)*
15	*(l m n o p q r s)*	*(l m n), (m n o), (n o p), (o p q), (p q r), (q r s)*
16	*(u v w x y z a b)*	*(u v w), (v w x), (w x y), (x y z), (a y z), (a b z)*
17	*(u v w x y z a)*	*(u v w), (v w x), (w x y), (x y z), (a y z)*
18	*(w x y z a x y z)*	*(w x y), (x y z), (a y z), (a x z), (a x y), ~~(x y z)~~*

Table 6.1: Input tweets with the corresponding 3−grams.

Step 1: building a cluster for each N−gram (n = 3), sorting the tweets and elimination duplicates (see Table 6.2):

The N-grams generated during the preprocessing are used as cluster centers or *centroids*. The original idea of using the k most frequently occurring N-grams as centroids (cp. Section 6.1.2) was abandoned in order to avoid unnecessary iterations through the entire data set. Instead, a separate cluster is defined for each of the N-grams of each tweet, which initially contains the same tweet from which it originated. If an N-gram, for which a cluster already exists (with this cluster as centroid), is processed, the tweet belonging to the N-gram is immediately included in the respective cluster. With this approach, only *one iteration* over all text messages is required for the clustering process. Each created cluster is considered as a *set* and therefore may not contain multiple tweets with the same ID[24] at any time. After this step, there is exactly one cluster for all different N-grams. Each of these clusters contain only tweets with corresponding centroid (or one of the unsorted permutations of the N−gram).

[24] Tweet ID: every tweet is assigned a unique identification number by Twitter at the moment it is posted.

centroid (size)	tweets ID	tweets text (tokens)
a b c (3)	11	*(a b c d e f g e)*
	12	*(a b c d e f g h)*
	13	*(a b c d e f g h)*
b c d (4)	11	*(a b c d e f g e)*
	12	*(a b c d e f h g)*
	13	*(a b c d e f g h)*
	14	*(b c d e f g h*
c d e (4)	11	*(a b c d e f g e)*
	12	*(a b c d e f h g)*
	13	*(a b c d e f g h)*
	14	*(b c d e f g h*
d e f (4)	11	*(a b c d e f g e)*
	12	*(a b c d e f h g)*
	13	*(a b c d e f g h)*
	14	*(b c d e f g h*
e f g (3)	11	*(a b c d e f g e)*
	13	*(a b c d e f g h)*
	14	*(b c d e f g h)*
	~~11~~	~~*(a b c d e f g e)*~~
e f h (1)	12	*(a b c d e f h g)*
f g h (3)	12	*(a b c d e f h g)*
	13	*(a b c d e f g h)*
	14	*(b c d e f g h*

centroid (size)	tweets ID	tweets text (tokens)
l m n (1)	15	*(l m n o p q r s)*
m n o (1)	15	*(l m n o p q r s)*
n o p (1)	15	*(l m n o p q r s)*
o p q (1)	15	*(l m n o p q r s)*
p q r (1)	15	*(l m n o p q r s)*
q r s (1)	15	*(l m n o p q r s)*
u v w (2)	16	*(u v w x y z a b)*
	17	*(u v w x y z a*
v w x (2)	16	*(u v w x y z a b)*
	17	*(u v w x y z a*
w x y (3)	16	*(u v w x y z a b)*
	17	*(u v w x y z a)*
	18	*(w x y z 1 x y z)*
x y z (3)	16	*(u v w x y z a b)*
	17	*(u v w x y z a)*
	18	*(w x y z a x y z)*
	~~18~~	~~*(w x y z a x y z)*~~
a y z (3)	16	*(u v w x y z a b)*
	17	*(u v w x y z a)*
	18	*(w x a y z x y z)*
a b z (1)	16	*(u v w x y z a b)*
a x z (1)	18	*(w x y a x z y z)*
a x y (1)	18	*(w x y z a x y z)*

Table 6.2: Clusters with the corresponding tweets, 3−grams used as centroids.

Step 2: combining of the overlapping clusters (here: $p = 1.00$) (Table 6.3 left),

Step 3: discarding of the clusters that are to small (here: $s = 3$) and

Step 4: returning of two remaining largest cluster (despite $k = 5$) (Table 6.3 right):

Next, all clusters that overlap to a certain percentage p are merged (step 2). This step is necessary because the clusters build in step 1 overlap and one tweet belongs to all clusters with the N−grams from this tweet as centroid. Considering clusters from Table 6.2 one can see that the clusters presented in it are partially *identical* (e.g., the clusters with centroids *bcd*, *cde* and *def*) or *very similar* (e.g., the clusters with centroids *bcd*, *abc*, *efg*, *fgh* and *efh*).

centroid (size)	tweets ID	tweets text (tokens)
b c d (4)	11	*(a b c d e f g e)*
	12	*(a b c d e f h g)*
	13	*(a b c d e f g h)*
	14	*(b c d e f g h)*
w x y (3)	16	*(u v w x y z a b)*
	17	*(u v w x y z a)*
	18	*(w x y z a x y z)*
l m n (1)	15	*(l m n o p q r s)*

centroid (size)	tweets ID	tweets text (tokens)
b c d (4)	11	*(a b c d e f g e)*
	12	*(a b c d e f h g)*
	13	*(a b c d e f g h)*
	14	*(b c d e f g h)*
w x y (3)	16	*(u v w x y z a b)*
	17	*(u v w x y z a)*
	18	*(w x y z a x y z)*
~~*l m n (1)*~~	~~15~~	~~*(l m n o p q r s)*~~

Table 6.3: Merging of the overlapping clusters, discarding of the small cluster.

Two clusters C_1 and C_2 are merged if the following condition applies:

$$\frac{|C_1 \cap C_2|}{\min(|C_1|, |C_2|)} \geq p \tag{6.1}$$

Merging means that all messages of the smaller cluster (based on the number of tweets) are included in the larger cluster and the former is deleted afterwards. Note that since the clusters are sets, the tweets with the same IDs are *not* added multiple times. For the parameter p a guideline value between 0.6 and 0.7 (or $60\% - 70\%$) is recommended based on experience. But the decision is left to the user, as this value is highly dependent on the *content* of the tweets. For the clusters describing, for example, a political or social event, the p value could also be lower, while for the clusters with the users' *emotional content* about certain event even 70% could be too low. Knowing the contents of the clusters allows to estimate the value of p parameter, without this knowledge the guideline of $60\% - 70\%$ should be respected. After merging, 3 clusters are left in the current example (see Table 6.3 left).

The resulting clusters are now either larger clusters, which were created by merging, or smaller clusters, which could not be merged. The latter are to be classified as insignificant outliers and cannot be used for the further procedure due to their small size. Furthermore, all clusters are discarded that fall below a certain minimum size s (step 3). A threshold value s depends on the total number of tweets as well as on their similarity in content and therefore varies from case to case. The definition of s is therefore again left to the user. However, the value should at least be chosen so that clusters with a one-digit number of tweets are eliminated (cluster with the centroid *lmn* in the current example, see Table 6.3 right). At the end of clustering, the remaining clusters are sorted in descending size order and, if available, the first k representatives are returned for further aggregation (step 4).

6.2.3 Aggregation

After cluster building is completed, the aggregation of the tweets contained in each set is carried out. The last of three approaches, which was briefly described in Chapter 6.1.3 and which is based on the Phrase Reinforcement algorithm, is implemented with some extensions (I use several paths for the final message and not only the path with the highest weight) and adaptations (the weights of the paths are calculated without stop words and weakly weighted nodes) and applied to the k largest clusters. Many text aggregation methods use neuronal networks, which require a huge amount of training data. The approach I use does not achieve the quality of the produced text, which is possible with neuronal networks, but it works on any amount of data without any kind of training and is easy to understand and to implement. Example 18 shows a simplified illustration of this approach with explanations:

Example 18 Aggregation of the tweets of a cluster and creation of the final message. Tweets are shown as sequences of their tokens, tokens are simplified as letters:

Input: one of the k largest clusters (see Table 6.4). Centroids are marked blue, tweets, which do not contain the centroid are highlighted red):

centroid (size)	**tweets**			
	ID	**text (tokens)**	**ID**	**text (tokens)**
d e f (20)	21	*(b c d e f g h i),*	22	*(c d e f gd e f),*
	73	*(a b c d e f),*	23	*(r u x d e f g h),*
	54	*(a b c d e f h g),*	46	*(c d e f h),*
	92	*(a b c d e f g h),*	48	*(v b c d e f g h),*
	31	*(l m n c d e f s t),*	50	*(m n c d e f s t u),*
	51	*(n c d e f s u t),*	86	*(m n c e d f h g),*
	37	*(c f d e g d),*	76	*(r u x d e f y z),*
	95	*(v b c d f e s t),*	26	*(d e f s t u v),*
	34	*(m n c d e f g h i),*	91	*(w x d e f s t u),*
	97	*(m n c d f e g h),*	60	*(a k l d k f g h)*

Table 6.4: Cluster with the tweets.

Step 1: building of the tweet graphs (see Figure 6.2):

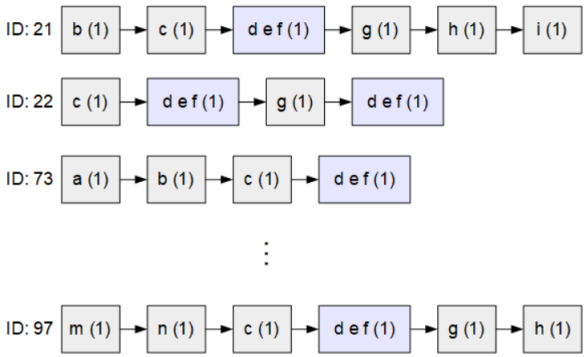


Figure 6.2: Single tweet graphs.

First, each tweet is represented as a separate graph (*tweet graph*), with its tokens forming the nodes and being connected by directed edges according to the reading direction (see Figure 6.2). The stop words are *not* included in the graph and the tokens representing the cluster centre are combined to one node. If a cluster contains tweets that do not contain the centroid (or its unsorted permutation), which can happen sometimes by combining the clusters, they will not be considered (e.g., tweet with ID 60 in Table 6.4). Furthermore, each node receives a standard weight of 1.

Step 2: merging the tweet graphs into a common cluster graph (see Figure 6.3):

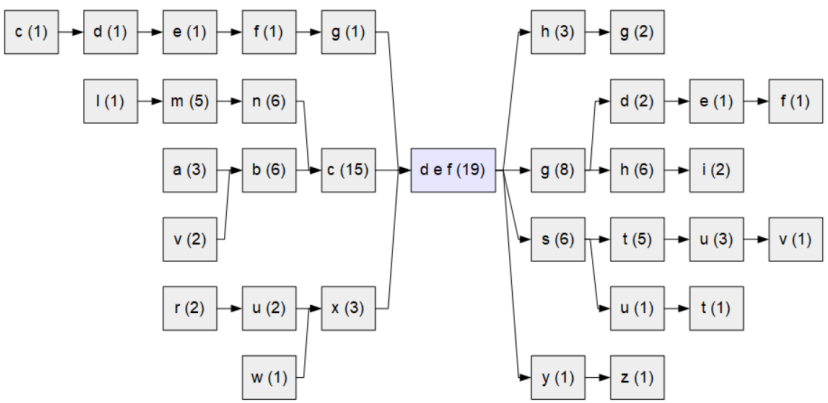


Figure 6.3: Complete cluster graph.

In this step, the tweet graphs of a cluster are connected via the centroid as a common intersection. This is done by "superimposing" the graphs at the central node. The combined centroid is assigned a new weight according to the number of combined tweets. Likewise, all identical sub-paths that begin or end at the centroid are superimposed. The adjustment of the node weights is done analogously. If the special case occurs that a tweet contains centroid of its cluster n times, where $n > 1$, this tweet is also included n times in the combined cluster graph. If this results in an overlap with other nodes, their weight will be increased regularly, if it was not already incremented before while one of the other $n-1$ centroid's occurrence of the current tweet was processed. This is to ensure that all centroid occurrences in a tweet are treated equally, but the weight of the node in tweet graph is not incremented repeatedly by a single tweet.

Step 3: reducing the cluster graph (here: removing all nodes with an initial weight of 1. A stricter reduction does not occur in this example, see Figure 6.4):

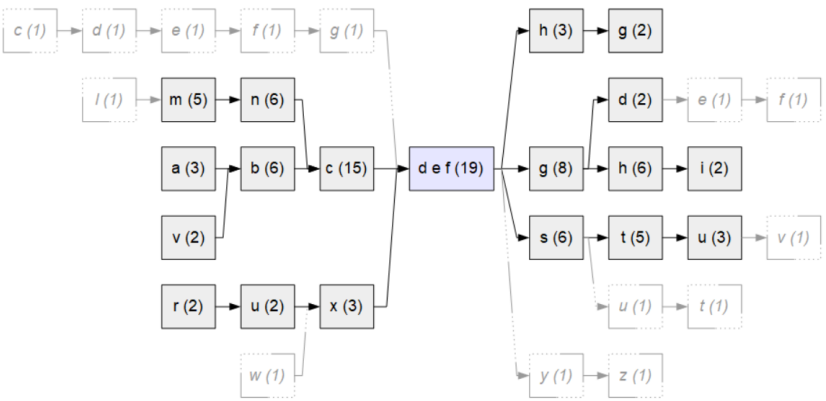


Figure 6.4: Reduced cluster graph (remote nodes are marked light grey).

Since the content of a cluster should be presented to the user in only a few sentences, some less important information is omitted in this step. *Strong weighted* nodes are considered *important* and *weakly weighted* nodes are considered *insignificant*. The centroid, which always has the maximum weight, therefore represents the core content of the cluster while nodes that still have their initial weight of 1 are regarded as marginal information and deleted from the cluster graph. In addition, the user has the option to make an even more stringent reduction by removing all nodes below a certain minimum weight.

Step 4: selection of the paths for the m parts of the final message (here: $m = 4$, see Tables 6.5–6.8). The most appropriate path in each phase is marked green:

i) Determination of the main path (the path with the highest total weight):

unused paths (U)	
path (weight)	**ovl. to M**
(m n c d e f g h i) - (61)	0
(m n c d e f s t u) - (59)	0
(a b c d e f g h i) - (59)	0
...	...

selected paths (M)	
nr.	**path** (weight)

Table 6.5: Determination of the main path.

ii) Determination of the second path (the path with the smallest overlap (ovl.) to the main path):

unused paths (U)	
path (weight)	**ovl. to M**
(a b c d e f h g) - (48)	$2/6 \approx 0.33$
(r u x d e f h g) - (31)	$1/6 \approx 0.17$
(a b c d e f s t u) - (57)	$2/7 \approx 0.29$
(r u x d e f s t u) - (40)	$1/7 \approx 0.14$
...	...

selected paths (M)		
nr.	**path** (weight)	
1	*(m n c d e f g h i) - (61)*	

Table 6.6: Determination of the second path.

iii) Determination of the third path (the path with the smallest overlap (ovl.) to all paths in M):

unused paths (U)	
path (weight)	**ovl. to M**
(a b c d e f h g) - (48)	$2/6 \approx 0.33$
(v b c d e f h g) - (47)	$2/6 \approx 0.33$
(a b c d e f g d) - (53)	$3/6 = 0.5$
(v b c d e f g d) - (52)	$3/6 = 0.5$
...	...

selected paths (M)		
nr.	**path** (weight)	
1	*(m n c d e f g h i) - (61)*	
2	*(r u x d e f s t u) - (40)*	

Table 6.7: Determination of the third path.

iv) Determination of the forth path (the paths with the smallest overlap (ovl.) to all paths in M):

unused paths (U)	
path (weight)	**ovl. to M**
(v b c d e f h g) - (47)	5/6 ≈ 0.83
(v b c d e f g d) - (52)	4/6 ≈ 0.67
(v b c d e f g h i) - (58)	6/7 ≈ 0.86
...	...

selected paths (M)	
nr.	**path** (weight)
1	*(m n c d e f g h i) - (61)*
2	*(r u x d e f s t u) - (40)*
3	*(a b c d e f h g) - (48)*

Table 6.8: Determination of the forth path.

In this step the individual paths of the graph are selected, which will be converted later into the sentences of the final message. A path starts with a node without incoming edges (*start node*) and ends with the one without outgoing edges (*end node*). A *path* is considered as a *complete sentence*.

According to the concept, several sentences per cluster should be presented to the user. Thus, several suitable paths must be determined. Let us take a closer look at this approach: one always starts with the path with the *highest total weight* (*main path*, see Table 6.5. Instead of using the paths with the next highest total weights afterwards, as it was the idea at the start, the paths that *differ the most* from those used so far must be selected. The reason for this is that the paths whose weights differ the least from each other have very similar contents. But the goal is to provide as much *additional* information from the cluster to the user as possible (compared to the main path only). Thus for each path that has not been used yet, it will be checked how much it overlaps with the already selected paths (which at the beginning, is only the main path). The path that has the least overlap or, in case of a tie, the higher weight, is selected as the next sentence of the final message, see Table 6.6. In this way, the greatest possible diversity among the selected information can be achieved. If there are still unused paths afterwards, the procedure is repeated for each further sentence until maximum m is reached (see Tables 6.7 and 6.8). The maximum number m of how many sentences the final message of a cluster should contain, is specified by the user.

Step 5: building of the sentences of the final message (inserting the light grey token sequences from Figure 6.4 (marked here orange), points and capital letters, see Table 6.9):

selected paths (M)			
nr.	**path** (weight)		
1	*(m n c d e f g h i) - (61)*	→	*L m n c d e f g h i.*
2	*(r u x d e f s t u) - (40)*	→	*R u x d e f s t u v.*
3	*(a b c d e f h g) - (48)*	→	*A b c d e f h g.*
4	*(v b c d e f g d) - (52)*	→	*V b c d e f g d e f.*

Table 6.9: Building the final message.

After a corresponding number (maximum m) of paths has been selected from each cluster graph, they are transformed into a human readable format.

Each path is a concatenation of a *starting path* (from start node to centroid) and an *ending path* (from centroid to end node). These sub-paths form a section of several tweets and are each replaced by one of the tweets whose sections they represent (e.g., by the last one that led to an increase of the node weights). The substitution is done using the lemmatized token sequences, which were saved during preprocessing and still contain stop words.

First, the position of the centroid in the respective token sequence is determined. Only tokens and intermediate stop words, which also belong to the partial path can be used. A better way is using an extended sequence - like, for a start path, the complete token sequence before the centroid and for an end path the complete sequence after the centroid. This variant produces clearly better results concerning readability and comprehensibility of the message and is therefore used as default setting.

At the end, the sentences of a final message are each ended by a dot, concatenated and the first letter of the token is converted into an uppercase letter, see Table 6.9.

6.3 Experiments

In this section I present some tests and experiments done for this approach.

6.3.1 Performance Test

The tests were performed with the following settings: one starts with a warm-up, in which about 900 simplified tweets from a stored file is read, processed, clustered and aggregated. Afterwards the actual time measuring starts with several tweet files. More precisely, the time needed from reading a file to the creation of the final messages is measured. This process is repeated 12 times in a row for each file, whereby the best and the worst result is discarded and the average from the remaining ten values is calculated.

The tests were performed on Intel® Xeon® Scalable Processor "Skylake" Silver 4110 with 2.10 GHz, 192GB DDR4 and 2x 4TB SATA3-HDD running Ubuntu Linux 18.04 LTS 64 bit.

The performance is measured for the previously mentioned *standard parameters*. N-grams of length 3 generated at the end of the preprocessing are used as centroids. All clusters that comprise less than 0.05% of the total number of all clustered tweets, as well as those clusters that overlap by at least 60% are discarded. Then the 10 (in terms of the number of contained tweets) largest clusters are aggregated. During the reduction of the aggregated cluster graphs, all nodes that have 1.5% or less of centroid weight are eliminated. Finally, a text with a maximum of 4 sentences each is formed for each cluster.

16 different disjoint files were tested, which contain between 27 000 and 4 300 000 English tweets. The exact numbers are shown in Table 6.10.

Besides the *file numbers* and the *file size* (in Mebibyte[25]) the number of tweets can be found in this table. This column shows on the left side (*total*) the amount of all records in the respective file and on the right side (*clustering*) the number of tweets that are clustered. The difference between the data in these two columns is due to the fact that only one of possibly multiple (re-)tweets is taken into account for clustering. On the other hand, this difference also results from the fact that not all processed tweets survive preprocessing. Especially for very short text messages or those with a high percentage of stop words, there may not be enough tokens left to form $N-$grams. If, e.g., $3-$grams are used as centroids, a tweet at the end of the preprocessing must still contain at least three tokens. Otherwise no $3-$grams can be build and the tweet is discarded before clustering. The table contains also information about the absolute

[25] 1 MiB = 2^{20} bytes

file size		tweets number		average runtime		
file	MiB	total	clustering	sec	min	ms/tweet
f1	158	27 268	27 041	5.89160	0.09819	0.21606
f2	410	74 033	63 113	11.52620	0.19210	0.15569
f3	926	165 822	143 175	25.42840	0.42381	0.15335
f4	1 061	166 382	160 776	28.62940	0.47716	0.17207
f5	919	172 507	170 575	36.98610	0.61644	0.21440
f6	981	173 358	147 577	26.47640	0.44127	0.15273
f7	986	179 608	177 294	37.28290	0.62138	0.20758
f8	1 127	182 426	174 611	32.24990	0.53750	0.17678
f9	1 554	277 741	241 272	41.54850	0.69248	0.14959
f10	2 348	408 521	352 172	61.15850	1.01931	0.14971
f11	4 348	799 004	682 682	109.06230	1.81771	0.13650
f12	5 422	962 521	821 616	135.03500	2.25058	0.14029
f13	6 634	1 160 599	1 000 558	169.80960	2.83016	0.14631
f14	18 567	3 281 852	2 788 772	461.83120	7.69719	0.14072
f15	21 338	3 721 695	3 174 048	538.39320	8.97322	0.14466
f16	24 691	4 253 183	3 628 391	609.02560	10.15043	0.14319

Table 6.10: Performance with standard parameters.

and relative execution time (in seconds and minutes and in milliseconds per tweet).

Some abnormalities in the results need to be explained. The amount of required memory is not always proportional to the total amount of the tweets included in the file. There are some possible explanations for this. For one thing, tweet texts that approximately exhaust the 280-character limit trivially require more space. On the other hand, a JSON tweet object contains more information than just the ID, text and language of a tweet (see Figure A.2 in Appendix A.1). For example, a retweet always includes its original message, so it has the size of two tweets. An additional information (e.g., location and time) also needs memory, if available. An example for such a disproportion is given in Table 6.10 by the test files *f1*, *f3* and *f5*, among others. The latter holds 172 507 tweets at a size of 919 MiB and the preceding file includes 166 382 tweets at 1 061 MiB.

Noticeable are the results for the *thematically pre-filtered* files (light grey lines in Table 6.10). The text messages stored in these rows were selected by a filter according to certain keywords. The aforementioned filtering was already carried out in advance, the current procedure is not involved. There is no other known difference between the pre-filtered and "normal" files than filtering for the measured anomalously high execution times. With sometimes more than 0.2 milliseconds per tweet, they are clearly higher than the values of their unfiltered counterparts. In order to exclude fluctuations, the tests with the pre-filtered files were repeated, which resulted in almost unchanged results. In this context it is also remarkable that the pre-filtered files each have the highest percentage (over 95%) of tweets that survive preprocessing (and are then clustered). The average value for other files is below 87% (see Table 6.11).

The pre-filtered text messages are selected with regard to current and meaningful topics, which are often tweeted more extensively (e.g., *covid-19* or *black lives matter*). It could explain why the texts of these tweets contain on average more characters than their unfiltered counterparts (178.15 vs. 119.65 characters per tweet). The longer the text, the higher the probability that it will go through the preprocessing and be clustered afterwards. Significantly larger texts would also result in a longer duration of the preprocessing. Table 6.11 contains also a column *clustering* with the percentage of tweets that participate in clustering, a column *retweets*, indicating the percentage of retweets in the file and a column *extend* with the percentage of text messages longer than 140 characters.

file size		number of	duration	tweet percentages			text length
file	MiB	tweets	ms/tweet	clustering	retweets	extended	char/tweet
f1	158	27 268	0.21606	99.17%	68.56%	58.71%	188.50
f2	410	74 033	0.15569	85.25%	58.31%	32.91%	120.23
f3	926	165 822	0.15335	86.34%	60.59%	33.22%	121.93
f4	1 061	166 382	0.17207	96.63%	85.18%	58.40%	175.79
f5	919	172 507	0.21440	98.88%	68.93%	66.31%	187.45
f6	981	173 358	0.15273	85.13%	60.03%	32.54%	120.86
f7	986	179 608	0.20758	98.71%	68.51%	62.69%	182.37
f8	1 127	182 426	0.17678	95.72%	79.43%	47.76%	156.62
f9	1 554	277 741	0.14959	86.87%	59.09%	34.18%	123.39
f10	2 348	408 521	0.14971	86.21%	62.09%	32.58%	123.12
f11	4 348	799 004	0.13650	85.44%	57.09%	30.69%	116.55
f12	5 422	962 521	0.14029	85.36%	59.50%	30.63%	118.53
f13	6 634	1 160 599	0.14631	86.21%	60.10%	32.86%	122.25
f14	18 567	3 281 852	0.14072	84.98%	60.65%	28.71%	114.86
f15	21 338	3 721 695	0.14466	85.29%	60.00%	30.05%	116.96
f16	24 691	4 253 183	0.14319	85.31%	61.20%	29.98%	117.42
average of the unfiltered files:				85.67%	59.88%	31.67%	119.65
average of teh filtered files:				97.82%	74.12%	58.77%	178.15

Table 6.11: Analysis of the used tweet files.

The very last column (*text length*) contains the information how many characters the tweets in the individual files contain on average. The data shows that the thematically pre-filtered files (light grey lines) have above-average values in all the criteria mentioned. So this seems to be the reason for the increased time needed to process the pre-filtered files.

6.3.2 Evaluation of the Aggregation

In this section I describe the results of the assessment of tweets' aggregation. These results were obtained with a survey, which was carried out in the context of Dominik Gröninger's master thesis at the University of Augsburg. The survey (in German) can be found in Appendix A.4. Since only a small number of people (10) took part

in the survey, the results serve more as an orientation on how this approach should be developed and improved further.

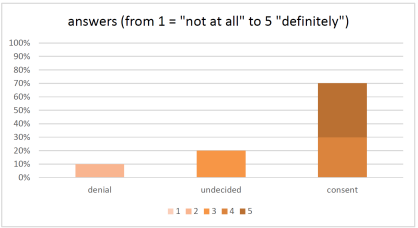
Five clusters generated by the current approach were presented to the participants. Each of them contains between 45 and 80 tweets of the same topic in English. These are the real text messages, which have only been cleaned of URLs and emojies and occasionally shortened. The task calls on the participants to read all the tweets and to write a short text as a summary for each cluster by hand. This first task aims to make each participant aware of the relevant content and to provide a basis for comparison for the second part of the survey.

The clusters were selected to simulate the query result, but at the same time not to overwhelm the participants. From a thematic point of view, the focus is primarily on current events. The first cluster with the topic *Corona Boris Johnson* includes 45 tweets. They deal with the COVID-19 disease of British Prime Minister Boris Johnson. The second cluster of 80 tweets deals with the death and the memory of the Swedish actor *Max von Sydow*, who died in March 2020. The 70 text messages of the third cluster, entitled *First day of summer*, deal with the beginning of summer. The fourth cluster, *Warren drops out*, deals with the departure of Senator Elizabeth Warren from the Democratic primaries for the US presidential election 2020 (75 tweets). The last cluster of the survey is entitled *Separating children* and contains 65 tweets dealing with the separation of children and their parents who are illegally in the US.

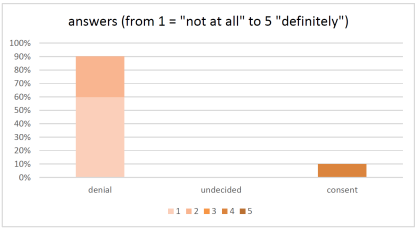
The second part of this survey presents the participants with an aggregated text generated by the current approach (*final message*). With the help of five questions each, these are to be evaluated in terms of their comprehensibility and content. In addition, a general assessment and evaluation will also be made in comparison to the summaries generated by hand.

Furthermore, the participants are asked to answer the questions that are intended to assess whether reading and aggregating a large number of tweets "by hand" is actually perceived by them as annoying.

The question whether they found *reading at least* 45 *tweets per cluster unpleasant/annoying*, 70% of participants answered positively (Figure 6.5a). 20% had no relevant opinion on this and 10% of interviewed people said that it was not a problem for them to read so many tweets as the query result. The participants could choose an answer between 1 (*not at all*) and 5 (*definitely*) on an integer scale. They were even more united about whether they would voluntarily read at least 45 search results (e.g., tweets) without having been asked to do so (Figure 6.5b). 90% disagreed on this point, with two-thirds of this number choosing the answer 1 - *not at all*. These results therefore support the assumption that users are not interested in dealing with a multitude of search results/tweets.



(a) "Did you find it uncomfortable to read at least 45 per cluster?"

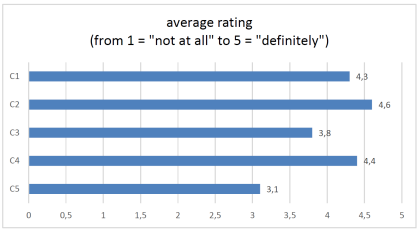


(b) "Would you also read at least 45 tweets if you were not asked to?"

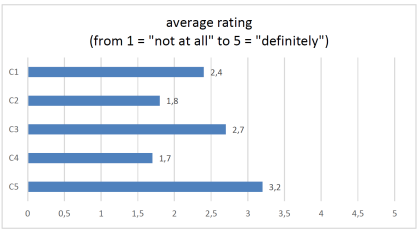
Figure 6.5: Survey results on the quality of aggregation (part 1).

In the second and third sections of the questionnaire, the participants received a summary for each cluster generated by the developed software. When asked whether these contained the *most important information* of their clusters, the participants answered rather positively for all five items (Figure 6.6a). There were again five possible answers to each question, ranging 1 (*not at all*) to 5 (*definitely*). The aggregations of clusters 1 (*Corona Boris Johnson*), 2 (*Max of Sydow*) and 4 (*Warren drops out*) get the average value in the range between 4.3 and 4.6 and thus a clear agreement from the participants. Less favourable was the aggregation of the 3rd cluster (*First day of summer*), which received an average score of 3.8. The lowest support with the average value of 3.1 received the aggregation of the fifth cluster (*Separating children*). However, the score received for this cluster also shows that the most important contents were (at least partially) recorded and aggregation can be tolerated as acceptable.

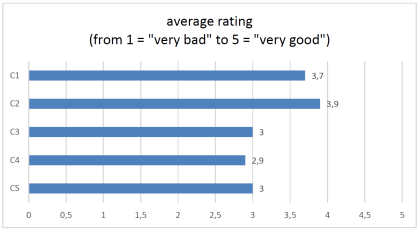
An important goal of the developed approach is, besides the collecting of the core contents, also the inclusion of further, supplementary information. Therefore, the survey participants were asked whether this had been successful or whether there was still *missing content* that should necessarily be included. The results, shown in Figure 6.6b), largely coincide with those of the previous question. For the aggregations of clusters 2 and 4, the participants agreed that these cover the most important contents. Accordingly, for these two aggregations with average scores of 1.7 and 1.8, they signalled that no mandatory information is missing. The participants were less positive about the automated summaries of cluster 1 and cluster 3, which with average scores of 2.4 and 2.7 respectively, are still below the average response of 3 indicating that some important information is missing. This is particularly interesting for the aggregation of the first cluster (*Corona Boris Johnson*), as in the previous question the participants answered that this cluster contains the most important information. The opinion on the fifth cluster is again in line with the evaluation regarding the question about the most important contents. For this cluster the participants were most



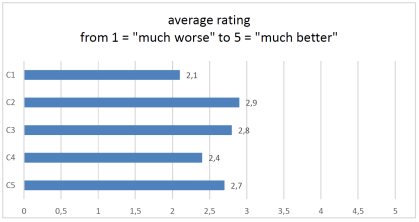
(a) "Do the individual aggregations contain the most important information of their clusters?"



(b) "Is there any content missing in the individual aggregations that should necessarily be present?"



(c) "How do you rate the individual aggregations' comprehensibility?"



(d) "How do you rate the individual aggregations compared to your own?"

Figure 6.6: Survey results on the quality of aggregation (part 2).

likely to identify shortcomings beforehand, so the tendency at this point is that more information has to be included. Overall, the question of missing essential content was answered negative in four out of five cases. Nevertheless, some information extension could be beneficial.

Apart from the content aspects, automated summaries are required to be easy to read and understand by people. For this reason, respondents were asked specifically how they assess *comprehensibility*. There were five possible answers from 1 (*very poor*) to 5 (*very good*). Contrary to expectations, the results were sobering (see Figure 6.6c): four out of the five aggregations were rated with average value of 3.0 or higher, which can be interpreted as acceptable or rather good. However, in the opinion of the participants, none of the aggregations was understandable enough to receive an average rating higher than 4.0. Furthermore, the aggregations to cluster 3 (*First day of summer*) and cluster 5 (*Separating children*) only just reached the 3.0 mark. This rather moderate assessment of comprehensibility can partly be explained by the fact that most words are reduced to their basic form, which means that there is still a need for rectification of defects.

In the final question, the survey participants were asked to assess the automatically created aggregations in *direct comparison to* their *self-written counterparts*. With answer options ranging from 1 (*much worse*) to 5 (*much better*), all generated aggregations were rated (rather) worse on average. The aggregation of the second cluster (*Max von Sydow*) was the most likely to reach the level of a hand-written summary with an average score of 2.9. The aggregation to cluster 5 (*Separating children*) received the worst average general assessment.

In general, it can be said that the automatically aggregated summaries received a predominantly acceptable or good grade from the respondents.

In this chapter I have given an overview of the approach that should summarize and aggregate preference based filtered set of tweets and procude a compact message without duplicates. After evaluation of the aggregated summaries with the real user, his approach can clearly be called promising, although it requires improvement primarily in terms of good readability of the generated message.

Chapter 7

Related Work

In this chapter I give a short overview of the papers that inspired me for the topic of my dissertation. These are the papers that deal with both *stream data* and *Twitter analysis*. Other papers I came across in my thesis were dedicated to *preferences*. And at the very end I added another topic, namely *aggregation of text messages* to my literature research.

Today, data streams are part of every area of life. The stream data processed by humans as well as computers is very large and rapidly increasing. Therefore also stream data processing is a highly relevant topic today. It is not amazing that many scientists all over the world try to process and to analyze these streams to extract important information from such continuous data flows. Many modern applications such as network monitoring, financial analysis, infrastructure manufacturing, sensor networks, meteorological observations, or social networks require query processing over data streams, cp. for example [CDTW00, BGS01, SST+09].

Research in this area has been going on for several decades. In [BBD+02] the authors motivate the need for and research issues arising from stream data processing. The paper gives an overview on past work and projects in the area of data streams, and explores topics in query languages, requirements and challenges in stream query processing. Babu and Widom [BW01], e.g., focus primarily on the problem how to define and evaluate continuous queries over data streams. Cherniack et. al [CBB+03] describe a large-scale distributed stream processing system and discusses approaches for addressing load management, high availability, and federated operation issues in such an environment. In [ABB+03] the authors investigate queries over continuous unbounded streams for applications like network monitoring, financial analysis, and sensor networks. For this they present the Stanford Data Stream Management System

(STREAM) for rapid streams and long-running queries, where the system resources may be limited.

Faria et. al [FGdCG16] describe various applications of novelty detection in data streams, and Krempl et. al [KvB+14] discuss challenges for data stream mining such as protecting data privacy, handling incomplete and delayed information, or analysis of complex data.

In addition to stream analysis, the preference queries [Kie02, Cho03, KEW11] also have received considerable attention in the past, due to their use in selecting the most preferred items, especially when the filtering criteria are contradictory. In particular, *Skyline queries* [BKS01, CGGL03, CCM13], a subset of preference queries, have been thoroughly studied by the database community to filter high relevant information from a dataset.

When dealing with Pareto-optimal objects and preferences in general, several models play an important role. For example, Kasabov and Song [KS02] and Dovžan et. al [DLŝ15] handle preferences with fuzzy values, whereas Boutilier et. al [BBD+04] use Ceteris-Paribus nets (CP-nets) do describe user wishes. Other models like Chomicki [Cho03] and Kießling [Kie02, KEW11] use strict partial orders to represent preferences in information systems and are often more flexible than other approaches.

But algorithms proposed for traditional database Skyline computation, e.g., [BKS01, GSG05, CCM13], are not appropriate for continuous data and therefore new techniques should be developed to fulfil the requirements posed by the data stream model. In [KPM10] the authors examine the characteristics of important preference queries (Skyline, top-k and top-k dominating) and review algorithms proposed for the evaluation of continuous preference queries under the sliding window streaming model. However, they do not present any framework for preference-based stream evaluation. Ribeiro et. al [RBdA+17] describe an approach for processing data streams according to temporal conditional preferences. In [LLK13], Lee et. al propose a new method for processing multiple continuous Skyline queries over a data stream.

One of the fastest growing research areas is the analysis of stream data provided by social networks. Therefore, it is not surprising that tweet analysis is a popular topic among scientists. One of the research directions, for example, is the analysis of user sentiment and public's feelings towards certain brand, business, event, etc., cp. [SHA12, SLA+17, PRPM16]. A content analysis of tweets on various topics is also often in the focus, cp. [ALA+17, CRKC+16, SVO+17]. In [SGŽ12] investigate the authors whether sentiment analysis of public mood, derived from large-scale collections of daily posts from Twitter can predict movements of stock prices.

In [SST+09] the authors try to build a news processing system from the tweets. The content of tweets are analyzed to determine if the tweet is news or not. All the messages that do not belong to the news domain must be removed. The remaining

data will be clustered and introduced to the system into topics. Then authors try to locate and extract geographic content from each cluster.

In [OM09] the authors describe an event notification system that monitors and delivers semantically relevant tweets if these meet the user's information needs. As an example they construct an earthquake prediction system targeting Japanese tweets. For their system they use keywords, the number of words, and the context of an event.

The problem addressing in [RM13] is to determine the popularity of social events (concerts, festivals, sport events, conferences, etc.) based on their presence in Twitter. Knowing the popularity of an event can help in improving the organization of the infrastructure in the area of the event's location (more public transport by the high popularity), or in alerting the event's organizers for the need of better promotion (low popularity).

In [PK13] the authors describe a system to detect events from tweets. They research their textual and temporal characteristics. The most important components are: an extraction scheme for event representative keywords, a mechanism to store their appearance patterns, and a hierarchical clustering technique.

For solving the last task of this work (summarizing the text messages), there exist multiple possibilities: from very simple that use label or keywords as some kind of summary (cp. [OKA10]) to more complex that choose one representative of the cluster with the most important information (see, e.g., [TO09]). Since I wanted to provide the user with result in form of a full text with lots of details rather than a general headline, but at the same time I did not want to get involved with neuronal networks because they require a huge amount of training data and are not well suited for real-time analysis, the approach described by Sharifi, Inouye and Kalita in [SIK14] seemed to me to be the best solution. The main idea of the *Phrase Reinforcement* algorithm is to display the words from text messages as nodes in a graph. These nodes have certain weights, depending on how often they appear in the text collection. Walking through the paths of this graph, it is possible to restore every text message. Some of these paths are selected, converted back into the texts and delivered to the user.

The literature sources listed in this chapter are only a small part of the work that scientists around the world have produced on the mentioned topics. There are certainly other interesting approaches and ideas. But at this particular moment in time, these works are those that have inspired my own research and led to this work.

Chapter 8

Conclusion

In this chapter I summarize the work I have done and outline the tasks that are still open issues.

Today, data processed by humans as well as computers is very large, rapidly increasing and often in form of data streams. Users want to analyze this data to extract personalized and customized information in order to learn from this ever-growing amount of data. Therefore stream data processing is a highly relevant topic today.

In this work I have described a framework that analyses stream data w.r.t. user preferences. As data source I used tweets. Twitter allows free access to these objects, but tweets also have a lot of attributes that are well suited for preference analysis. It is also a very popular social network with hundreds of millions of users that produce a lot of data every second, which has made it a good candidate for my purposes.

The choice of Twitter as a data source had a great impact on my work. The core problem and its solution are applicable to stream data in general, but many decisions and details were influenced by the character of Twitter and its objects. This includes, for example, the design and development of a *new categorical basic preference CONTAINS*, which can be applied to the freely written texts. The existing categorical preferences were well suited for the attributes with the finite domains, but could hardly be helpful for the text messages, such as tweets. The tweets have many attributes to which the preferences can be applied. This helps to reduce the data stream and adapt the results to the user's interests. But the possibility to select the tweets directly was what I really missed. This led to the development and implementation of a new preference that works with the text messages and can evaluate them in terms of their content.

Twitter produces the data every second and its volume is growing. So it was clear

to me right from the start of my work that I not only need a new concept that would allow me to evaluate the stream data with the preferences, but that this should happen efficiently and in real time. This is why the Stream Lattice Skyline algorithm was developed, which does not depend on object-to-object comparison and uses the lattice structure constructed by a Pareto query. This algorithm was developed for efficient evaluation of Pareto preferences on streams and can be used for any type of data with lowe cardinality domains, provided that the data can be mapped to natural numbers. The extensive tests with different data have confirmed that Stream Lattice Skyline can handle the Pareto requests efficiently.

The final step in the framework was also inspired by the character of the Twitter data. The tweets are short texts that often contain only partial information. In addition, Twitter allows the so-called reposts, when the original message is posted by other users without any changes. This results in many duplicates, which are also present in the amount filtered by preferences. As far as the result set is concerned, it often consists of a relatively large number of text messages that are not very comfortable to read. This led to the idea of *aggregating the tweets* resulting after the preference evaluation and presenting them to the user in the form of a compact text. It should be noted that both the new CONTAINS preference and the aggregation of text messages are suitable as concepts for any type of short text. But implementation is realised tweets specifically and takes into account such special units as hashtags or user references that are missing in the other texts.

I would like to explore some open questions further in the future. These include, for example, the approach for *finding the misspelled words* in tweets implemented in the CONTAINS preference. The experiments have shown that the current implementation delivers more false positive results than true positive ones. I also want to continue working on the aggregation of tweets, especially to *improve the readability of the final message*. But there is still some work to be done on the *selection of the information* that will find its place in the final message. The developed Stream Lattice Skyline algorithm can also be extended. For example, the *parallelisation of some tasks* would be conceivable.

In general, it can be said that every scientific problem that seems to have been solved poses a whole series of new challenges that can always be worked on. And that is what makes this field so exciting.

Bibliography

[AB15] Elmasri R. A. and Navathe S. B., *Fundamentals of Database Systems (7th Edition)*, Pearson, 2015.

[ABB+03] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom, *STREAM: The Stanford Stream Data Manager*, SIGMOD '03 (New York, USA), ACM, 2003, pp. 665–665.

[ALA+17] J. W. Ayers, E. C. Leas, J. Allem, A. Benton, M. Dredze, B. M. Althouse, T. B. Cruz, and J. B. Unger, *Why do People Use Electronic Nicotine Delivery Systems (Electronic Cigarettes)? A Content Analysis of Twitter, 2012-2015*, PLOS ONE **12** (2017), no. 3, 1–8.

[Arr59] K. J. Arrow, *Rational Choice Functions and Orderings*, Economica **26** (1959), no. 102, 121–127.

[AW00] R. Agrawal and E. L. Wimmers, *A Framework for Expressing and Combining Preferences*, SIGMOD Rec. **29** (2000), no. 2, 297–306.

[BBD+02] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, *Models and Issues in Data Stream Systems*, Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (New York, NY, USA), PODS '02, ACM, 2002, pp. 1–16.

[BBD+04] C. Boutilier, R. I. Brafman, C. Domshlak, H. H. Hoos, and D. Poole, *CP-nets: A Tool for Representing and Reasoning with Conditional Ceteris Paribus Preference Statements*, J. Artif. Intell. Res. **21** (2004), 135–191.

[Bea20] A. Beaulieu, *Learning SQL: Generate, Manipulate, and Retrieve Data Taschenbuch*, O'Reilly UK Ltd., 2020.

[BGS01] P. Bonnet, J. Gehrke, and P. Seshadri, *Towards Sensor Database Systems*, MDM '01 (London, UK), Springer-Verlag, 2001, pp. 3–14.

[BKS01] S. Börzsönyi, D. Kossmann, and K. Stocker, *The Skyline Operator*, Proceedings of ICDE '01 (Washington, USA), IEEE Computer Society, 2001, pp. 421–430.

[BW01] S. Babu and J. Widom, *Continuous Queries over Data Streams*, SIGMOD Rec. **30** (2001), no. 3, 109–120.

[CBB+03] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, Y. Cetintemel, U.and Xing, and S. Zdonik, *Scalable Distributed Stream Processing*, CIDR 2003 - First Biennial Conference on Innovative Data Systems Research (Asilomar, CA), January 2003.

[CCM13] J. Chomicki, P. Ciaccia, and N. Meneghetti, *Skyline Queries, Front and Back*, SIGMOD **42** (2013), no. 3, 6–18.

[CDTW00] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang, *NiagaraCQ: A Scalable Continuous Query System for Internet Databases*, SIGMOD '00 (New York, USA), ACM, 2000, pp. 379–390.

[CGGL03] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, *Skyline with Presorting*, Proceedings of ICDE '03, 2003, pp. 717–816.

[Cho02] J. Chomicki, *Querying with Intrinsic Preferences*, Proceedings of the 8th International Conference on Extending Database Technology: Advances in Database Technology (Berlin, Heidelberg), EDBT '02, Springer-Verlag, 2002, pp. 34–51.

[Cho03] ———, *Preference Formulas in Relational Queries*, ACM Trans. Database Syst. **28** (2003), no. 4, 427–466.

[CM12] G. Cugola and A. Margara, *Processing Flows of Information: From Data Stream to Complex Event Processing*, ACM Comput. Surv. **44** (2012), no. 3, 15:1–15:62.

[CRKC+16] P. Cavazos-Rehg, M. Krauss, S. Costello, S. Connolly, C. Rosas, M. Bharadwaj, and L. Bierut, *A Content Analysis of Depression-related Tweets*, Computers in Human Behavior **54** (2016), 351–357.

[CWB+11] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, *Natural Language Processing (Almost) from Scratch*, The Journal of Machine Learning Research **12** (2011), 2493–2537.

[Dam64] F. J. Damerau, *A Technique for Computer Detection and Correction of Spelling Errors*, ACM **7** (1964), no. 3, 171–176.

[DLŜ15] D. Dovžan, Vito Logar, and Igor Škrjanc, *Implementation of an Evolving Fuzzy Model (eFuMo) in a Monitoring System for a Waste-Water Treatment Process*, IEEE Trans. Fuzzy Systems **23** (2015), no. 5, 1761–1776.

[DP02] B. A. Davey and H. A. Priestley, *Introduction to Lattices and Order [2nd ed.]*, Cambridge University Press, Cambridge, 2002.

[DP18] M. Dayarathna and S. Perera, *Recent Advancements in Event Processing*, ACM Comput. Surv. **51** (2018), no. 2, 33:1–33:36.

[DPE08] S. Döring, T. Preisinger, and M. Endres, *Advanced Preference Query Processing for E-Commerce*, SAC '08: Proceedings of the 2008 ACM symposium on Applied computing (New York, NY, USA), ACM, 2008, pp. 1457–1462.

[DPH09] Manning C. D., Raghavan P., and Schütze H., *Introduction to Information Retrieval*, online edition ed., Cambridge University Press, Cambridge, England, apr 2009.

[ECM17] ECMA-404, *ECMA-404 International Standard: The JSON Data Interchange Syntax*, Tech. report, ECMA Internation, 2017.

[ECM19] ECMA-262, *ECMAScript 2019 Language Specification*, Tech. report, ECMA International, 2019.

[EK14] M. Endres and W. Kießling, *High Parallel Skyline Computation over Low-Cardinality Domains*, Proceedings of ADBIS '14, Springer, 2014, pp. 97–111.

[EK16] M. Endres and W. Kießling, *Parallel Skyline Computation Exploiting the Lattice Structure*, Journal of Database Management (JDM) **26** (2016), no. 4, 18–43.

[EKR18] M. Endres, J. Kastner, and L. Rudenko, *Analyzing and clustering Pareto-optimal objects in data streams*, Learning from Data Streams in Evolving Environments: Methods and Applications (M. Sayed-Mouchaweh, ed.), Springer-Verlag, 2018.

[End11] M. Endres, *Semi-Skylines and Skiline Snippets*, Ph.D. thesis, University of Augsburg, 2011.

[EP17] M. Endres and T. Preisinger, *Beyond Skylines: Explicit Preferences*, Proceeding of DASFAA '17 (Cham), Springer International Publishing, 2017, pp. 327–342.

[ER18] M. Endres and L. Rudenko, *A Tour of Lattice-Based Skyline Algorithms*, In M. K. Habib (Eds.): Emerging Investigation in Artificial Life Research and Development, IGI Global, 2018.

[FGdCG16] E. R. Faria, I. J. C. R. Gonçalves, A. C. P. L. F. de Carvalho, and J. Gama, *Novelty detection in data streams*, Artificial Intelligence Review **45** (2016), no. 2, 235–269.

[Fli] Flink, *Flink DataStream API Programming Guide*.

[GL94] T. Gaasterland and J. Lobo, *Qualified Answers That Reflect User Needs and Preferences*, VLDB (Santiago de Chile, Chile), 1994.

[GO03] L. Golab and M. T. Özsu, *Issues in Data Stream Management*, SIGMOD Rec. **32** (2003), no. 2, 5–14.

[GR09] M. Golfarelli and S. Rizzi, *Expressing OLAP Preferences*, SSDBM '09 (Berlin, Heidelberg), SSDBM 2009, Springer-Verlag, 2009, pp. 83–91.

[GSG05] P. Godfrey, R. Shipley, and J. Gryz, *Maximal Vector Computation in Large Data Sets*, Proceedings of VLDB '05, VLDB Endowment, 2005, pp. 229–240.

[HK05] B. Hafenrichter and W. Kießling, *Optimization of Relational Preference Queries*, Proceedings of ADC '05 (Darlinghurst, Australia), 2005, pp. 175–184.

[HKP01] V. Hristidis, N. Koudas, and Y. Papakonstantinou, *PREFER: A System for the Efficient Execution of Multi-parametric Ranked Queries*, SIGMOD Rec. **30** (2001), no. 2, 259–270.

[JZMG15] X. Junchang, W. Zhiqiong, B. Mei, and W. Guoren, *Reverse Skyline Computation over Sliding Windows*, Mathematical Problems in Engineering **2015** (2015), 1–19.

[KEW11] W. Kießling, M. Endres, and F. Wenzel, *The Preference SQL System - An Overview*, Bulletin of the Technical Commitee on Data Engineering, IEEE Computer Society **34** (2011), no. 2, 11–18.

[KH19] V. Kalavri and F. Hueske, *Stream Processing with Apache Flink*, O'Reilly Media, Inc., Sebastopol, California, USA, 2019.

[Kie02] W. Kießling, *Foundations of Preferences in Database Systems*, Proceedings of the 28th International Conference on Very Large Data Bases, VLDB '02, VLDB Endowment, 2002, pp. 311–322.

[Kie05] ———, *Preference Queries with SV-Semantics*, COMAD '05 (Goa, India), Computer Society of India, 2005, pp. 15–26.

[KPM10] M. Kontaki, A. N. Papadopoulos, and Y. Manolopoulos, *Continuous Processing of Preference Queries in Data Streams*, Proceedings of SOFSEM '10 (Spindleruv Mlýn, Czech Republic), Springer Berlin Heidelberg, 2010, pp. 47–60.

[KR93] R. L. Keeney and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Trade-Offs*, Cambridge University Press, 1993.

[Krä07] J. Krämer, *Countinuous Queries over Data Streams - Semantics and Implementation*, Ph.D. thesis, Philipps Universität Marburg, 2007.

[KS02] N. K. Kasabov and Q. Song, *DENFIS: Dynamic Evolving Neural-Fuzzy Inference System and its Application for Time-Series Prediction*, IEEE Trans. Fuzzy Systems **10** (2002), no. 2, 144–154.

[Kuk92] K. Kukich, *Techniques for Automatically Correcting Words in Text*, ACM Comput. Surv. **24** (1992), no. 4, 377–439.

[KvB+14] G. Krempl, I. Žliobaite, D. Brzeziński, E. Hüllermeier, M. Last, V. Lemaire, T. Noack, A. Shaker, S. Sievi, M. Spiliopoulou, and J. Stefanowski, *Open Challenges for Data Stream Mining Research*, SIGKDD Explor. Newsl. **16** (2014), no. 1, 1–10.

[Lid01] E. D. Liddy, *Natural Language Processing.*

[LLK13] Y. W. Lee, K. Y Lee, and M. H. Kim, *Efficient Processing of Multiple Continuous Skyline Queries over a Data Stream*, Information Science **221** (2013), 316–337.

[LM11] S. Linckels and C. Meinel, *Natural Language Processing*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[Luc01] D. C. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[Mel93] J. Melton, *Understanding the New SQL: A Complete Guide*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[MKEK15] S. Mandl, O. Kozachuk, M. Endres, and W. Kießling, *Preference Analytics in EXASolution* , Proceedings of BTW, BTW'15, 2015.

[MPG07] M. Morse, J. M. Patel, and W. I. Grosky, *Efficient Continuous Skyline Computation*, Inf. Sci. **177** (2007), no. 17, 3411–3437.

[MPJ07] M. Morse, J. M. Patel, and H. V. Jagadish, *Efficient Skyline Computation over Low-cardinality Domains*, Proceedings of VLDB '07, 2007, pp. 267–278.

[MS98] J. McMahon and F. J. Smith, *A Review of Statistical Language Processing Techniques*, Artificial Intelligence Review **12** (1998), no. 5, 347–391.

[MSM93] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, *Building a Large Annotated Corpus of English: The Penn Treebank*, Computational Linguistics **19** (1993), no. 2, 313–330.

[Nor13] P. Norvig, *English Letter Frequency Counts: Mayzner Revisited or ETAOIN SRHLDCU*, 2013.

[OKA10] B. O'Connor, M. Krieger, and D. Ahn, *TweetMotif: Exploratory Search and Topic Summarization for Twitter*, Proceedings of the International AAAI Conference on Weblogs and Social Media (Washington, DC, USA), 2010.

[OM09] M. Okazaki and Y. Matsuo, *Semantic Twitter: Analyzing Tweets for Real-Time Event Notification*, BlogTalk, Lecture Notes in Computer Science, vol. 6045, Springer, 2009, pp. 63–74.

[Pet86] J. L. Peterson, *A Note on Undetected Typing Errors*, ACM **29** (1986), no. 7, 633–637.

[PK07] T. Preisinger and W. Kießling, *The Hexagon Algorithm for Pareto Preference Queries*, Proceedings of the 3rd Multidisciplinary Workshop on Advances in Preference Handling in conjunction with VLDB '07 (Vienna, Austria), 2007.

[PK13] R. Parikh and K. Karlapalem, *ET: Events from Tweets*, WWW (Companion Volume), International World Wide Web Conferences Steering Committee / ACM, 2013, pp. 613–620.

[Por80] M. F. Porter, *An Algorithm for Suffix Stripping*, Program **40** (1980), 211–218.

[Pre09] T. Preisinger, *Graph-based Algorithms for Pareto Preference Query*, Books on Demand, 2009.

[PRPM16] V. S. Pagolu, K. N. Reddy, G. Panda, and B. Majhi, *Sentiment analysis of Twitter data for predicting stock market movements*, International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES), oct 2016, pp. 1345–1350.

[RBdA+17] M. R. Ribeiro, M. C. N. Barioni, S. de Amo, C. Roncancio, and C. Labbé, *Reasoning with Temporal Preferences over Data Streams*, Florida Artificial Intelligence Research Society Conference (FLAIRS '17) (Marco Island, USA), 2017.

[RE17] L. Rudenko and M. Endres, *Personalized Stream Analysis with PreferenceSQL*, Datenbanksysteme für Business, Technologie und Web - BTW 2017, Workshopband (Stuttgart, Germany) (B. Mitschang, N. Ritter, H. Schwarz, M. Klettke, A. Thor, O. Kopp, and M. Wieland, eds.), LNI, vol. P-266, GI, 2017, pp. 181–184.

[RE18] ———, *Real-Time Skyline Computation on Data Streams*, New Trends in Databases and Information Systems - ADBIS 2018 (Budapest, Hungary) (A. Benczúr, B. Thalheim, T. Horváth, S. Chiusano, T. Cerquitelli, C. István Sidló, and P. Z. Revesz, eds.), Communications in Computer and Information Science, vol. 909, Springer, 2018, pp. 20–28.

[REH+12] P. Roocks, M. Endres, A. Huhn, W. Kießling, and S. Mandl, *Design and Implementation of a Framework for Context-Aware Preference Queries*, Journal of Computing Science and Engineering (JCSE) **6** (2012), no. 4, 243–256.

[REMK12] P. Roocks, M. Endres, S. Mandl, and W. Kießling, *Composition and Efficient Evaluation of Context-Aware Preference Queries*, DASFAA '12: Proceedings of the 17th international conference on Database systems for advanced applications, 2012.

[RERK16] L. Rudenko, M. Endres, P. Roocks, and W. Kießling, *A Preference-based Stream Analyzer*, Proceedings of the Workshop on Large-scale Learning from Data Streams in Evolving Environments - STREAMEVOLV 2016 (Riva del Garda, Italy) (M. Sayed Mouchaweh, H. Bouchachia, J. Gama, and R. Paula Ribeiro, eds.), CEUR Workshop Proceedings, vol. 2069, CEUR-WS.org, 2016.

[RH20] L. Rudenko and C. Haas, *Preference-based Twitter Analytics*, Digital ECAI 2020, M-PREF'20 Workshop (Santiago de Compostela, Spain), 2020.

[RHE20] L. Rudenko, C. Haas, and M. Endres, *Analyzing Twitter Data with Preferences*, ADBIS 2020 (Lyon, France), 2020.

[RM13] C. Railean and A. Moraru, *Discovering Popular Events From Tweets*, Conference on Data Mining and Data Warehouses (SiKDD) (2013).

[SGŽ12] J. Smailović, M. Grčar, and M. Žnidaršič, *Sentiment Analysis on Tweets in a Financial Domain*, 4th Jozef Stefan International Postgraduate School Students Conference (2012), 169–175.

[SHA12] H. Saif, Y. He, and H. Alani, *Semantic Sentiment Analysis of Twitter*, The Semantic Web - ISWC 2012 (Berlin, Heidelberg) (P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J. X. Parreira, J. Hendler, G. Schreiber, A. Bernstein, and E. Blomqvist, eds.), Springer Berlin Heidelberg, 2012, pp. 508–524.

[SIK14] B. P. Sharifi, D. I. Inouye, and J. K. Kalita, *Summarization of Twitter Microblogs*, The Computer Journal **57** (2014), no. 3, 378–402.

[SKE11] K. Stefanidis, G. Koutrika, and E. Pitoura E., *A Survey on Representation, Composition and Application of Preferences in Database Systems*, ACM TODS **36** (2011), no. 3, 19:1–19:45.

[SL18] A. Samir and Z. Lahbib, *Stemming and Lemmatization for Information Retrieval Systems in Amazigh Language*, Big Data, Cloud and Applications - Third International Conference (BDCA), Kenitra, Morocco, apr 2018, pp. 222–233.

[SLA+17] V. Subramaniyaswamy, R. Logesh, M. Abejith, S. Umasankar, and A. Umamakeswari, *Sentiment Analysis of Tweets for Estimating Criticality and Security of Events*, Journal of Organizational and End User Computing **29** (2017), 51–71.

[SOM13] T. Sakaki, M. Okazaki, and Y. Matsuo, *Tweet Analysis for Real-Time Event Detection and Earthquake Reporting System Development*, IEEE Trans. on Knowl. and Data Eng. **25** (2013), no. 4, 919–931.

[SST+09] J. Sankaranarayanan, H. Samet, B. E. Teitler, M. D. Lieberman, and J. Sperling, *Twitterstand: News in Tweets*, ACM '09, 2009, pp. 42–51.

[SVO+17] J. Sutton, S. Vos, M. Olson, C. Woods, E. Cohen, C. Gibson, N. Phillips, J. Studts, J. Eberth, and C. Butts, *Lung Cancer Messages on Twitter: Content Analysis and Evaluation*, Journal of the American College of Radiology **15** (2017).

[TO09] H. Takamura and M. Okumura, *Text Summarization Model Based on the Budgeted Median Problem*, Proceedings of the 18th ACM Conference on Information and Knowledge Management (Hong Kong, China), CIKM '09, ACM, 2009, pp. 1589–1592.

[TP06] Y. Tao and D. Papadias, *Maintaining Sliding Window Skylines on Data Streams*, IEEE Trans. on Knowl. and Data Eng. **18** (2006), no. 3, 377–391.

[Wal] J. Walton, *Twitter vs. Facebook vs. Instagram: What's the Difference?*

[WEMK12] F. Wenzel, M. Endres, S. Mandl, and W. Kießling, *Complex Preference Queries Supporting Spatial Applications for User Groups*, PVLDB **5** (2012), no. 12, 1946–1949.

[WK02] Kießling W. and G. Köstler, *Preference SQL — Design, Implementation, Experiences*, VLDB '02: Proceedings of the 28th International Conference on Very Large Databases (San Francisco), Morgan Kaufmann, 12 2002, pp. 990–1001.

[WKK13] F. Wenzel, D. Köppl, and W. Kießling, *Interactive Toolbox for Spatial-Textual Preference Queries*, Advances in Spatial and Temporal Databases (Berlin, Heidelberg) (M. A. Nascimento, T. Sellis, R. Cheng, J. Sander, Y. Zheng, H. Kriegel, M. Renz, and C. Sengstock, eds.), Springer Berlin Heidelberg, 2013, pp. 462–466.

[XYWH05] L. Xuemin, Y. Yidong, W. Wei, and L. Hongjun, *Stabbing the Sky: Efficient Skyline Computation over Sliding Windows*, Proceedings of ICDE '05 (Washington, DC, USA), IEEE Computer Society, 2005, pp. 502–513.

Appendix A

A.1 Tweet Object

a) Tweet (which is actually a retweet) on the web site in the user's feed:

Figure A.1: (Re)tweet screenshot by the user Dill Peekle.

b) Tweet from the Figure A.1 in JSON format:


```
{
"created_at":"Tue Jun 02 16:55:19 +0000 2020",
"id":1267862371076321281,
"id_str":"1267862371076321281",
"text":"RT @StephenKing: Dear fundamentalist Christian Trump supporters:
        If Obama had held the Bible backwards and upside down,
        you would immediate ...",
"source":"<a href=\"http://twitter.com/download/iphone\"
          rel=\"nofollow\">Twitter for iPhone<\/a>",
"truncated":false,
"in_reply_to_status_id":null,
"in_reply_to_status_id_str":null,
"in_reply_to_user_id":null,
"in_reply_to_user_id_str":null,
"in_reply_to_screen_name":null,

"user":
  {
  "id":3041644622,
  "id_str":"3041644622",
  "name":"Dill Peekle",
  "screen_name":"dont_do_krugs",
  "location":null,
  "url":null,
  "description":"Parker Kruger ; TXST '22 ; she/her ;
  it's all because of my no-good-dirty-rotten-pig-stealing-
  great-great-grandfather",
  "translator_type":"none",
  "protected":false,
  "verified":false,
  "followers_count":273,
  "friends_count":321,
  "listed_count":1,
  "favourites_count":27608,
  "statuses_count":12794,
  "created_at":"Wed Feb 25 17:32:48 +0000 2015",
  "utc_offset":null,
  "time_zone":null,
  "geo_enabled":false,
  "lang":null,
  "contributors_enabled":false,
  "is_translator":false,
  "profile_background_color":"000000",
  "profile_background_image_url":
  "http://abs.twimg.com/images/themes/theme1/bg.png",
  "profile_background_image_url_https":
  "https://abs.twimg.com/images/themes/theme1/bg.png",
  "profile_background_tile":false,
  "profile_link_color":"E81C4F",
  "profile_sidebar_border_color":"000000",
  "profile_sidebar_fill_color":"000000",
  "profile_text_color":"000000",
  "profile_use_background_image":false,
  "profile_image_url":
  "http://pbs.twimg.com/profile_images/1121891532146126848/FY2XlnTa_normal.jpg",
  "profile_image_url_https":
  "https://pbs.twimg.com/profile_images/1121891532146126848/FY2XlnTa_normal.jpg",
```

```
    "profile_banner_url":
    "https://pbs.twimg.com/profile_banners/3041644622/1585288414",
    "default_profile":false,
    "default_profile_image":false,
    "following":null,
    "follow_request_sent":null,
    "notifications":null
    },

  "geo":null,
  "coordinates":null,
  "place":null,
  "contributors":null,

  "retweeted_status":
    {
    "created_at":"Tue Jun 02 10:45:01 +0000 2020",
    "id":1267769182843789312,
    "id_str":"1267769182843789312",
    "text":"Dear fundamentalist Christian Trump supporters:
           If Obama had held the Bible backwards and upside down,
           you would im... https://t.co/0L9s99P5SH",
    "source":"<a href=\"https://mobile.twitter.com\"
             rel=\"nofollow\">Twitter Web App</a>",
    "truncated":true,
    "in_reply_to_status_id":null,
    "in_reply_to_status_id_str":null,
    "in_reply_to_user_id":null,
    "in_reply_to_user_id_str":null,
    "in_reply_to_screen_name":null,

    "user":
          {
          "id":2233154425,
          "id_str":"2233154425",
          "name":"Stephen King",
          "screen_name":"StephenKing",
          "location":null,
          "url":"http://stephenking.com",
          "description":"Author",
          "translator_type":"none",
          "protected":false,
          "verified":true,
          "followers_count":5901407,
          "friends_count":118,
          "listed_count":14281,
          "favourites_count":25,
          "statuses_count":4706,
          "created_at":"Fri Dec 06 15:26:35 +0000 2013",
          "utc_offset":null,
          "time_zone":null,
          "geo_enabled":false,
          "lang":null,
          "contributors_enabled":false,
          "is_translator":false,
          "profile_background_color":"C0DEED",
          "profile_background_image_url":
          "http://abs.twimg.com/images/themes/theme1/bg.png",
          "profile_background_image_url_https":
          "https://abs.twimg.com/images/themes/theme1/bg.png",
```

```
        "profile_background_tile":false,
        "profile_link_color":"1DA1F2",
        "profile_sidebar_border_color":"C0DEED",
        "profile_sidebar_fill_color":"DDEEF6",
        "profile_text_color":"333333",
        "profile_use_background_image":true,
        "profile_image_url":
        "http://pbs.twimg.com/profile_images/378800000836981162/
         b683f7509ec792c3e481ead332940cdc_normal.jpeg",
        "profile_image_url_https":
        "https://pbs.twimg.com/profile_images/378800000836981162/
        b683f7509ec792c3e481ead332940cdc_normal.jpeg",
        "default_profile":true,
        "default_profile_image":false,
        "following":null,
        "follow_request_sent":null,
        "notifications":null
        },

"geo":null,
"coordinates":null,
"place":null,
"contributors":null,
"is_quote_status":false,

"extended_tweet":
        {
        "full_text":"Dear fundamentalist Christian Trump supporters:
                     If Obama had held the Bible backwards and upside down,
                     you would immediately have called him the Antichrist.",
        "display_text_range":[0,156],

        "entities":
          {
          "hashtags":[],
          "urls":[],
          "user_mentions":[],
          "symbols":[]
          }
        },

"quote_count":706,
"reply_count":1820,
"retweet_count":20656,
"favorite_count":92073,

"entities":
        {
        "hashtags":[],
        "urls":[{"url":"https://t.co/0L9s99P5SH",
                 "expanded_url":
                 "https://twitter.com/i/web/status/1267769182843789312",
                 "display_url":"twitter.com/i/web/status/1\u2026",
                 "indices":[117,140]}],
        "user_mentions":[],
        "symbols":[]
        },
```

```
   "favorited":false,
   "retweeted":false,
   "filter_level":"low",
   "lang":"en"
   },

"is_quote_status":false,
"quote_count":0,
"reply_count":0,
"retweet_count":0,
"favorite_count":0,

"entities":
   {
   "hashtags":[],
         "urls":[],
         "user_mentions":[{"screen_name":"StephenKing",
                           "name":"Stephen King",
                           "id":2233154425,
                                   "id_str":"2233154425",
                                   "indices":[3,15]}],
         "symbols":[]
         },

"favorited":false,
"retweeted":false,
"filter_level":"low",
"lang":"en",
"timestamp_ms":"1591116919665"
}
```


Figure A.2: JSON representation of the (re)tweet from Figure A.1

A.2 Default List of Stopwords

1 &
2 a
3 about
4 above
5 actually
6 after
7 again
8 against
9 all
10 already
11 also
12 am
13 an
14 and
15 another
16 any
17 anyway
18 are
19 as
20 at
21 back
22 be
23 because
24 been
25 before
26 being
27 below
28 between
29 both
30 but
31 by
32 c
33 can
34 cause
35 cha
36 could
37 did
38 do
39 does
40 doing
41 done
42 down
43 duh
44 during
45 each
46 even
47 ever
48 every
49 few
50 for
51 from
52 further
53 get
54 go
55 goes
56 gonna
57 gunna
58 had
59 haha
60 has
61 have
62 having
63 he
64 hello
65 her
66 here
67 hers
68 herself
69 hey
70 hi
71 him
72 himself
73 his
74 how
75 however
76 huh
77 i
78 if
79 in
80 into
81 is
82 it
83 its
84 itself
85 just
86 least
87 less
88 like
89 made
90 make
91 many
92 may
93 me
94 might
95 more
96 most
97 must
98 my
99 myself
100 never
101 no
102 nor
103 not
104 now
105 of
106 off
107 oh
108 on
109 once
110 one
111 only
112 or
113 other
114 ought
115 our
116 ours
117 ourselves
118 out
119 over
120 own
121 please
122 pls
123 put
124 r
125 said
126 same
127 say
128 says
129 see
130 seen
131 shall
132 she
133 should
134 since
135 so
136 some
137 still
138 such
139 take
140 than
141 that
142 the
143 their
144 theirs
145 them
146 themselves
147 then
148 there
149 these
150 they
151 this
152 those
153 through
154 to
155 too
156 til
157 u
158 under
159 until
160 up
161 ur
162 us
163 v
164 very
165 was
166 way
167 we
168 well
169 were
170 what
171 when
172 where
173 whether
174 which
175 while
176 who
177 whom
178 why
179 will
180 with
181 would
182 y
183 ya
184 yall
185 yeah
186 yet
187 you
188 your
189 yours
190 yourself
191 yourselves

A.3 Short Description of Selected Tweet Attributes

attribute	description
`coordinates`	Very precise geographical position of the user at the moment when he posted the current tweet. Unfortunately this value is almost always `null`.
`created_at`	Date and time in UTC format the account was created.
`description`	Description of the own account entered by the user.
`favourites_count`	Number of tweets, which the user liked.
`followers_count`	Number of followers for the current account.
`friends_count`	Number of users the current account is following.
`hashtags`	Word or phrase preceded by a hash sign (#) to identify messages on a specific topic.
`lang`	Machine-detected language of the tweet text.
`location`	User-defined location of his account, not always a real location, is often `null`.
`name`	Name of the user chosen by him, not unique.
`place`	Place, which is associated with the current tweet, is often `null`.
`screen_name`	Name that identifies the current account, unique.
`source`	Utility used to post the tweet.
`statuses_count`	Number of tweets (including retweets) posted by the user.
`text`	Text of the current tweet.
`user`	Author of the tweet.
`verified`	Set to `true` means that the account of public interest is authentic.

Table A.1: Alphabetically sorted selected attributes of tweet object with short description.

A.4 Survey

Umfrage zur Masterarbeit 19.04.2020

Entwicklung und Implementierung von Verfahren zur Aggregation thematisch ähnlicher Twitter-Nachrichten

von Dominik Gröninger

Im Rahmen meiner Masterarbeit erarbeite ich Methoden, um Kurznachrichten mit ähnlicher Thematik zu gruppieren und deren Inhalt zu aggregieren (zusammenzufassen). Ziel ist es dabei, menschenlesbare Inhaltsangaben zu erzeugen, die sich auf wenige Sätze beschränken, aber dennoch möglichst viele Informationen wiedergeben. Um die Qualität der automatisiert generierten Aggregationen einstufen zu können, muss ein Vergleich mit von Hand verfassten Pendants stattfinden. Hierbei bitte Ich Sie um Ihre Mithilfe! Vielen Dank im Voraus!

Worum es geht:

- In den unten stehenden Tabellen finden Sie eine Reihe von Textnachrichten (**Tweets**), die im Sozialen Netzwerk **Twitter** veröffentlicht wurden. Diese Nachrichten sind auf Englisch verfasst und stammen von Privatpersonen, Institutionen (z.B. Rundfunkanstalten), etc. Entsprechend können einerseits sachliche Fakten, andererseits aber auch Meinungen, Gedanken und Emotionen sowie Rechtschreibfehler enthalten sein. Zur Verbesserung der Lesbarkeit wurden URLs und Emojies entfernt, Nutzerverlinkungen (@...) und Hashtags (#...) sind aber nach wie vor vorhanden.
- Jede Tabelle steht für eine Gruppierung (**Cluster**) von Tweets, die mithilfe der entwickelten Software entstanden ist. In der Folge behandeln die enthaltenen Tweets auch mutmaßlich das gleiche Gesprächsthema, welches jeweils als Überschrift angegeben ist. Die Spalte *Nr.* dient Ihnen zur Orientierung, die Spalte *Tweet-ID* können Sie ignorieren.

Aufgabenstellung:

- Fassen Sie die (wichtigsten) Inhalte jedes Clusters in 1 – 5 Sätzen (auf Englisch) zusammen,
- aber versuchen Sie auch, möglichst viele vorhandene Informationen mit einzubinden!

  Salopp: Geben Sie in gekürzt wieder, was Sie einer Vielzahl an Tweets entnehmen mussten!

- Beantworten Sie abschließend die am Ende aufgeführten Fragen!

Beachten Sie dazu bitte folgende Hinweise/Kriterien:

- Berücksichtigen Sie nur die Inhalte derjenigen Tweets, die Ihrer Meinung nach auch mit dem angegebenen Thema zu tun haben!
- Inkludieren Sie Informationen, die über die Kernaussagen hinaus gehen (z.B. Wer? Wie? Wo? Wann?), falls es Ihnen sinnvoll und passend erscheint! Dies schließt auch die verwendeten Nutzerverlinkungen und Hashtags mit ein!
- Ergänzen Sie keine Informationen, die nicht in mindestens einem Tweet des gleichen Clusters zu finden sind! Beschränken Sie sich ausschließlich auf den Inhalt der angegebenen Textnachrichten!
- Ihre Zusammenfassung darf einen Umfang von 5 Sätzen nicht überschreiten – Sie müssen das gesetzte Limit allerdings auch nicht ausschöpfen!
- Welche Länge Ihre Sätze im Einzelnen haben, bleibt Ihnen überlassen. Achten Sie aber stets darauf, die Lesbarkeit bzw. Verständlichkeit Ihrer Zusammenfassung aufrechtzuerhalten!
- **Ansonsten gilt:** Es gibt kein *richtig* oder *falsch*! Verfahren Sie nach Ihrem Ermessen!

Cluster 1: „Corona Boris Johnson“ (45 Tweets)

Nr.	Tweet-ID	Text
1	1247240407781724163	Boris Johnson moved to intensive care after being admitted to hospital with persistent coronavirus symptoms
2	1247241256612347906	Prime Minister Boris Johnson has been moved to intensive care after his coronavirus symptoms worsened, Downing Street has confirmed.
3	1247241734058323968	Corona: Prime Minister Boris Johnson moved to intensive care after condition worsens
4	1247242052993200128	Downing Street says Prime Minister Boris Johnson has been moved to intensive care after his coronavirus symptoms worsened...
5	1247242253199912960	PM was moved to intensive care at 7pm. Before he went he asked Dominic Raab to deputise for him. As far as coronavirus is concerned, Raab is the effective acting Prime Minister
6	1247242481068126209	APNEWSALERT: LONDON (AP) — British Prime Minister Boris Johnson moved to intensive care unit of hospital after coronavirus symptoms worsen.
7	1247242494489878528	UK Prime Minister Boris Johnson has been moved to an intensive care unit after his COVID-19 condition worsened over the course of this afternoon, Downing Street says
8	1247242770546376706	Boris Johnson moved to intensive care after coronavirus symptoms worsen
9	1247242790481866752	#BREAKING #UK Prime Minister Boris #Johnson moved to intensive care after #coronavirus condition 'worsened'.
10	1247242851924221952	#Breaking Prime Minister Boris Johnson has been moved to intensive care after his coronavirus symptoms worsened, Downing Street has confirmed
11	1247243216455434244	BREAKING: British Prime Minister Boris Johnson moved to intensive care unit of hospital after coronavirus symptoms worsen.
12	1247243351893499909	British Prime Minister #BorisJohnson moved to intensive care unit after coronavirus symptoms worsen #Coronavirus
13	1247243439927971840	This is an incredibly serious situation for the British prime Minister. Boris Johnson's condition with #coronavirus at St Thomas's Hospital has dramatically worsened in the last few hours & he was moved into intensive care at 7pm. Fight hard Boris - we're all rooting for you.
14	1247243461914394632	#BREAKING: Boris Johnson moved to intensive care as coronavirus symptoms worsen
15	1247243592042717186	Coronavirus: Boris Johnson moved to intensive care
16	1247243747051651072	BREAKING: British PM Boris Johnson has been moved into intensive care at St. Thomas’ Hospital, London, as his Coronavirus symptoms worsen
17	1247243757348704256	BREAKING: UK Prime Minister Boris Johnson has been moved to intensive care as he continues to battle coronavirus #COVID19
18	1247243868870971397	U.K. is so badly organised, it cannot even prevent its own top political leader from becoming pandemic victim. @BorisJohnson Johnson moved to intensive care after #coronavirusuk symptoms worsen via The Irish Times
19	1247244105450651649	Boris Johnson moved to Intensive Care #COVID19 #BorisJohnson
20	1247244146303217665	Boris Johnson moved to intensive care after being admitted to hospital with coronavirus. Downing Street statement says UK Foreign Secretary Dominic Raab has been assigned to act as the PM's deputy in all necessary matters.
21	1247244229564391424	BBC News - Coronavirus: Boris Johnson moved to intensive care
22	1247244312993083392	#BREAKING :-Prime Minister #BorisJohnson has been moved to intensive care after his #CoronaVirusUpdate symptoms....
23	1247244414109413376	UK PM Boris Johnson moved to INTENSIVE CARE suffering from Covid-19
24	1247244505063002117	BREAKING: Prime Minister Boris Johnson has been moved into Intensive Care after he was admitted to hospital with coronavirus, according to reports. More to follow...
25	1247244971960344578	Coronavirus: Prime Minister Boris Johnson moved to intensive care after condition worsens

26	1247244996874559493	Boris Johnson has been moved to ICU due to Coronavirus and the Left cheers. The Conservative responds with “Really? That’s not nice. It could be you! I wouldn’t wish that on anyone”. Wake up normie, your efforts are in vain. Your enemies want you dead. Negotiations are over.
27	1247245020907913218	BREAKING - Boris Johnson moved to intensive care in hospital with coronavirus
28	1247245076855693313	British Prime Minister Boris Johnson, sick with covid-19, is moved to intensive care after his condition worsens
29	1247245088754982913	Prime Minister Boris Johnson moved to intensive care after being admitted to hospital with coronavirus - ITV News. Get well soon @BorisJohnson we’re all routing for you. #PrayForBoris
30	1247245601114411011	#BREAKING: British Prime Minister Boris Johnson has been moved to the intensive care unit of a London hospital after his #coronavirus symptoms worsened, per @AP. Johnson’s office says Johnson is conscious and does not require ventilation at the moment.
31	1247245701727150080	#BREAKING British Prime Minister Boris Johnson moved to intensive care after coronavirus symptoms worsen
32	1247245789824299012	Coronavirus: Boris Johnson moved to intensive care
33	1247245970200551425	Anybody tweeting hatefulness about @BorisJohnson being moved to intensive care should be named and shamed and ashamed for life. This is a human being with a partner and baby who has been leading this country well through unprecedented times #Covid_19 #BorisJohnson
34	1247246112823574530	Boris Johnson moved to intensive care after coronavirus symptoms worsen, Downing Street confirms #Topbuzz
35	1247246300069769217	Boris Johnson moved to intensive care... Thoughts: Me, it makes me sad. This is real people Covid19 don't give a damn how much money you have. Which God you worship or what your political sway is. Dead is dead
36	1247246301537775622	Boris Johnson moved to ICU as COVID-19 symptoms worsen
37	1247246842599026689	Boris Johnson moved to intensive care with COVID-19
38	1247246985217925120	Coronavirus: Boris Johnson moved to intensive care – BBC News
39	1247247320783224833	BBC News - Coronavirus: Boris Johnson moved to intensive care
40	1247249748802879489	Pray for the Prime Minister of the the U.K., Boris Johnson he's been moved to the ICU because of his worsening Covid-19 symptoms
41	1247250256770633729	#BREAKING UK Prime Minister Boris Johnson has been moved to intensive care via Spectator Index #Covid_19 #BorisJohnson #CoronavirusInUK
42	1247250601492262912	BBC News - Coronavirus: Boris Johnson moved to intensive care as symptoms 'worsen'
43	1247251447940395008	Boris Johnson moved to ICU with 'worsening' symptoms of #COVID19
44	1247256665256312832	Sad development, hope he pulls through. Stay Home people! Coronavirus: Boris Johnson moved to intensive care as symptoms 'worsen'
45	1247416083885285376	UK PM Boris Johnson moved to intensive care as coronavirus symptoms worsen; asks Foreign Secretary Dominic Raab to deputise 'where necessary'

Ihre Zusammenfassung für Cluster 1 („Corona Boris Johnson“):

Cluster 2: „Max von Sydow“ (80 Tweets)

Nr.	Tweet-ID	Text
1	1236983214994001920	Very very sad to hear of the passing of Max Von Sydow. He brought gravitas and grandeur no matter how silly the material. Flash Gordon would be a lesser film without him playing Ming. His moments on screen in films like Judge Dredd and The Force Awakens elevated them.
2	1236984802177421312	RIP Max von Sydow
3	1236985889269694464	The Exorcist and Star Wars actor Max von Sydow dies aged 90
4	1236986959962218497	Farewell then, Max von Sydow, owner of one of cinema's greatest faces
5	1236987704702898179	R.I.P. Max von Sydow, who has passed away at the age of 90.
6	1236988194475950080	#BREAKING Actor Max von Sydow dies at 90, wife tells French media
7	1236988198020173824	#Breaking Actor Max von Sydow has died at his home in France aged 90, his agent said
8	1236988309026545664	Actor Max von Sydow dies at 90, wife tells French media
9	1236988585380679680	R.I.P. Max von Sydow. Will always remember this iconic chess game.
10	1236988707502280704	R.I.P. Max von Sydow (1929-2020)
11	1236988896912777216	“It is with a broken heart and with infinite sadness that we have the extreme pain of announcing the departure of Max von Sydow,” his widow Catherine said. The actor dies on Sunday, March 8.
12	1236989196436475904	BREAKING Game Of Thrones star Max von Sydow dies aged 90
13	1236989619398496258	Max von Sydow, best known for his work with Ingmar Bergman (and once hailed as the “greatest actor alive” (has died. He was 90.
14	1236989746670260224	Max von Sydow, star of The Exorcist and The Seventh Seal, dies aged 90
15	1236989814798462977	You have to admire the majesty Max von Sydow brought to everything. Just an absolutely commanding screen presence. RIP.
16	1236990119774695426	Max von Sydow, known for his work in the films of Ingmar Bergman, has died at the age of 90.
17	1236990255502409728	Max von Sydow has passed away at the age of 90. In his long career he worked with such greats as Bergman, Friedkin, Pollack, Lynch, Wenders, Argento, Spielberg, and Scorsese. A true legend if there ever was one.
18	1236990284539613189	Max von Sydow, star of The Exorcist and The Seventh Seal, dies aged 90
19	1236990461719580673	Somewhere between this and the other world, Max von Sydow has finally finished that game of chess.
20	1236990705714769921	RIP Max Von Sydow - Chief Judge Fargo has taken the long walk @2000AD
21	1236990709561012224	Actor Max Von Sydow, who appeared in more than 100 films and TV series, has died at the age of 90
22	1236990904877166592	RIP Max Von Sydow (1929-2020) A legend and an icon of cinema #FilmTwitter
23	1236990919540441089	Farewell to Max von Sydow (1929-2020), an actor of considerable gravitas but with a gentle twinkle and the ability to move from intellectual drama to major genre cinema. His legacy is assured by two brilliant but very different films: THE EXORCIST (1973) and FLASH GORDON (1980).
24	1236991017787809793	Legendary actor Max Von Sydow has passed away at the age of 90. He recently starred in Star Wars: The Force Awakens and Game of Thrones, and is known for his iconic roles in The Exorcist and The Seventh Seal.
25	1236991102449782784	RIP Max von Sydow. And thanks for the amazing performances!
26	1236991166425554957	Game of Thrones and Star Wars actor Max von Sydow has died
27	1236991391508713472	Max Von Sydow, one of Sweden’s greatest actors has passed away. This silver screen legend will be missed by many! R.I.P.
28	1236991503492435969	One of my favourite actors has died. RIP. Max von Sydow, star of The Exorcist and The Seventh Seal, dies aged 90

29	1236991932926173185	May the force be with you, Max von Sydow. #RIP
30	1236992000244801538	Rest in peace Max von Sydow. You were a Swedish acting giant, and I'll never forget you.
31	1236992008427896834	MAX VON SYDOW who appeared in films such as The Exorcist , The Minority Report and The Seventh Seal has died aged 90 - RIP
32	1236992241593450496	BREAKING: Actor Max Von Sydow, known for his roles in The Seventh Seal, Flash Gordon, Star Wars, and Game of Thrones, has died at 90
33	1236992620292947969	‘Seventh Seal,’ ‘Star Wars’ Actor Max Von Sydow Dies at 90
34	1236992868201435136	The Exorcist's Max von Sydow has passed away at the age of 90. (10 April 1929 - 09 March 2020)
35	1236993303930900481	One of my favourite openings of a film ever. Max Von Sydow’s incredible voiceover.
36	1236993503886036993	RIP Max von Sydow.
37	1236993583611367424	JUST IN: ‘Game of Thrones’, Ingmar Bergman’s ‘The Seventh Seal’ star Max von Sydow dies aged 90
38	1236993660979380229	Max von Sydow, star of The Exorcist and The Seventh Seal, dies aged 90 RIP
39	1236994667603869698	Max Von Sydow dead: Star Wars, Game of Thrones and Exorcist actor dies aged 90
40	1236995234082353153	RIP Max von Sydow, who has died aged 90. He was best known for his roles in The Exorcist, The Seventh Seal, and Flash Gordon. More recently, he appeared in Game of Thrones and Star Wars: The Force Awakens
41	1236995446956011525	Max von Sydow, star of The Exorcist and The Seventh Seal, dies aged 90
42	1236995464517558272	Ah, Max von Sydow. Everyone will go on about The Seventh Seal but he'll always be the nice Nazi in Escape to Victory and...
43	1236995678439530497	I just watched you in Wild Strawberries (Ingmar Bergman, 1957) last week. Thank you for your stunning contributions in the world of cinema, Max von Sydow. You will be greatly missed! (10 April 1929 - 8 March 2020)
44	1236995690724761601	Max Von Sydow: A great actor and a lovely man. It was my great honour to meet him briefly on the set of TFA. Like Guinness & Cushing before him he added artistic weight, legitimacy and class to the #StarWars universe.
45	1236995699402629121	Max Von Sydow: Actor dies aged 90
46	1236995787227365377	Max von Sydow, the legend of Swedish cinema who dueled with Death in a game of chess in Ingmar Bergman's The Seventh Seal and portrayed the resolute Father Merrin in William Friedkin's The Exorcist (1973), has died. He was 90.
47	1236995816851673088	Rip Max Von Sydow
48	1236995854789206016	Max von Sydow. What a wonderful game of chess! So many exquisite moves. So many unforgettable moments. And that face, that voice that mirrored the awes and horrors of our existence. And by the way - you won the game. Immortal. #maxvonsydow
49	1236996150281945088	One of the greatest actors there has ever been. We've lost a true icon of cinema but he leaves behind an indelible career that spanned over six decades. Rest in peace, Max von Sydow.
50	1236996198172692480	"The more I had to act like a saint, the more I felt like being a sinner." - Max von Sydow, who has passed away at the age of 90.
51	1236996318427521026	Raise a glass for one of the true all-time greats. Rest in peace, Max von Sydow.
52	1236996433393442821	R.I.P. Max Von Sydow
53	1236996835161583617	Max von Sydow, The Exorcist, Seventh Seal, and Game of Thrones star, dies at 90
54	1236996926593093633	RIP Max Von Sydow, who was great in *everything*, from THE SEVENTH SEAL to THE FORCE AWAKENS. There’ll never be another career quite like his.
55	1236997749242912768	Max von Sydow, who played the role of Three-Eyed Raven in Game of Thrones is no more. He was 90 years old. Valar Morghulis
56	1236998006265860098	R.I.P. Max von Sydow
57	1236998136301867016	Ah man. RIP Max von Sydow. You were an always welcome presence in any film.

58	1236998650116616194	rip max von sydow aka lor san tekka thank you for being kylo's first on screen kill
59	1236998662238212099	BREAKING: Max von Sydow, an actor known to art house audiences through his work with Swedish director Ingmar Bergman and later to moviegoers everywhere when he played the priest in the horror classic "The Exorcist," has died at age 90.
60	1236998746325618689	Rest in peace Max Von Sydow looks like death finally won that game of chess....
61	1236998981290463233	Actor Max von Sydow dies aged 90
62	1236999092594769920	RIP, Max von Sydow. And now, my favorite line you ever uttered.
63	1236999218436411396	I guess I should rewatch The Exorcist again soon. #RIP Max Von Sydow.
64	1236999591721140227	max von sydow in "the virgin spring"
65	1236999679801528328	Max Von Sydow is gone. Wow.
66	1236999948236984320	'Exorcist' actor Max von Sydow dies at age 90
67	1236999984928653312	RIP to one of cinema's greatest artists; the legendary Actor Max von Sydow (1929-2020)
68	1237000391587487744	Breaking News: Max von Sydow, the Swedish actor who starred in "The Seventh Seal" and "Exorcist," is dead at 90
69	1237000392812134400	Max von Sydow, Swedish Star of 'The Seventh Seal' and 'Exorcist,' Dies at 90
70	1237000564791291904	It's not fair to wake up and realize Max von Sydow has left us. We should all feel so lucky that we got to live in the same lifetime as he did.
71	1237000664749834240	Actor Max von Sydow has died at age 90.
72	1237000751748190209	RIP Max von Sydow - Thank you for all your wonderful performances and the pleasure and thrills you've given throughout your great career. #maxvonsydow
73	1237001131227832321	Max von Sydow, star of The Seventh Seal and The Exorcist, dies aged 90
74	1237001673559748609	Max von Sydow (1929-2020)
75	1237001751779094528	RIP Max von Sydow. One of the greatest actors of World Cinema, his every performance touches your soul and mind. But his performance in Bergman's 'Shame' as Jan Rosenberg hit me real hard.
76	1237002171524268032	Legendary actor Max von Sydow, whose career spanned from The Seventh Seal, to The Exorcist, to Star Wars: The Force Awakens, has passed away at the age of 90:
77	1237002288658616321	Actor Max Von Sydow, who played the priest in "The Exorcist" and appeared in films and TV series including "Flash Gordon," "Game of Thrones" and "Star Wars: The Force Awakens," has died at the age of 90.
78	1237003158892752896	Max von Sydow, the Oscar-nominated actor best known for his roles in The Exorcist, The Seventh Seal, Flash Gordon, and Game of Thrones, has died: #maxvonsydow
79	1237003282431827968	R.I.P. Max von Sydow (1929 - 2020)
80	1237003479513792512	Actor Max Von Sydow, who appeared in more than 100 films and TV series, has died at the age of 90.

Ihre Zusammenfassung für Cluster 2 („Max von Sydow"):

Cluster 3: „First Day of Summer“ (70 Tweets)

Nr.	Tweet-ID	Text
1	1009623271292198912	The first day of summer will feel very much like an early spring day because, well, we live in Nebraska. #LNK #LNKforecast
2	1009732658350485504	It's the first day of Summer! Summer officially arrives at 6:07 a.m.
3	1009735302129311744	First day of summer and the longest day of the year good thing it’s payday
4	1009760660836831233	Happy First Day of Summer! 15 hours & 17 minutes of daylight today! Longest day of the year!
5	1009761931194982400	Happy first day of summer! I don’t think I’ve ever been so happy to begin a new season. Spring was a rough one. Bring it on sunshine and summertime
6	1009766266377293824	IN HAPPIER NEWS it’s the first day of summer!!!!!!!!
7	1009768978200322048	Happy Friday eve from the Hill my friends!! Guess what!! It's the first day of summer!! Woohoo! Have a great day!
8	1009769309541814273	oh right it's the summer solstice :'))) happy first day of summer, i guess :')))
9	1009770429421117440	It's a full-house in Canada's wetlands for the first day of summer! Ducklings, goslings, tadpoles, minnows and many others make wetlands THE place to see wildlife this season. #getoutdoors #nature #summer #firstdayofsummer #equinox #summerequinox #wetlands #Canada #canvasback
10	1009771665650343936	It’s official. The first day of summer is surfing on in.
11	1009771704518938625	The first day of Summer is finally here! #SummerIsHere #Summer2018 #FacilityMaintenance
12	1009772761454178304	First day of summer longest day of the yr
13	1009773245170675712	Before summer playdates, the American Academy of Pediatrics wants you to ask if there are unlocked guns in the house, and they've designated today -- the first day of summer
14	1009773608707805184	Happy first day of Summer. Remember to be the biggest hoe you can be.
15	1009773776500862976	Happy First Day of Summer. Spread the love and joy this season!
16	1009775928170438658	100% chance of rain till 5pm. Happy first day of summer
17	1009775991072415750	It's the first day of summer, which means it's also the longest day of the year! How are you spending today? #summersolstice #firstdayofsummer
18	1009776439137325059	It's dreary. It's rainy. There's more rain on the way. But, it IS the first day of summer, so here's a perfectly summery (shocked that that is actually a word) view of #Pittsburgh from the Duquesne Incline.
19	1009778612508110848	Happy first day of summer! If you have any changes with your insurance needs or have any questions, give us a call.
20	1009779237476339713	Happy first day of summer to everyone who isn’t living in south Texas or Arizona, because we’ve already been in summer since February #Texas #Arizona #SummerSolstice
21	1009779467978407936	It's June 21st, the First day of Summer and instead of being out playing where they belong, 2000 children will be spending their day in prison, unable to hug their siblings & having no clue if they will see their parents ever again. Welcome to America #SummerSolstice
22	1009779971475365889	Did you know that today is the longest day of the year? VCNP wishes you a happy first day of summer!
23	1009781062002724864	Retweeted Brian Krassenstein (@krassenstein): It's June 21st, the First day of Summer and instead of being out playing where they belong, 2000 children will be spending their day in prison
24	1009782792543178752	This is the first day of summer?
25	1009782999750041600	It's the official first day of summer! Sedlak's is your source for the best outdoor furniture this season:
26	1009783096236036097	Happy Summer Solstice! First Day of Summer! Enjoy!

27	1009783224216801280	It's the first day of #Summer and it's beautiful outside. #thisisedinburgh #sunnyedinburgh #beautifulcity
28	1009783280789606400	Today kicks off the first day of summer and the longest day of the year. Enjoy the extra sunlight today and make the most of it! #SummerSolstice
29	1009783451493588994	It's the first day of summer! Make sure you pull out your summer sun hats to do it right! #VintageFashion #FirstDayofSummer #SunHats
30	1009783670859878400	The #SummerSoltice marks the first day of Summer!
31	1009783935101034497	Happy first day of summer! Who's ready to paint some beachy colours in their home or cottage?
32	1009784392275775489	Happy first day of summer! If you have any changes with your insurance needs or have any questions, give us a call.
33	1009784711038865408	Hey .... Happy first day of summer! Happy longest day of the year! Happy Thursday! Smile, there's lots to be happy about
34	1009784937128456192	What do you have planned now that it is officially the first day of #summer?
35	1009785665318543361	Good morning Toronto. The sun is out. The humidity is low. It's going to be a good day in the city. Enjoy the first day of summer.
36	1009787835791151104	It's the first day of Summer and my Yankees won last night (again). No one can ruin my vibe with their bitter vibes, move along!
37	1009788058114383872	Happy First Day of Summer.
38	1009788498512154624	Happy first day of summer!
39	1009788527368921089	Happy first day of summer, Terriers! The warmer weather brings plenty of opportunities for fun in Boston that will make you fall more in love with this city. Try them all!
40	1009789312177602560	First day of summer forecast
41	1009789874214449152	It's officially the first day of summer!! Take 30% off everything in the shop when you use code SUMMERLOVIN at checkout!
42	1009790385919537154	Happy First Day of Summer! Spend time in the sun today #SummerSolstice #Summer
43	1009790651788070913	Happy first day of summer! @GeorgeHamilton is ready to catch some rays #FirstDayOfSummer
44	1009791015094345735	its the first day of summer and in AZ its already going to be 92 degrees at 9 am wtf
45	1009791472256913409	Welcome to the first day of summer, otherwise known as the 85th day of summer in the Sunshine State.
46	1009791983957749760	Happy first day of summer everybody
47	1009793628141649920	It's the first day of summer! To help you keep things as stress-free as possible, I'm sharing my best tips for keeping a clean house this summer (and beyond!). #SummerSolstice #cleaning #schedule #cleanhouse
48	1009794148239642624	It's the first day of summer.....
49	1009794339009105922	Today is the first day of summer. Stay safe when working outside with #WaterRestShade
50	1009794676688343040	@Lilymorseee Good thing it's only the first day of summer
51	1009796442519687168	It's the first day of summer! My favorite season! #summerchild (yes, still a child at heart!)
52	1009797860190277632	First day of summer and longest day of the year
53	1009797899922956288	First day of summer means popsicles...end of story #SummerSolstice
54	1009798100813320192	Today marks the first day of SUMMER Grab your Hawaiian shirt & join us for a beach party today on TPIR
55	1009798137005989889	It's the first day of #summer! It's important to always to wear sunscreen and know what to look for when it comes to skin cancer self-screening. Check out our tips!
56	1009798560639062016	Happy first day of summer! Who is ready to hit the beach? #BoucherandCo #FirstDayofSummer #Summer #Beach #NYC #NewYork #NewYorkCity #Nature #Relax

57	1009798619359268865	It’s the first day of summer and the first day of my DPT program @TempleUniv #NotACoincidence
58	1009798753576988673	First day of summer, last day of school. They grow up so fast.
59	1009799470777856000	First Day of Summer
60	1009799600801243138	Happy First Day of Summer
61	1009799697308012544	Happy First Day of Summer! Before leaving on vacation, follow these housekeeping tips to minimize any damage from severe weather.
62	1009799865059237888	Happy First Day of Summer! Before leaving on vacation, follow these housekeeping tips to minimize any damage from severe weather.
63	1009799881853194240	Happy First Day of Summer! Before leaving on vacation, follow these housekeeping tips to minimize any damage from severe weather.
64	1009799965705719813	Happy First Day of Summer! Before leaving on vacation, follow these housekeeping tips to minimize any damage from severe weather.
65	1009799978292740097	Happy First Day of Summer! Before leaving on vacation, follow these housekeeping tips to minimize any damage from severe weather.
66	1009800196413411329	Happy First Day of Summer! Before leaving on vacation, follow these housekeeping tips to minimize any damage from severe weather.
67	1009800980739842049	Did you know that back-to-school shopping begins the first day of summer vacation? Yes, I'm picking up thrifted pieces here and there to enhance My Teen's wardrobe. What have you scored at the thrift store lately?
68	1009801635063848966	Happy first day of Summer, everyone! Stop by CCD to check out our selection of Summer books! #FLPkids #SummerSolstice #kidlit
69	1009801953805729792	First day of summer and I’m freezing at work
70	1009802859796402176	First day of summer is a rainy one

Ihre Zusammenfassung für Cluster 3 („First Day of Summer“):

Cluster 4: „Warren drops out“ (75 Tweets)

Nr.	Tweet-ID	Text
1	1235230218295685122	Bernie would have easily won Massachusetts, Minnesota, and Texas if neoliberal tool Elizabeth Warren had dropped out. But that's exactly why she stayed in the race: to split the "progressive" vote. That's why oligarchs dumped $14 million into her super PAC.
2	1235344882174234624	All seriousness aside for a moment, if Elizabeth Warren drops this in a fundraising email she’s getting my life savings.
3	1235586284413845504	Wait, it's Wednesday and Elizabeth Warren still hasn't dropped out? Nevertheless, she persisted.
4	1235590936085680134	Breaking News: Elizabeth Warren is dropping out of the 2020 U.S. presidential race. Once a front-runner, she was unable to build broad support and placed 3rd in her home state.
5	1235591196463869952	Elizabeth Warren is dropping out of the race per two sources.
6	1235591254299127808	BREAKING: Elizabeth Warren is dropping out of the presidential race, according to The New York Times. It's now down to Joe Biden vs. Bernie Sanders.
7	1235591365347721216	BREAKING: Elizabeth Warren is dropping out of the race for President. I would like to congratulate and thank this amazing woman for her fight. Let's hope that she gets a spot in the future administration of Joe Biden or Bernie Sanders. We love you Liz!
8	1235591376122847232	BREAKING: Elizabeth Warren is dropping out of the 2020 U.S. presidential race
9	1235591778226536449	JUST IN: Elizabeth Warren is dropping out of the presidential race, a source familiar with her plans tells CNN, following another round of disappointing finishes in primary contests across the country on Super Tuesday.
10	1235591900272431105	BREAKING: NYT: Elizabeth Warren is dropping out of the presidential race.
11	1235591937169788928	Elizabeth Warren is dropping out.
12	1235591964784906240	#BREAKING on #OANN: Massachusetts Senator Elizabeth Warrens drops out of the 2020 Presidential race #Elections2020
13	1235592115859709953	Elizabeth Warren has dropped out of the 2020 presidential race.
14	1235592121450508289	Breaking: Elizabeth Warren dropping out. Took her an extra day to conclude the obvious, she had no path to the nomination. Now officially a two-person race
15	1235592252241653760	Elizabeth Warren is dropping out of the presidential race. Now it’s really a two-man contest between Biden and Bernie.
16	1235592394348822530	Massachusetts Senator Elizabeth Warren drops out of race for Democratic presidential nomination #CJAD800
17	1235592468529324034	BREAKING: Elizabeth Warren drops out of the presidential race after a Super Tuesday defeat.
18	1235592519691493377	Elizabeth Warren is dropping out
19	1235592524091125761	BREAKING: Elizabeth Warren drops out of the presidential race.
20	1235592994096443392	Elizabeth Warren Drops Out of Presidential Race
21	1235593018255818752	JUST IN: Sen. Elizabeth Warren is dropping out of the 2020 presidential race after a disappointing Super Tuesday showing.
22	1235593076573458432	#BREAKING: Sen. Elizabeth Warren drops out of the 2020 presidential race. #OANN
23	1235593111729926144	Breaking: Pocahontas! See ya!! Elizabeth Warren drops out of presidential race
24	1235593208043679745	Elizabeth Warren has dropped out of the presidential race.
25	1235593287731310592	ELIZABETH WARREN DROPS OUT IF RACE TODAY. Elizabeth Warren, Once a Front-Runner, Will Drop Out of Presidential Race
26	1235593324318224384	Sen. Elizabeth Warren drops out of the race for president, according to reports, after failing to win any primary states
27	1235593337320636416	#BREAKING Elizabeth Warren dropping out of Democratic presidential race: US media

28	1235593354827661313	Elizabeth Warren drops out of 2020 presidential race
29	1235593430773923846	Elizabeth Warren drops out of 2020 presidential race
30	1235593559358590977	BREAKING: Elizabeth Warren is dropping out of the presidential race, ending a campaign that at one point seemed poised to win the Democratic nomination
31	1235593581324361729	Prayers up to the Native American community for Elizabeth Warren dropping out of the election. That was their candidate
32	1235593621484732417	JUST IN: CNN reports that Elizabeth Warren is dropping out of the presidential race.
33	1235593673854812165	BREAKING: Sen. Elizabeth Warren is dropping out of the 2020 presidential race after a disappointing Super Tuesday showing - NBC
34	1235593746772787200	I'm sad about Elizabeth Warren dropping out. She was one of my favorite candidates in the race.
35	1235593812023685120	#BREAKING Elizabeth Warren drops out of 2020 presidential race
36	1235593845561147392	Elizabeth Warren Dropping Out Of Presidential Race
37	1235593864569896963	**BREAKING** Elizabeth Warren drops out... her campaign trail of tears has finally come to an end.
38	1235593884404723712	BREAKING: Elizabeth Warren is dropping out of the Democratic primary, according to a person with her campaign, after losing across the map on Super Tuesday, including in her home state of Massachusetts
39	1235593921054617602	Warren is Out! #ElizabethWarren Elizabeth Warren Dropping Out of Presidential Race
40	1235594084653453313	Sen. Elizabeth Warren drops out of presidential race, according to reports
41	1235594231223390208	Elizabeth Warren has dropped out. She was the last Mohican in the race. Now there are Nohicans.
42	1235594379357638660	Elizabeth Warren says she's dropping out of the race for president, solidifying the Democratic contest as a two-way race between Joe Biden and Bernie Sanders
43	1235594397431074817	BREAKING: U.S. Senator Elizabeth Warren is dropping out of the race for president - NBC
44	1235594629333929985	Elizabeth Warren Drops out of Presidential Race ... bye bye Pocahontas @realDonaldTrump
45	1235594680223625222	Breaking: Elizabeth Warren will be dropping out of the presidential race today
46	1235594860587036672	Elizabeth Warren To Drop Out of Presidential Race
47	1235595007387734016	@realDonaldTrump @PamelaGeller Trump I CAN'T TAKE ALL THIS WINNING. BREAKING: Sen. Elizabeth Warren is dropping out of the 2020 presidential race after a disappointing Super Tuesday showing - NBC
48	1235595134978478082	Massachusetts Sen. Elizabeth Warren is dropping out of the 2020 presidential race.
49	1235595217102729216	Wise choice.Sen. Elizabeth Warren Drops Out Of 2020 Presidential Race
50	1235595225499922433	Elizabeth Warren is dropping out of the 2020 race - CNNPolitics
51	1235595256315469825	Warren Out: Elizabeth Warren is dropping out today. If @ewarren endorses Bernie Sanders, I know for a fact, a lot of her supporters are not going to follow her. I know I won't. I want no part of the cantankerous con-man from Vermont.
52	1235595306873401344	Sen. Elizabeth Warren Drops Out Of The 2020 Presidential Race
53	1235595313575915521	Elizabeth Warren to drop out of Democrat race
54	1235595435214962688	Elizabeth Warren drops out of presidential race #KMOV
55	1235595481344073731	Elizabeth Warren has Dropped out of the 2020 Democratic Presidential Race.
56	1235595917564071939	Sen. Elizabeth Warren Drops Out Of 2020 Presidential Race
57	1235595956021874688	BREAKING: Massachusetts Sen. Elizabeth Warren is dropping out of the 2020 presidential race.
58	1235596232128487424	absolutely the best time to find out that elizabeth warren dropped out is while i was on hold with my insurance company to find out why they denied coverage for my therapist appointments

59	1235596274042318848	Elizabeth Warren drops out of the presidential race, following another round of disappointing finishes in primary contests across the country on #SuperTuesday. The Massachusetts senator centered her bid on a promise to wipe out corruption in #Washington
60	1235596316526252033	BREAKING: Elizabeth Warren Drops Out Of Race.
61	1235596393735053316	Fun fact: If Elizabeth Warren had dropped out before #SuperTuesday, Bernie would most likely have the most delegates right now. It sure feels like a DNC set up. #RiggedPrimary
62	1235596965876031488	JUST IN: Elizabeth Warren to drop out of the 2020 presidential race
63	1235597251327791106	Elizabeth Warren is dropping out of the 2020 presidential race.
64	1235597469448155136	Extremely sad. Consolidation is we'll still have her a major force in Senate. Elizabeth Warren is dropping out of the 2020 presidential race. via @HuffPostPol
65	1235597604370739200	BREAKING: Elizabeth Warren drops out of presidential race after disappointing Super Tuesday finish
66	1235598018214129664	Breaking: Elizabeth Warren has dropped out of the Democratic presidential race, just days after the onetime front-runner couldn't win a single Super Tuesday state - not even her own.
67	1235598799034343427	Elizabeth Warren to drop out of Democrat race to be presidential candidate
68	1235599411410907137	Elizabeth Warren has dropped out LOL LOL LOL LOL LOL LOL!
69	1235600180704993281	now that elizabeth warren dropped out can we please vote for bernie sanders thanks
70	1235600453762715648	Democrats Disenfranchise Their Own Voters by Dropping Out !!!!And Elizabeth Warren just bit the dust !!!!!!
71	1235600569017880578	Elizabeth Warren Drops Out of White House Race via @BreitbartNews
72	1235600946534604800	I'm really sad about Elizabeth Warren dropping, and she will forever be my favorite – but now I'd love for her to endorse Sanders, campaign her heart out for him, and deliver us from Joe Biden (and Trump). She singlehandedly destroyed Bloomberg, so I have high hopes!
73	1235601065959014401	Scenes from the Democratic Primaries as Elizabeth Warren drops out
74	1235602599077752833	After Bloomberg, once a frontrunner Elizabeth Warren drops out of Democratic Presidential race
75	1235602833946238976	Elizabeth Warren dropping out after decimating Bloomberg's campaign, which cleared the way for a Biden consolidation, is maddening yet wholly unsurprising to any woman who has ever worked in a male-dominated field.

Ihre Zusammenfassung für Cluster 4 („Warren drops out“):

Cluster 5: „Separating children“ (65 Tweets)

Nr.	Tweet-ID	Text
1	1008828027088900097	In my latest for @Playboy, I note that children separated from their parents at the border may end up de facto trafficked via mostly religious adoption agencies. White supremacist theocracy is here. #FamiliesBelongTogether #Exvangelical
2	1008878718943617024	Up to 200 children separated from their parents at the U.S.-Mexico border are being held in this tent city outside Tornillo, Texas
3	1009116764230356992	We all should be able to agree that in the United States of America, we will not intentionally separate children from their parents. We will not do that. We are better than that! We are so much better!
4	1009138091066523648	Today I'm recalling four Virginia National Guard soldiers and one helicopter from Arizona. Virginia will not devote any resource to border enforcement actions that support the inhumane policy of separating children from their parents.
5	1009177548343963648	To ALL border agents: This is a public call asking all Border Agents to Refuse any and all orders to separate children from their parents. You will go down in history as a hero, and as a patriot. You will not regret it! Please share if you support this call to action!
6	1009397278124249090	Brexit has made us so desperate for a trade deal with Trump, that even the "Leader of the Opposition" can't risk asking the government to condemn the inhuman treatment of children separated from their parents and caged. #PMQs
7	1009425028113461249	Kirstjen Nielsen should resign immediately. She is leading an agency that is separating children from their parents for no reason. It’s immoral, it’s unnecessary, and it’s got to stop. She should go
8	1009443871967850497	Newflash: you can be conservative on immigration policy and still be humane and compassionate enough to reject the idea of separating children from their parents. Come on, people.
9	1009447406956498947	I am appalled by the state of the children who are being separated from their parents, after getting caught at the US-Mexico border. What I don’t understand is, what other options does the US administration have? What did Obama govt do for similar cases #WorldRefugeeDay
10	1009456082505248769	(Thread) There are a lot of people writing about how separating children from parents is “not the law.” In fact, what’s happening is unconstitutional under two leading Supreme Court cases. Here you go everyone: Twitter Law 101
11	1009474822290771969	hi it's me, guy who had a full on nervous breakdown because of a politically correct thing in a movie last month. heres why the uproar about children being separated from their parents and put in cages is irrational female hysterics caused by an absence of testosterone and logic
12	1009475925962977280	NYT reports that, rather than separating children from parents, the Trump administration will hold children indefinitely with their parents, violating a court settlement and inevitably triggering a legal challenge
13	1009483722876432384	We will make sure that voters in November remember which members of Congress allowed the President to separate children from their parents.
14	1009486573581275136	It personally hurts me that American children are separated from their parents because of bogus child cases, lost in the foster care system, stolen & sold into sex trafficking AND there is not much outrage about that? I guess you have to be an illegal to matter
15	1009498042410655744	Dear Democrats, there are currently 400,000 children in foster care in the United States. Children who were “separated” from their parents. TAKE CARE OF OUR CHILDREN and stop pushing your false narrative just because you hate Trump.
16	1009503862275878912	I'd like to thank everyone making sure that politicians who endorsed separating children from their parents get separated from their jobs.
17	1009507453631901696	Hey Spanky @realDonaldTrump don't cha just hate when you get caught lying? Like the time you tweeted, Democrats are responsible for separating children from their parents, when in fact, everyone knew you were Lying, since it's your Policy! Sad Liar!
18	1009522546478342144	@peopleenespanol Editor in Chief @ArmandoCorrea: "Those caged children who are separated from their parents and cry out in desperation, could be our children."

19	1009536707664936966	Glad to see that the Administration is listening to Americans on the moral atrocity of separating children from parents at the border. This is a good first step. Now let's fix this system. We can have security while still showing compassion to those fleeing violence.
20	1009542866962509826	Just to be clear, separating children from parents is inhumane, and keeping children with parents is inhumane. Which means the only humane solution is release. Which is the entire agenda.
21	1009565034446385152	"There will not be a grandfathering of existing cases." Trump admin has no plans to promptly reunite the 2,300 children who've already been separated from their parents
22	1009569771732553729	The Democrat pivot from "Trump is a monster who wont help children separated from parents at the border" to "Trump is monster who imprisons WHOLE FAMILIES!" is so shameless, its hilarious and depressing at the same time
23	1009570811991207936	Trump's executive order stops the horrors of separating children from their parents. As for the 1000s already separated? Nothing. They remain alone, interned, crying, with no one to console them. An Cult45 doesnt give a damn.
24	1009574132248449024	There are 239 immigrant children separated from their parents who are now in the custody of a social service organization in East Harlem that's placing them in foster homes, according to Mayor de Blasio. The youngest is just 9 months old.
25	1009578928544124929	BREAKING: HHS says that children who have already been separated from their parents will NOT be reunited while their parents face hearings. #ExecutiveOrder
26	1009579930525499394	NEW: Per @CNN, the Trump administration has confirmed that Trump's #ExecutiveOrder does *nothing* to address the thousands of children who have already been separated from their parents under Trump's "zero-tolerance" policy.
27	1009581345025163264	What happened at the Border should alarm everyone. That things devolved that quickly should scare anyone with even a scintilla of common decency. If you can separate a child from their parents, dehumanize them to that degree, then you've already set the ground work for genocide.
28	1009589194736455680	Children already separated from parents will NOT be reunited with their parents. and placed in some sort of foster care. So, we kidnap the kids, deport or indefinitely detain the parents, then hand the kid over to the system? Some are so you young they do not know their parents
29	1009605359600668672	First, Trump created a crisis by repealing DACA, and threatened to deport DREAMErs if he didn't get his wall. Next, he created a crisis by separating children from their parents, and held them hostage. This nightmare isn't over. And it won't be over until his presidency is.
30	1009606854610219008	Trump retreats because his family separation policy was evil, and because of you. Your outrage & activism beat @realDonaldTrump. But keeping 2,300 babies & children separated from their parents continues to be evil. We need your activism to increase. Also, vote this November.
31	1009610961769127936	Ok, listen carefully. Putting children in jail with their parents is not a solution to separating parents from their children.
32	1009622275027623936	When Trump won the elections Melania decided not to move their 10 yo son to DC in the middle of the school year because "At that age, it's hard to explain to them", "at that age, every child would worry" But separating children from their parents while fleeing persecution is fine
33	1009629975169982464	"President Trump's executive order is little solace to the children and parents who have already been separated from one another," @RepRosaDelauro says
34	1009633888950472704	He said today that the problem of separating children from parents at the border has been going on for years. NO, IT HAS NOT. The policy began with his administration. The law allowed him to do it but the law did not call for it. Sessions & Miller admitted they devised the plan.
35	1009637094136958976	This ugly chapter in American history is not closed. There are 2,000+ children separated from their parents. @DHSgov must reunite them NOW. #ReuniteFamilies
36	1009643916688265216	@CNNPolitics So, the legislation would eliminate separating children from their parents - which is what @RepLujanGrisham wanted - but, but you're not gonna support it? It's almost like you're just a liar and have no compassion for "the children". Like you were using them for political gain.

37	1009655579562991616	Side 1: Separates children from their parents, throws them in cages, and forcibly injects some of them with psychiatric drugs. Side 2 : Respond with vituperative words.***** NYT: Both sides, guys. Both sides are behaving badly.
38	1009656797408518145	Both Obama and Clinton separated children from parents at the border so SHUT THE ENTIRE FUCK UP WITH YOUR FAKE MORAL OUTRAGE ABOUT TRUMP.
39	1009681797326745600	How is it evil to support separating children from their parents? How is it evil to support putting children in cages? How is it evil to defend the indefensible? You tell me?
40	1009754732695465990	@realDonaldTrump I challenge Trump supporters to prove Obama was separating children from their parents at the border. Obama did not prosecute people for seeking asylum. The pictures of children being held under Obama are of minors who arrived unaccompanied, not with parents.
41	1009757064359538693	Guess who is not turned away at the border? Legal Immigrants. Guess which children are not separated from their parents? Children of parents that have followed our countries rule of law. #EnforceUSLaws #KeepAmericaSafe #MAGA
42	1009759695698096129	Today at the WH, Pres holds a Cabinet Meeting, his 14th since taking office. Includes a photo op, where he's certain to be asked how the Government will reunite immigrant children already separated from their parents detained elsewhere. His Exec Order makes no provision for that.
43	1009762778125651968	American, Frontier, Southwest and United airlines are refusing to fly immigrant children separated from their parents for the federal government.
44	1009763659588030464	HAPPENING NOW-Foster parents are walking immigrant children separated from their parents at the border into the Cayuga Center in East Harlem. Some of them so young they're being carried in or hands held. #nbc4ny
45	1009764012417044481	Dems forced to show their hand: After the President's EO to eliminate children being separated from their parents, the Democrats shifted their tactic from child abuse to the elimination of all Border violation enforcement. That's been their goal all along. Zero law enforcement.
46	1009766838417416192	Why didn't CNN care when the Obama administration separated children from their parents when they had crossed the border illegally?
47	1009767468250742784	@StatesWatchman Did separating children from their parents fix, alleviate or in any way assuage any of those very real problems? If so, how?
48	1009768152849362945	Most Texas voters oppose separating children and parents apprehended while trying to enter the country illegally. But our polling found that male Republicans support separating families.
49	1009773341991948288	These images were provided to us by @HHSGov - the agency claims this is a facility in Bristow, VA. which is housing children who've been separated from their parents after allegedly crossing the southern border illegally; boys & girls as young as infants are at this facility
50	1009775592735215619	Democrats have been defending and promoting abortion for over fifty years. Now they expect everyone to believe they care about separating children from parents.
51	1009778059711565824	Trump signs executive order to undo his own policy that separated children from their parents @politicususa
52	1009780214757494785	But the point all along was to break up families in order to cause fear in (read: terrorize) parents that they might be separated from their children and therefore shouldn't think about immigrating to the us
53	1009780495780114434	What about all the children separated from there parents due to drug overdoses..... #KeepFamiliesTogether
54	1009781461287931904	CNN's Erica Hill: Are you confident that the children separated from their parents will be reunited? Rep. Elijah Cummings: "No, I am not confident ... The Trump administration has told us all kinds of untruths, so why should we believe them now?"
55	1009782492239400960	"there are more than 2,000 children already separated from their parents; the executive order does nothing to address that nightmare." #WelcometoAmerica
56	1009783030733471744	#Congress, you can stop this madness. End @realDonaldTrump's horrific policy of separating children from their parents. Now. CALL & don't stop: (844)899-8261 #KeepFamiliesTogether ADD YOUR NAME & join this event

57	1009784282863239168	Families Belong Together Rally - 6/30: DPOP supports Families Belong Together. We cannot remain silent as families seeking a new life in the U.S. are torn apart and children are separated from their parents.
58	1009785285721427968	@realDonaldTrump Yes, Republicans, now we get it. Having a President who separates children from their parents & throws them into concentration camps makes us EVERY BIT AS ANGRY as it made you when we had a President who raised taxes to pay for national health care
59	1009786850838532096	I'm just finding all this Trump hatred funny. Y'all live in a country with a decades old industrial prison complex that separates children from parents on a DAILY basis, yet THIS is what triggered you. Welcome to America fam.
60	1009786892068577280	@realDonaldTrump #TrumpPolicy created the #BorderCrisis. #TrumpPolicy interned children. #TrumpPolicy separated children from their parents and sent them across state lines, some as young as 9mos old. #TrumpPolicy enables the corruption of the #TrumpAdministration. #TrumpPolicy needs to be stopped.
61	1009788510872666112	We need a countdown clock beginning the minute Trump signed the order backing down from his own policy of separating children from their parents at the border. The clock will stop the minute the last child is reunited with their parents. I predict the clock will run for months.
62	1009789047932379136	NEA prez @Lily_NEA calls Trump plan to merge federal agencies an attempt to distract Americans from humanitarian crisis he created by separating children from their parents. #FamiliesBelongTogther
63	1009792127985946624	Thank you for your kind words. Much respect to you too. However, if you dig down to what this zero tolerance policy has spawned you will see this is not liberal rhetoric. Children separated from their parents ripples through their lives forever.
64	1009793372263858177	@Auxarcman @AngelWhyspr @bartramscenic @Piatfernandez @SenSchumer @realDonaldTrump You haven't shared a single piece of evidence dude! Do you actually hear yourself? You think separating children from their parents ensures their safety when we have provided you plenty of scientific evidence otherwise
65	1009802092247044096	@Sheilaehuffman @realDonaldTrump Going forward.. does nothing for the 2300 children already separated from their parents.

Ihre Zusammenfassung für Cluster 5 („Separating Children“):

Abschließende Fragen – Teil 1:

Bitte beantworten/vervollständigen Sie die nachfolgenden Fragen/Aussagen (z.B. durch Markieren/Durchstreichen)! Hier ist ausschließlich nach Ihrem persönlichen Eindruck gefragt – es gibt weder *richtig* noch *falsch*!

1. Das Zusammenfassen von Cluster 1 („Corona Boris Johnson“) fiel mir...

Sehr leicht (1) (2) (3) (4) (5) Sehr schwer

2. Das Zusammenfassen von Cluster 2 („Max von Sydow“) fiel mir...

Sehr leicht (1) (2) (3) (4) (5) Sehr schwer

3. Das Zusammenfassen von Cluster 3 („First Day of Summer“) fiel mir...

Sehr leicht (1) (2) (3) (4) (5) Sehr schwer

4. Das Zusammenfassen von Cluster 4 („Warren drops out“) fiel mir...

Sehr leicht (1) (2) (3) (4) (5) Sehr schwer

5. Das Zusammenfassen von Cluster 5 („Separating Children“) fiel mir...

Sehr leicht (1) (2) (3) (4) (5) Sehr schwer

6. Fanden Sie es unangenehm/lästig pro Cluster mindestens 45 Tweets zu lesen?

Überhaupt nicht (1) (2) (3) (4) (5) Auf jeden Fall

7. Würden Sie auch von sich aus eine Vielzahl (mindestens 45) von Suchergebnissen (z.B. Tweets) lesen, wenn Sie nicht darum gebeten werden?

Überhaupt nicht (1) (2) (3) (4) (5) Auf jeden Fall

8. Fanden Sie es unangenehm/lästig, die angegebenen Cluster zusammenzufassen?

Überhaupt nicht (1) (2) (3) (4) (5) Auf jeden Fall

9. Die Zusammenfassung welches Clusters war für Sie am ehesten/meisten unangenehm?

Cluster (1) (2) (3) (4) (5)

10. Was genau fanden Sie an der Zusammenfassung des unter 9. genannten Clusters unangenehmer als bei den anderen Clustern (Mehrfachnennungen erlaubt)?

- O Die aufwühlende/uninteressante Thematik
- O Das Identifizieren der wichtigsten Inhalte
- O ☐
- O ☐
- O ☐

Abschließende Fragen – Teil 2:

Nachfolgend finden Sie für jedes Cluster eine automatisiert erstellte Aggregation. Bitte beantworten Sie auch hierzu die angegebenen Fragen! Nutzen Sie dafür zunächst nur die jeweils unter **a)** aufgeführten Antwortmöglichkeiten! Zeile **b)** benötigen Sie später.

- **Agg_C1**: Aggregation von Cluster 1 („Corona Boris Johnson“):

  #breaking british prime minister boris johnson move to intensive care after he coronavirus symptom worsen downing street have confirm. British prime minister #borisjohnson move to intensive care unit of hospital after coronavirus symptom worsen. Bbc news coronavirus boris johnson move to intensive care after be admit to hospital with coronavirus itv news get well soon @borisjohnson we be all route for you #prayforboris.

11. Wie bewerten Sie die Verständlichkeit der angegebenen Agg_C1?

a)	Sehr schlecht	(1)	(2)	(3)	(4)	(5)	Sehr gut
b)	Sehr schlecht	(1)	(2)	(3)	(4)	(5)	Sehr gut

12. Enthält die angegebene Agg_C1 die wichtigsten Informationen des Clusters?

a)	Überhaupt nicht	(1)	(2)	(3)	(4)	(5)	Auf jeden Fall
b)	Überhaupt nicht	(1)	(2)	(3)	(4)	(5)	Auf jeden Fall

13. Fehlen in der angegebenen Agg_C1 noch Inhalte, die zwingend vorhanden sein sollten?

a)	Überhaupt nicht	(1)	(2)	(3)	(4)	(5)	Auf jeden Fall
b)	Überhaupt nicht	(1)	(2)	(3)	(4)	(5)	Auf jeden Fall

14. Wie bewerten Sie die angegebene Agg_C1 im Allgemeinen?

a)	Sehr schlecht	(1)	(2)	(3)	(4)	(5)	Sehr gut
b)	Sehr schlecht	(1)	(2)	(3)	(4)	(5)	Sehr gut

15. Wie bewerten Sie die angegebene Agg_C1 hinsichtlich Ihrer eigenen Zusammenfassung? Im Vergleich zu meiner von Hand verfassten Aggregation ist Agg_C1…

a)	Viel schlechter	(1)	(2)	(3)	(4)	(5)	Viel besser
b)	Viel schlechter	(1)	(2)	(3)	(4)	(5)	Viel besser

- **Agg_C2:** Aggregation von Cluster 2 („Max von Sydow“):

  Rip max von sydow star of the exorcist and the seventh seal die age 90. The exorcist and star war actor max von sydow who appear in more than 100 film and tv series have die at the age of 90. Rest in peace max von sydow die at 90 wife tell french media.

16. Wie bewerten Sie die Verständlichkeit der angegebenen Agg_C2?

a)	Sehr schlecht	(1)	(2)	(3)	(4)	(5)	Sehr gut
b)	Sehr schlecht	(1)	(2)	(3)	(4)	(5)	Sehr gut

17. Enthält die angegebene Agg_C2 die wichtigsten Informationen des Clusters?

a)	Überhaupt nicht	(1)	(2)	(3)	(4)	(5)	Auf jeden Fall
b)	Überhaupt nicht	(1)	(2)	(3)	(4)	(5)	Auf jeden Fall

18. Fehlen in der angegebenen Agg_C2 noch Inhalte, die zwingend vorhanden sein sollten?

a)	Überhaupt nicht	(1)	(2)	(3)	(4)	(5)	Auf jeden Fall
b)	Überhaupt nicht	(1)	(2)	(3)	(4)	(5)	Auf jeden Fall

19. Wie bewerten Sie die angegebene Agg_C2 im Allgemeinen?

a)	Sehr schlecht	(1)	(2)	(3)	(4)	(5)	Sehr gut
b)	Sehr schlecht	(1)	(2)	(3)	(4)	(5)	Sehr gut

20. Wie bewerten Sie die angegebene Agg_C2 hinsichtlich Ihrer eigenen Zusammenfassung? Im Vergleich zu meiner von Hand verfassten Aggregation ist Agg_C2…

a)	Viel schlechter	(1)	(2)	(3)	(4)	(5)	Viel besser
b)	Viel schlechter	(1)	(2)	(3)	(4)	(5)	Viel besser

- **Agg_C3:** Aggregation von Cluster 3 („First Day of Summer“):

  Happy first day of summer before leave on vacation follow these housekeeping tip to minimize any damage from severe weather. It be june 21st the first day of summer and instead of be out play where they belong 2000 child will be spend they day in prison unable to hug they sibling & have no clue if they will see they parent ever again welcome to america #summersolstice. Before summer playdate the american academy of pediatrics want you to ask if there be unlock gun in the house and they have designate today the first day of summer if you have any change with you insurance need or have any question give we a call.

21. Wie bewerten Sie die Verständlichkeit der angegebenen Agg_C3?

a)	Sehr schlecht	(1)	(2)	(3)	(4)	(5)	Sehr gut
b)	Sehr schlecht	(1)	(2)	(3)	(4)	(5)	Sehr gut

22. Enthält die angegebene Agg_C3 die wichtigsten Informationen des Clusters?

a)	Überhaupt nicht	(1)	(2)	(3)	(4)	(5)	Auf jeden Fall
b)	Überhaupt nicht	(1)	(2)	(3)	(4)	(5)	Auf jeden Fall

23. Fehlen in der angegebenen Agg_C3 noch Inhalte, die zwingend vorhanden sein sollten?

a)	Überhaupt nicht	(1)	(2)	(3)	(4)	(5)	Auf jeden Fall
b)	Überhaupt nicht	(1)	(2)	(3)	(4)	(5)	Auf jeden Fall

24. Wie bewerten Sie die angegebene Agg_C3 im Allgemeinen?

a)	Sehr schlecht	(1)	(2)	(3)	(4)	(5)	Sehr gut
b)	Sehr schlecht	(1)	(2)	(3)	(4)	(5)	Sehr gut

25. Wie bewerten Sie die angegebene Agg_C3 hinsichtlich Ihrer eigenen Zusammenfassung? Im Vergleich zu meiner von Hand verfassten Aggregation ist Agg_C3…

a)	Viel schlechter	(1)	(2)	(3)	(4)	(5)	Viel besser
b)	Viel schlechter	(1)	(2)	(3)	(4)	(5)	Viel besser

- **Agg_C4:** Aggregation von Cluster 4 („Warren drops out“):

  Break elizabeth warren drop out of the 2020 presidential race after a disappointing super tuesday show nbc. @pamelageller trump i can not take all this win breaking sen elizabeth warren be drop out of the presidential race accord to the new york time it be now down to joe biden vs bernie sanders. #breaking on #oann massachusetts senator elizabeth warren drop out of the democratic presidential race just day after the onetime frontrunner could not win a single super tuesday state not even she own.

26. Wie bewerten Sie die Verständlichkeit der angegebenen Agg_C4?

a)	Sehr schlecht	(1)	(2)	(3)	(4)	(5)	Sehr gut
b)	Sehr schlecht	(1)	(2)	(3)	(4)	(5)	Sehr gut

27. Enthält die angegebene Agg_C4 die wichtigsten Informationen des Clusters?

a)	Überhaupt nicht	(1)	(2)	(3)	(4)	(5)	Auf jeden Fall
b)	Überhaupt nicht	(1)	(2)	(3)	(4)	(5)	Auf jeden Fall

28. Fehlen in der angegebenen Agg_C4 noch Inhalte, die zwingend vorhanden sein sollten?

a)	Überhaupt nicht	(1)	(2)	(3)	(4)	(5)	Auf jeden Fall
b)	Überhaupt nicht	(1)	(2)	(3)	(4)	(5)	Auf jeden Fall

29. Wie bewerten Sie die angegebene Agg_C4 im Allgemeinen?

a)	Sehr schlecht	(1)	(2)	(3)	(4)	(5)	Sehr gut
b)	Sehr schlecht	(1)	(2)	(3)	(4)	(5)	Sehr gut

30. Wie bewerten Sie die angegebene Agg_C4 hinsichtlich Ihrer eigenen Zusammenfassung? Im Vergleich zu meiner von Hand verfassten Aggregation ist Agg_C4…

a)	Viel schlechter	(1)	(2)	(3)	(4)	(5)	Viel besser
b)	Viel schlechter	(1)	(2)	(3)	(4)	(5)	Viel besser

- **Agg_C5:** Aggregation von Cluster 5 („Separating children“):

  We need a countdown clock begin the minute trump sign the order back down from he own policy of separate child from they parent at the border the clock will stop the minute the last child be reunite with they parent i predict the clock will run for month. There be 239 immigrant child already separate from parent will not be reunite with they parent and place in some sort of foster care so we kidnap the kid deport or indefinitely detain the parent then hand the kid over to the system some be so you young they do not know they parent. Dem force to show they hand after the president s eo to eliminate child from they parent & throw they into concentration camp make we every bit as angry as it make you when we have a president who raise tax to pay for national health care.

31. Wie bewerten Sie die Verständlichkeit der angegebenen Agg_C5?

a)	Sehr schlecht	(1)	(2)	(3)	(4)	(5)	Sehr gut
b)	Sehr schlecht	(1)	(2)	(3)	(4)	(5)	Sehr gut

32. Enthält die angegebene Agg_C5 die wichtigsten Informationen des Clusters?

a)	Überhaupt nicht	(1)	(2)	(3)	(4)	(5)	Auf jeden Fall
b)	Überhaupt nicht	(1)	(2)	(3)	(4)	(5)	Auf jeden Fall

33. Fehlen in der angegebenen Agg_C5 noch Inhalte, die zwingend vorhanden sein sollten?

a)	Überhaupt nicht	(1)	(2)	(3)	(4)	(5)	Auf jeden Fall
b)	Überhaupt nicht	(1)	(2)	(3)	(4)	(5)	Auf jeden Fall

34. Wie bewerten Sie die angegebene Agg_C5 im Allgemeinen?

a)	Sehr schlecht	(1)	(2)	(3)	(4)	(5)	Sehr gut
b)	Sehr schlecht	(1)	(2)	(3)	(4)	(5)	Sehr gut

35. Wie bewerten Sie die angegebene Agg_C5 hinsichtlich Ihrer eigenen Zusammenfassung? Im Vergleich zu meiner von Hand verfassten Aggregation ist Agg_C5…

a)	Viel schlechter	(1)	(2)	(3)	(4)	(5)	Viel besser
b)	Viel schlechter	(1)	(2)	(3)	(4)	(5)	Viel besser

Abschließende Fragen – Teil 3:

Sie haben die automatisiert generierten Zusammenfassungen ohne Vorwissen gelesen und bewertet. Nehmen Sie nun bitte folgende Sachverhalte, die Eigenheiten/Folgen des Erstellungsprozesses sind, zur Kenntnis:

- Jede Aggregation ist auf maximal 3 Sätze festgelegt, die (wenn möglich) auch immer ausgeschöpft werden. Eine Folge hiervon sind u.a. sich teils überschneidende Informationen.
- Je häufiger bestimmte Informationen vorkommen, desto wichtiger werden sie im Allgemeinen eingestuft.
- Alles außer dem jeweils ersten Buchstaben eines jeden Satzes ist grundsätzlich klein geschrieben.
- Die meisten Wörter in den Aggregationen sind auf ihre Grundform reduziert, z.B. (*„am“, „are“, „is“ → „be“*), *(„his“ → „he“*) oder (*„goes“, „went“ → „go“*).

Bitte beantworten Sie die Fragen 11 – 35 in Anbetracht dieser neu gewonnenen Informationen erneut! Nutzen sie dafür die unter **b)** aufgeführten Antwortmöglichkeiten! Beantworten Sie die Fragen bitte auch, falls sich Ihr Meinungsbild nicht verändert haben sollte!

Falls Sie Ihrerseits noch Anmerkungen oder Vorschläge haben, können Sie diese im nachfolgenden Textfeld hinterlassen:

VIELEN DANK

Appendix B

B.1 Code

Link to the code:
`https://git.rz.uni-augsburg.de/rudenkol/diss`

B.2 Publications

A Database Approach for Categorical Preferences on Hierarchies
P. Roocks, F. Wenzel, L. Rudenko, M. Endres, W. Kießling
9th Multidisciplinary Workshop on Advances in Preference Handling,
in conjunction with IJCAI-15, Buenos Aires, Argentina

A Preference-based Stream Analyzer
L. Rudenko, M. Endres, P. Roocks, W. Kießling
ECML PKDD 2016 Workshop on Large-scale Learning from Data Streams in Evolving Environments, Riva del Garda, Italien, Sep. 2016

Personalized Stream Analysis with PreferenceSQL
L. Rudenko, M. Endres
Workshop Präferenzen und Personalisierung in der Informatik PPI17 @ BTW 2017, Stuttgart, Deutschland

Preference-based Stream Analysis for Efficient Decision-Support Systems
L. Rudenko
Doctoral Consortium @ ADBIS 2017, Nicosia, Zypern

Real-Time Skyline Computation on Data Streams
L. Rudenko, M. Endres
New Trends in Databases and Information Systems @ ADBIS 2018, Budapest, Hungary

Analyzing and Clustering Pareto-Optimal Objects in Data Stream
M. Endres, J. Kastner and L. Rudenko
In M. Sayed-Mouchaweh (Eds.): "Large-scale Learning from Data Streams in Evolving Environments", Springer, 2018

A Tour of Lattice-Based Skyline Algorithms
M. Endres and L. Rudenko
In M. K. Habib (Eds.): "Emerging Investigations in Artificial Life Research and Development", IGI Global, 2018

Preference-based Twitter Analytics
L. Rudenko, C. Haas
12th Multidisciplinary Workshop on Advances in Preference Handling
M-PREF2020 on ECAI 2020, Santiago de Compostela, Spain (online)

Analyzing Twitter Data with Preferences
L. Rudenko, C. Haas and M. Endres
New Trends in Databases and Information Systems @ ADBIS 2020, Lyon, France (online)

List of Figures

List of Tables

List of Algorithms