CLUSTERDATASPLIT: Exploring Challenging Clustering-Based Data Splits for Model Performance Evaluation

Hanna Wecker[1,2] **Annemarie Friedrich**[2] **Heike Adel**[2]
[1]Ludwig-Maximilians-University, Munich, Germany
[2]Bosch Center for Artificial Intelligence, Renningen, Germany
[1]hanna.wecker@gmx.de [2]firstname.lastname@de.bosch.com

Abstract

This paper adds to the ongoing discussion in the natural language processing community on how to choose a good development set. Motivated by the real-life necessity of applying machine learning models to different data distributions, we propose a clustering-based data splitting algorithm. It creates development (or test) sets which are lexically different from the training data while ensuring similar label distributions. Hence, we are able to create challenging cross-validation evaluation setups while abstracting away from performance differences resulting from label distribution shifts between training and test data. In addition, we present a Python-based tool for analyzing and visualizing data split characteristics and model performance. We illustrate the workings and results of our approach using a sentiment analysis and a patent classification task.

1 Introduction

In natural language processing (NLP), the standard approach for tuning and selecting machine learning models is by means of using a held-out development set. However, recent work has pointed out that evaluation scores on a development set are often not indicative of the model performance on an unseen test set (Reimers and Gurevych, 2018; Zhou et al., 2020). In addition, it is an open research question how to choose a good development set. While Gorman and Bedrick (2019) suggest to use random splits instead of a given benchmark development set, Søgaard et al. (2020) argue that randomly selecting a development set is not the best option either. This currently ongoing discussion in the NLP community highlights the need for more extensive research on model development using a variety of data splits.

In this paper, we directly add to this discussion by proposing a strategy in which models are evaluated in a setup that is challenging given the available dataset. For machine learning models, it is of utmost importance that they are applicable to different data distributions, possibly even coming from different domains. We argue that models should also be tested under data splits reflecting such real-world settings. Therefore, we propose a **clustering-based data splitting approach** that creates data splits where the development or test data differ from the training set. Our clustering algorithm ensures a similar label distribution across the produced cross-validation folds in order to abstract away from challenges due to label distribution shifts.

In addition, we present CLUSTERDATASPLIT, a **suite of Jupyter notebooks** implementing several possibilities for splitting data into training and development sets, or into folds for cross-validation. In addition, our tool provides functionalities for visualizing different data splits and thus may help clarifying their influence on model performance. Furthermore, it offers several ways to inspect the data, such as visualization of dataset key figures, scatter plots, label distributions and sentence length distributions. The tool is publicly available.[1]

In sum, our contributions are as follows: (i) We propose a clustering-based data splitting algorithm that creates a challenging evaluation setup and has the potential to reveal difficulties when the model is applied on data that deviates from the training data (Section 3). (ii) We present CLUSTERDATASPLIT, a tool that allows to split data into training and development sets and provides different visualizations for analyzing the data splits as well as model performance (Section 4). Finally, we demonstrate a worked example of using our data inspection tool as well as results for our clustering-based data splitting methods for two sequence classification

[1]https://github.com/boschresearch/clusterdatasplit_eval4nlp-2020

Proceedings of the First Workshop on Evaluation and Comparison of NLP Systems (Eval4NLP), pages 155–163,
November 20, 2020.

tasks, sentiment analysis and patent classification (Section 5).

2 Related Work

In this section, we give an overview of related work on evaluation and data splitting techniques as well as analysis tools in the NLP community.

2.1 Data Splits for Model Evaluation

Gorman and Bedrick (2019) show that model rankings on standard splits (Collins, 2002) can often not be reproduced using randomly generated splits. In a follow-up study, Søgaard et al. (2020) find that evaluation results on random splits are often too optimistic, even for in-domain test samples. In order to make the data splits more challenging, they introduce heuristic splits based on sentence length and adversarial splits based on Wasserstein distance. The clustering-based data split we propose in this paper follows the same idea of creating a challenging evaluation setup. In contrast to the adversarial splits proposed by Søgaard et al. (2020), our splitting strategy controls for label distribution, allowing to abstract away from the effect of different label distributions on the evaluation score (Johnson and Khoshgoftaar, 2019; Buda et al., 2018).

2.2 Challenging Evaluation Sets

Another direction of work aims at tailoring more challenging test sets by either grouping or handcrafting datasets. These approaches typically require manual work for each dataset. Hendrycks et al. (2020), for instance, introduce a robustness benchmark[2] by assigning similar datasets as out-of-distribution (OOD) test sets. In their experiments, they show that the OOD test setting leads to severe performance drops for many models except transformers. Gardner et al. (2020) create contrast sets[3] for commonly used NLP benchmark datasets by adding hand-crafted data points for each test set example. Another direction of creating challenging evaluation sets comes from the idea of adversarial training (Szegedy et al., 2013; Goodfellow et al., 2015). In the context of NLP, the creation of adversarial examples typically involves task and dataset specific methods and often relies on hand-crafted rules or other forms of human influence. Examples are reading comprehension or question answering datasets with altered questions or documents (Jia and Liang, 2017; Wallace et al., 2019) or machine translation datasets for which typos are introduced (Belinkov and Bisk, 2018). In contrast to all those approaches, our data splitting approach is purely data-driven and creates a challenging evaluation setting within one dataset fully automatically.

2.3 Tools for Analyzing NLP models

Existing NLP model analysis tools are often tailored towards specific tasks or models (e.g., Wang et al., 2019; Zhou et al., 2020). In the remainder of this section, we give examples for model-agnostic tools as they are more related to our tool.
Graliński et al. (2019) introduce GEVAL,[4] a tool for identifying features in the test set (e.g., n-grams) which are especially challenging to models. The tool CHECKLIST[5] by Ribeiro et al. (2020) explores different model capabilities, such as robustness, vocabulary or temporal understanding. Furthermore, it supports the creation of test examples via templates. In contrast to those two tools, our tool offers visualizations of a variety of statistically interesting aspects of data splits in order to better understand model behaviours. Wu et al. (2019) provide an interactive tool for error analysis called ERRUDITE.[6] It supports, i.a., automated counterfactual rewriting for testing hypotheses about errors. In contrast to all mentioned tools, our tool implements different data splitting techniques, making it easy to compare model performance when using different data splits.

3 Clustering-based Data Splitting

We here propose a novel algorithm dubbed **Size and Distribution Sensitive K-means (SDS K-means)** which has two important properties relevant to generating challenging clustering-based cross-validation folds. The SDS K-means algorithm produces clusters that (1) each have approximately the same size, i.e., a similar number of data points, and that (2) are controlled for label distribution. In the default case, all clusters have a similar label distribution. By clustering the data points, we ensure that training and development data are different (for now in a lexical sense), hence creating a challenging evaluation setup. The SDS K-means algorithm thereby overcomes the following two

[2] https://github.com/camelop/NLP-Robustness
[3] https://allennlp.org/contrast-sets
[4] https://gonito.net/gitlist/geval.git/
[5] https://github.com/marcotcr/checklist
[6] https://github.com/uwdata/errudite

difficulties: (1) Varying cluster sizes: If clusters had different sizes, performance differences could simply be attributed to varying amounts of training data. (2) Varying label distributions: If clusters had differing label distributions, performance differences could be primarily due to label distribution mismatches between training and test data. Hence, when using SDS K-means generated data folds with similar label distributions per cluster, differences in model performance can be attributed to qualitative rather than quantitative differences between the folds. In the experiments of this paper, we keep the label distribution fixed. If the user wants to deviate from the default case, s/he can also use the SDS K-means algorithm to generate folds with varying label distributions, and thereby also investigate the effects of different label distributions on model performance.

In the following, we describe technical details and the derivation of our algorithm. All algorithms described in this section produce K clusters that are intended to be used in K-fold cross-validation.

3.1 Preprocessing

As a prerequisite for clustering, we transform the text data (sentences or clauses in our case) into vector representations. First, each token is turned into a vector representation using a pre-trained Word2Vec (Mikolov et al., 2013) model. Then, for each input example the word vectors are averaged. The vector representations are centered and scaled, and dimensionality reduction by principal component analysis is performed. The vectors obtained by these preparation steps then serve as input for the K-means based algorithms.

3.2 K-means and Size Sensitive K-means

For the generation of clustering-based data splits, we decided to work with K-means based clustering algorithms because of their low time complexity and high computing efficiency (Xu and Tian, 2015). The standard **K-means** algorithm (Lloyd, 1982) belongs to the group of partitioning clustering algorithms, i.e., the number of clusters to be formed needs to be specified beforehand. It is an expectation maximization algorithm that has the goal of minimizing the cluster-internal variances. As such, it iterates between an *expectation* step in which data points are assigned to clusters, and a *maximization* or *update* step in which cluster centers are re-calculated. The standard K-means algorithm produces clusters with strongly varying size and label distributions.

The **Same Size K-means** algorithm is a variant of the K-means algorithm that ensures that all clusters are assigned approximately the same number of data points. We implement this algorithm following a tutorial[7] by Schubert and Zimek (2019). During the initial assignment step, points are assigned to the different cluster centers following an order measure, which corresponds to the difference in distance from the point to the closest and the furthest cluster center. This means that points which have the highest absolute difference in distance from closest to furthest cluster center are assigned to their closest cluster center first. Once one of the clusters reaches its maximum size, the order measure is re-calculated and points are again sorted before continuing the assignment process. In the following, the algorithm iterates between a maximization step in which cluster centers are recalculated and an update step that differs from standard K-means as follows. During the update step, data points can swap assigned clusters in a 1-on-1 fashion if the swap is associated with a decrease of the overall cluster-internal variances. While this algorithm ensures that cross-validation folds will be of equal size, the label distribution within the clusters may vary and hence result in favoring models that are misled by an unrealistic label distribution in the training or development data.

3.3 Size and Distribution Sensitive K-means Algorithm (SDS K-means)

As a remedy for the above mentioned problems, we propose an extension, the **SDS K-means** algorithm. Like the Same Size K-means algorithm, it consists of an initial assignment and swapping-based update steps. However, in this case, the maximum number of points per cluster are determined separately for each label, corresponding to the desired distribution of labels for each cluster as specified by the user. In the default case, the label distribution per cluster corresponds to the overall label distribution in the training data. The initial assignment and the update steps are conducted separately for each label. This ensures that the label distribution per cluster matches exactly the distribution specified by the user. The pseudo-code of the algorithm is outlined in Figure 1. We initialize the algorithm

[7] https://elki-project.github.io/tutorial/same-size_k_means

A Initial cluster assignment

1 Initialize cluster centers

2 Sort data points by labels

3 For each label:

- I Order data points by difference between distance to closest and to furthest cluster center that is still "open" for the respective label
- II Assign points in this order to the different clusters, until one of the clusters reaches its maximum size for the respective label, i.e., until we "close" the cluster for the respective label. (Each cluster has a specific maximum size for each label, i.e., the order in which the labels are assigned to the clusters does not change the clustering output.)
- III Iterate over I and II until all points for the respective label are assigned to clusters

B Update cluster assignment

1 Update cluster centers (using all data points belonging to the cluster, irrespective of their labels)

2 Sort data points by label

3 For each label:

- I For each data point, calculate difference between distance to current cluster center and distance to other cluster centers
- II Perform 1-on-1 swapping of points with positive difference at step I

4 Repeat step 1-3 until no swaps happen any more or maximum number of iterations is reached

Figure 1: Pseudo code for **SDS K-means algorithm**.

multiple times and choose the run with lowest average cluster-internal variances as the final partition.

4 CLUSTERDATASPLIT Tool

The tool CLUSTERDATASPLIT consists of three Jupyter Notebooks. Hence, using the tool requires basic Python skills. Communication between the tool and user code for machine learning models is based on .tsv files containing the text data instances and labels. Figure 2 illustrates the workflow and the separation of tasks between the tool and client code. Currently, our tool and algorithm only support sequence classification tasks with a single label per dataset instance. We leave extensions for sequence tagging tasks to future work.

4.1 DATA ANALYSIS

The first notebook provides an introduction to key NLP dataset characteristics, such as label distribution, sentence token length and token frequency. It serves for a first exploration of the data and its key figures before using the data for model training and evaluation.

4.2 CREATING DATA SPLITS

In order to generate clustering-based data splits, this notebook groups the data into a pre-defined number of groups, the so-called data folds. These data folds then serve as input in a cross-validation setting, where they are combined to build the data split in training and development data. To group the data, different K-means based algorithms, which are outlined in detail in Section 3, are available. Moreover, the tool also supports the generation of randomized partitions, which can be combined to form randomized (baseline) data splits.

For generating the data splits, the user has to input the complete training data in a .tsv format and select an algorithm to generate the data folds. In most cases, s/he will want to compare the SDS K-means and randomized splitting. The tool then generates an output file with the data point IDs and the fold ID information per data point. The user then has to input this information into his/her model training setup, using a cross-validation framework in which the model is trained K times, training on K-1 folds and evaluating on the remaining fold in each iteration.

To date, our vector representations of the input texts are mostly based on lexical information (see Section 3.1).[8] For clustering, we use the K-means implementation in the Python scikit-learn package (Pedregosa et al., 2011), and apply the initialization method of Arthur and Vassilvitskii (2007).

4.3 PERFORMANCE ANALYSIS

After the training and evaluation steps based on different data splits are completed, the user can input the predictions obtained on the different evaluation sets into the third notebook using a .tsv format. The notebook calculates performance statistics and analyzes the dependence of results on data split characteristics. For example, the notebook visualizes data split characteristics such as relative size

[8] In future versions of the tool, other representations reflecting, e.g., syntactic or contextualized word embedding information, may be included. However, we here opt for a simple lexically-based representation for clustering that does not intend to already capture too many features that may later on be used by the models themselves. If the user of the tool wants to substitute this input embedding method, s/he can easily do so by overwriting the respective Python functions.

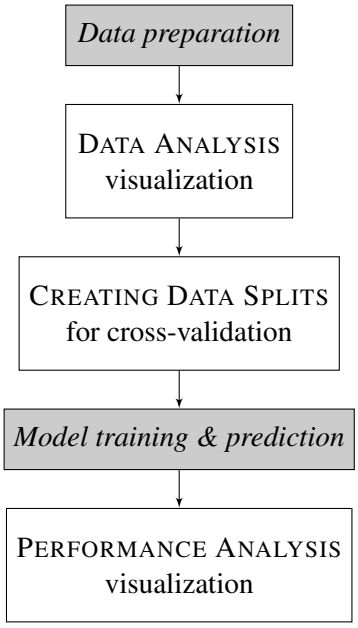

Figure 2: Workflow of the CLUSTERDATASPLIT tool. Grey boxes indicate client code, white boxes indicate functionalities included in Jupyter notebooks.

of the clusters, label distribution for the clusters and the mean sentence length. It also facilitates the comparison of different data splits and the performances obtained on these data splits.

5 Worked Examples

In this section, we give worked examples of using our proposed data splitting method for two sequence classification tasks, the Stanford Sentiment Treebank (SST) binary sentiment analysis task and a patent multi-class classification task.

5.1 Models

As our classifiers, we use simple neural networks for sequence classification based on BERT for SST (Devlin et al., 2019) and SciBERT (Beltagy et al., 2019) for the patent classification task. The latter model was trained on a corpus of scientific publications and is hence closer to the kind of language present in patents. For both models, we feed the CLS token into a linear layer that outputs logits corresponding to the number of classes and apply a softmax activation. For model training, we use a cross-entropy loss. We implement our models using the HuggingFace Transformers library (Wolf et al., 2019). The maximum sequence length of word piece tokens input to the BERT model is 128 and 256 for the two tasks, respectively. We use a batch size of 8, and AdamW (Loshchilov and Hutter, 2019) with learning rates of $4e^{-6}$ and $4e^{-5}$, respectively. Otherwise, we apply default parameters. We train the models for up to 100 epochs.

5.2 Stanford Sentiment Treebank

We here give an example of using our proposed analysis for a binary sequence classification task.

Dataset. The Stanford Sentiment Treebank (SST) (Socher et al., 2013) is based on movie review excerpts from the website `rottentomatoes.com`. For the task of binary sentiment classification, we use the SST-2 dataset, which is a variant of the original dataset containing only sentences and phrases with the label *positive* or *negative*. The dataset is slightly imbalanced with 44.28% negative and 55.72% positive labels. For our experiments, we use 61,398 sentences and phrases from the training and development part of the SST-2 dataset.

Data splits. We perform the SST classification experiments in two settings, using our SDS K-means based clustered and a randomized cross-validation setting. Figure 3 shows the visualization of the folds/clusters in two dimensions as generated by the CLUSTERDATASPLIT tool.

Performance results. Table 1 provides the results of training and evaluating models on clustering-based and randomized data splits in a cross-validation (CV) setting. The model summarized under the heading "CV-1" was trained on data folds 2-5 and evaluated on data fold 1. Note that the individual CV folds are not comparable between SDS K-means data splits (DS) and randomized DS, as each experimental setting uses different data splits. On average, the models trained and evaluated on the clustering-based data splits have a lower model performance than the models trained on the randomized data splits. Moreover, the standard deviation in model performance scores is higher for the clustering-based data splits than for the randomized data splits. Inspecting the differences between the clustered folds using CLUSTERDATASPLIT revealed that the sentences in the evaluation fold performing worst are on average shorter than the ones in the other folds, often consisting of short phrases that are difficult to classify also for human annotators. This underlines that the formation of training and development data based on the SDS K-means algorithm constitutes a more challenging evaluation environment than the random division of data

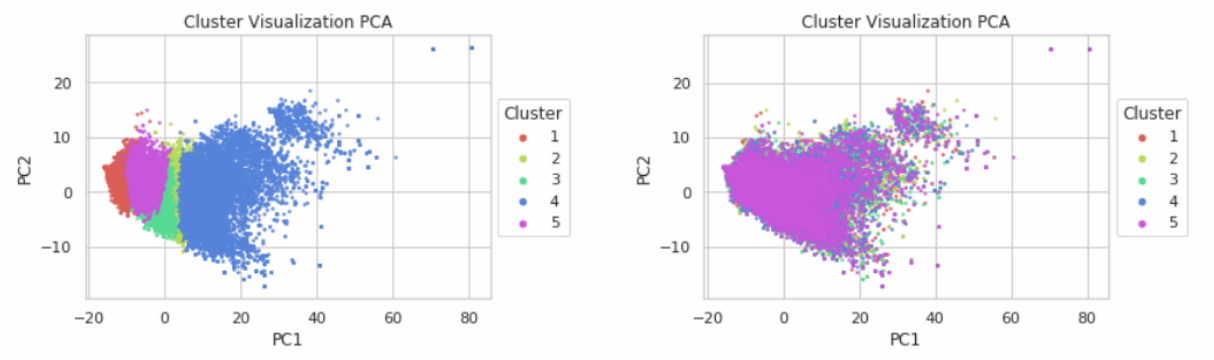

Figure 3: Visualization of data splits for SST dataset: SDS K-means clusters (left) vs. randomized (right).

	CV-1	CV-2	CV-3	CV-4	CV-5	Mean	Std
SDS K-means DS	94.5	89.8	95.2	93.9	92.8	93.2	1.9
Randomized DS	94.8	95.1	95.1	94.7	94.7	94.9	0.2

Table 1: F1 scores for binary sentiment classification on SST data. (Scores for individual folds are **not** comparable.)

into training and development data splits.

5.3 Patent Classification

In this section, we report results for a multi-class classification task, i.e., assigning the correct Cooperative Patent Classification (CPC) code to a patent.

Section	Description
A	Human Necessities
B	Performing Operations, Transporting
C	Chemistry, Metallurgy
D	Textiles, Paper
E	Fixed Constructions
F	Mechanical Engineering, Lighting, Heating, Weapons, Blasting
G	Physics
H	Electrity

Table 2: CPC patent classification scheme.[9]

Dataset. We retrieve a dataset of patents from USPTO[10] and represent each patent by its title and abstract. The latter are rather short, most sequences are shorter than 300 tokens. CPC codes indicate topics or application areas of a patent, and CPC classification is actually a hierarchical multi-label multi-class classification task. For simplicity, as our goal here is to demonstrate how our evaluation methods work for a simple multi-class classification task, we filter the dataset, keeping only instances that carry just a single label at the section level (7 labels, A-H, see Table 2). This leaves us with 6,458 instances with a skewed label distribution as shown in Figure 4. Of course, due the availability of machine-readable patents in large quantities, it would be possible to sample a larger training set in order to improve classification accuracy. However, our goal here is not to create an ideal CPC classifier but to highlight the importance of constructing challenging evaluation setups especially in low-resource settings. For reproducibility, we open-source the patents dataset together with our tool.

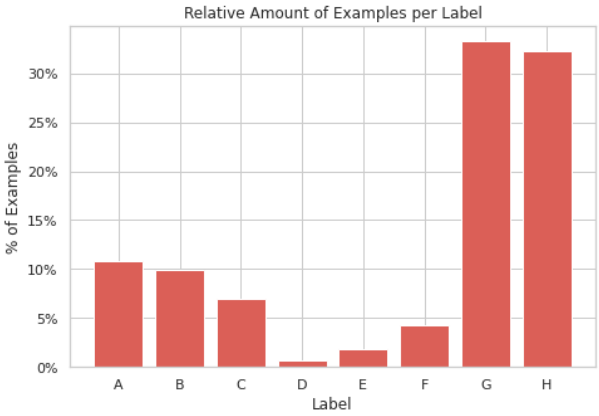

Figure 4: Distribution of labels in patents dataset.

Performance results. Table 3 shows the results obtained for the various cross-validation folds when assigning documents to folds randomly or

[9] https://www.uspto.gov/web/patents/classification/cpc/html/cpc.html

[10] https://www.patentsview.org/download

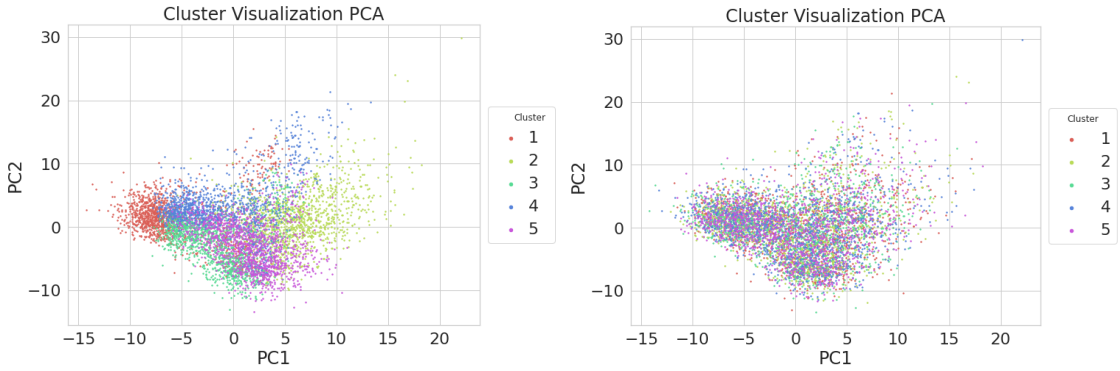

Figure 5: Visualization of data splits for patents dataset: SDS K-means clusters (left) vs. randomized (right).

	CV-1	CV-2	CV-3	CV-4	CV-5	Mean	Std
SDS K-means DS	53.4	57.4	43.3	61.3	51.1	53.3	6.1
Randomized DS	55.1	56.6	56.7	58.2	52.4	55.8	2.0

Table 3: Macro-average F1 scores for multi-class classification on patents dataset. (Scores for individual folds are **not** comparable.)

when using our clustering-based data splits. Macro-avg. F1 is computed as the average over per-class F1s.

For the patent classification task, differences in performance obtained in the SDS K-means evaluation setup differ even more strongly from the randomized cross-validation setup than in the case of the sentiment analysis task. Mean accuracy across the tasks is estimated as 55.8 in the randomized setting, but only as 53.3 in the SDS K-means setting. Again, as in the sentiment analysis task, the standard deviation is much higher when using clustered data folds. Two folds are notable, one exhibiting a much lower F1 and one having a much higher F1 score than the average. There are no sentence length differences in this case. Figure 5 shows that fold 3 concentrates in one region of the plot, while fold 4 has a much higher within-cluster variance. However, this does not yet explain why fold 3 instances are harder to classify, as for instance cluster 1 is also very concentrated in one region and achieves around average results. This finding hints at the fact that while our approach is able to generate challenging and diverse evaluation setups, further research is necessary to develop a systematic understanding of the observed performance differences in the produced clustering-based cross-validation setup. A very likely reason for the result in this case, as we are comparing macro-average F1, is low performance on the rare classes in the folds which possibly do not contain many "good" examples of these classes in the training folds.

6 Conclusion and Outlook

In this paper, we have introduced the concept of clustering-based data splits for model evaluation of sequence classification tasks. We have outlined the steps necessary to generate clustering-based data splits and described different K-means based algorithms for the creation of clustering-based data splits. Our newly proposed SDS K-means algorithm is able to generate clusters with equal (or controllable) size and label distribution. These properties make the algorithm perfectly suited for generating clustering-based data splits for challenging cross-validation experiments. Our worked examples show that model evaluation on clustering-based data splits generated by the SDS K-means algorithm is more challenging than model evaluation on randomly selected data splits.

The experiments conducted in this paper present a first step in exploring clustering-based data splits. Directions for **future research** involve the steps necessary to generate the clustering-based data splits, further exploration of the data splits and additional use cases. With regard to the generation of clustering-based data splits, different text vector representations or clustering algorithms, like for example density-based approaches, could be explored. Experiments with different text vector representations and clustering algorithms could shed some light on the impact of different cluster structures on the evaluation setup. Moreover, it would be interesting to study why clustering-based data splits seem

to have a stronger effect on some datasets than on others. Søgaard et al. (2020) introduce an experiment setting to compare the predictive performance of model evaluation on different data splits with regard to test sets coming from the same domain as the training data. Applying our clustering-based data splits in this experiment setting thus could deliver important information about the predictive quality of model performance scores obtained on clustering-based data splits.

Currently, our method works only for classification tasks and clustering is performed mainly based on lexical information. Hence, another interesting direction for future work is extending our ideas to data splitting for sequence tagging tasks, and to integrate other types of information such as syntactic features.

Acknowledgments

We thank Christian Heumann for his insightful comments regarding this work. We also thank the members of the NLP and Semantic Reasoning Group at the Bosch Center for Artificial Intelligence for their support and fruitful discussions on the ideas presented in this paper.

References

David Arthur and Sergei Vassilvitskii. 2007. K-Means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035, New Orleans, Louisiana. Society for Industrial and Applied Mathematics.

Yonatan Belinkov and Yonatan Bisk. 2018. Synthetic and natural noise both break neural machine translation. In *International Conference on Learning Representations*, Vancouver, Canada.

Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. SciBERT: A pretrained language model for scientific text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3615–3620, Hong Kong, China. Association for Computational Linguistics.

Mateusz Buda, Atsuto Maki, and Maciej A Mazurowski. 2018. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106:249–259.

Michael Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pages 1–8. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Matt Gardner, Yoav Artzi, Victoria Basmova, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, Nitish Gupta, Hanna Hajishirzi, Gabriel Ilharco, Daniel Khashabi, Kevin Lin, Jiangming Liu, Nelson F. Liu, Phoebe Mulcaire, Qiang Ning, Sameer Singh, Noah A. Smith, Sanjay Subramanian, Reut Tsarfaty, Eric Wallace, Ally Zhang, and Ben Zhou. 2020. Evaluating NLP models via contrast sets. *Computing Research Repository*, arXiv:2004.02709. Version 1.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, San Diego, California.

Kyle Gorman and Steven Bedrick. 2019. We need to talk about standard splits. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2786–2791, Florence, Italy. Association for Computational Linguistics.

Filip Graliński, Anna Wróblewska, Tomasz Stanisławek, Kamil Grabowski, and Tomasz Górecki. 2019. GEval: Tool for debugging NLP datasets and models. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 254–262, Florence, Italy. Association for Computational Linguistics.

Dan Hendrycks, Xiaoyuan Liu, Eric Wallace, Adam Dziedzic, Rishabh Krishnan, and Dawn Song. 2020. Pretrained transformers improve out-of-distribution robustness. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2744–2751, Online. Association for Computational Linguistics.

Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2021–2031, Copenhagen, Denmark. Association for Computational Linguistics.

Justin M Johnson and Taghi M Khoshgoftaar. 2019. Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):27.

Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory*, 28(2):129–137.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, Lake Tahoe, Nevada.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Nils Reimers and Iryna Gurevych. 2018. Why comparing single performance scores does not allow to draw conclusions about machine learning approaches. *Computing Research Repository*, arXiv:1803.09578. Version 1.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4902–4912, Online. Association for Computational Linguistics.

Erich Schubert and Arthur Zimek. 2019. ELKI: A large open-source library for data analysis–ELKI release 0.7.5 "Heidelberg". *Computing Research Repository*, arXiv:1902.03616. Version 1.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington. Association for Computational Linguistics.

Anders Søgaard, Sebastian Ebert, Joost Bastings, and Katja Filippova. 2020. We need to talk about random splits. *Computing Research Repository*, arXiv:2005.00636. Version 1.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. In *International Conference on Learning Representations*, Banff, Canada.

Eric Wallace, Pedro Rodriguez, Shi Feng, Ikuya Yamada, and Jordan Boyd-Graber. 2019. Trick me if you can: Human-in-the-loop generation of adversarial examples for question answering. *Transactions of the Association for Computational Linguistics*, 7:387–401.

Changhan Wang, Anirudh Jain, Danlu Chen, and Jiatao Gu. 2019. VizSeq: a visual analysis toolkit for text generation tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations*, pages 253–258, Hong Kong, China. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. HuggingFace's transformers: State-of-the-art natural language processing. *Computing Research Repository*, arXiv:1910.03771. Version 5.

Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel Weld. 2019. Errudite: Scalable, reproducible, and testable error analysis. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 747–763, Florence, Italy. Association for Computational Linguistics.

Dongkuan Xu and Yingjie Tian. 2015. A Comprehensive Survey of Clustering Algorithms. *Annals of Data Science*, 2(2):165–193.

Xiang Zhou, Yixin Nie, Hao Tan, and Mohit Bansal. 2020. The curse of performance instability in analysis datasets: Consequences, source, and suggestions. *Computing Research Repository*, arXiv:2004.13606. Version 1.