Job Title Classification - A Comparision of Vectorization and Classification Techniques for German Job Postings

Masterthesis

A thesis submitted in partial fulfilment of the requirments for the degree of the Master of Social and Economic Data Science at the University of Konstanz

submitted by

Rahkakavee Baskaran

01/950253

at

Universität Konstanz

Period of completion:

1st supervisor: Professor Doctor Susumu Shikano

2nd supervisor: Junior Professor Juhi Kulshresthra

Konstanz, February 4, 2022

Contents

List of Figures

List of Tables

Abbreviations

SVM Support Vector Machine 8

KNN K-Nearest-Neighbor 8

NB Naive Bayes 9

MLR Multinomial Logistic Regression 31

OA Overall Accuracy 38

KldB Klassifikation der Berufe 2010 6

ISCO International Standard Clasification of Occupations 16

BOW Bag of Words 9

TF-IDF Term Frequency-Inverse Document Frequency 7

TF Term Frequency 21

IDF Inverse Document Frequency 21

DF Document Frequency 21

CBOW Continous Bag of Words 22

1 Introduction

Job titles are key information within the area of labor market. They are helpful for job seekers to find jobs (Marinescu and Wolthoff, 2020); they are an essential component of job search engines (Slamet et al., 2018; Javed et al., 2015, 2016) and job recommendation systems (Malherbe et al., 2014); and lastly, they serve as a valuable data source for various analyses, such as job market trends (Martin-Caughey, 2021; Li et al., 2021), job perception (Smith et al., 1989; Boydston and Hirst, 2020) or social science analyses (Martin-Caughey, 2021). However, since job titles are not normalized, it is challenging to structure them appropriately for downstream tasks. Various institutions developed job taxonomies to structure and generalize job titles. Established taxonomies are, for example, the “International Standard Classification of Occupation” for the European job market or the “Klassifikation der Berufe 2010 (KldB)” for the German job market (Uter, 2020). Classifying job titles from job postings into classes based on these taxonomies is indispensable to improve job search engines or recommendation systems or to enable comprehensive analysis of the labor market.

In the literature, there exist some works on the classification of job titles (Zhu et al., 2016; Javed et al., 2015; Decorte et al., 2021; Wang et al., 2019; Neculoiu et al., 2016). While job title classification systems like “Carotene” (Javed et al., 2015) have been developed for English job titles, to the best of my knowledge, there have not been any classification attempts for German titles yet. However, accurate classification of job titles with the KldB taxonomy would facilitate several downstream tasks for the German job market. With the KldB 2010, a taxonomy was created that reflects the current trends in the labor market based on empirical and theoretical foundations, which gives an excellent basis for the normalization. In addition, most previous work does not have a solid database due to the lack of labeled data. This often leads researcher to resort to unsupervised methods (Decorte et al., 2021; Javed et al., 2015). For the German job market, the Federal Employment Agency provides job postings with labeled titles on their job board, creating large labeled data for text classification.

Over time, the literature on Nature Language Processing (NLP) has been virtually flooded with algorithms for text classification. Starting with traditional techniques followed by ensemble techniques, nowadays, deep learning methods dominate the research. Choosing a classification algorithm is far from trivial: no algorithm works equally well for every task. This makes exploration and comparison of different algorithms necessary.

However, not only the choice of the classifier itself is challenging, but also the input for the algorithms plays a significant role. Methods have been developed that take into account various properties, such as the context of the text or the consideration of word order. This results in vectors with different densities and dimensions. In order to develop a powerful algorithm, various techniques have to be explored and compared.

Task-specific requirements necessitate further adjustments in the training process. The focus of this paper is on the challenge of job titles being concise. Short texts suffer from sparseness, few word co-occurrences, missing shared context, noisiness, and ambiguity. The literature offers several approaches. Enriching features with additional knowledge has proven to be a promising method (Wang et al., 2014, 2017a).

Given the lack of a classification system for German job titles and the challenges this task entails, I define the following research goal: A comparison of different vectorization and classification methods for classifying German job titles under consideration of short text classification. The comparison should help to develop a German job title classification system based on the KldB taxonomy to facilitate downstream tasks for the German labor market. Based on the state of the literature, the comparison is carried out under three aspects: First, different feature extraction techniques are implemented and compared. The comparison includes the sparse vectors count vectorizer and Term Frequency-Inverse Document Frequency (TF-IDF) and the dense techniques word2vec, doc2vec, and BERT. Second, different classifiers from different research directions are considered. The traditional methods SVM and LR, the ensemble technique RF, and BERT as a deep learning method are applied and compared. Finally, the task-specific challenge of short texts is considered by enriching word2vec and doc2vec, each with a second model with additional knowledge from the taxonomy.

Some interesting trends emerge from the training. Bidirectional Encoder Representations from Transformers (BERT) as a vectorization technique is the clear winner. The sparse vectors perform better than word2vec and doc2vec, except for Random Forest (RF) where word2vec has similar performance to the sparse vectors. Except for BERT, the other classifiers developed label bias in different settings, which explains most of the performance scores. The additional knowledge may have positively affected the word2vec embedding technique, while it may have been detrimental to doc2vec.

This paper is structured as follows: The first section gives an overview of the related work to derive the research gap and the framework for the task. The following section presents the datasets and elaborates on the taxonomy. This is followed by the third chapter, with a detailed explanation of the pipeline and the associated methods. Finally a conclusion is drawn at the end of the paper and limitations are pointed out.

2 Related Work

Job Title Classification For the English job market, there are already some studies dealing with job title classification. In terms of classifiers, the corresponding work can be categorized into traditional classifiers or deep learning methods. Zhu et al. (2017) for example, use a K-Nearest-Neighbor (KNN) classifier in combination with document embedding as a feature selection strategy. Javed et al. (2015) rely on traditional methods as well, by combining a Support Vector Machine (SVM) classifier and a KNN classifier for their job recommendation system. In contrast, the approaches of Decorte et al. (2021), Wang et al. (2019) and Neculoiu et al. (2016) are based on deep learning methods. From a differentiated framework perspective, there is another dividing line between the approaches. As mentioned earlier, job title normalization can be considered a typical text classification task (Wang et al., 2019; Javed et al., 2015; Zhu et al., 2017). Decorte et al. (2021) and Neculoiu et al. (2016), however, formulate the task as a string representation approach of similar job titles.

While there is some work on job title classification for the English speaking job market, to the best of my knowledge, there have not been any classification attempts for the German job market. However, an accurate classification of job titles with the German KldB taxonomy would facilitate several downstream tasks for the German job market. Besides the lack of a classification system, most of the work on job title classification suffers from the lack of solid databases. Therefore, Decorte et al. (2021), for example, use skills mentioned in job advertisements to understand the meaning of job titles and to avoid manually labelling them. Javed et al. (2015) rely on a weakly supervised approach to get enough labelled data. The advantage of classifying for the German job market is that the Federal Employment Agency of Germany provides a dataset with job titles and the possibility of linking them with die KldB classes, which offers a huge and powerful training dataset which in turn allows for more flexibility in which algorithms are applicable. Following Wang et al. (2019); Javed et al. (2015); Zhu et al. (2017) the task will be framed as a text classification task.

Text Classification Algorithms During the last decades, researchers developed a various number of classifiers. In their survey of classifiers, Kowsari et al. (2019) summarize in their survey of classifiers, differentiate between three groups of approaches. The first group contains traditional methods like Naive Bayes (NB), SVM, KNN, Logistic Regression (LR) or Decision Trees (Vijayan et al., 2017; Colas and Brazdil, 2006; Kowsari et al., 2019; Sebastiani, 2002). Deep learning methods like Convolutional Neural Network (CNN) or Recurrent Neural Networks (RNN), which are currently the most advanced algorithms for NLP, form the second group. The last group consists of ensemble learning techniques like Boosting and Bagging. Each group can be further split mainly into supervised and unsupervised learning techniques. Since labeled data is available for the classification task in this study, the focus will be on supervised learning techniques.

Classification algorithms can be selected on different criteria. Certainly, one of the most important criteria is performance. Currently, deep learning methods often outperform traditional methods and ensemble techniques. Deep Learning methods are advanced methods of traditional machine learning algorithms. In contrast to traditional methods, they do not require a substantive understanding of feature extraction since they automatically extract important features. Many comparative studies on traditional and embedding techniques vs. deep learning text classification tasks show the strength of deep learning. Wang and Qu (2017) compared SVM and KNN for web text classification against CNN. Their results reveal a better performance of CNN over the traditional methods. Hassan and Mahmood (2017) likewise show that CNN, but also RNN outperform traditional methods with Bag of Words (BOW) feature extraction. These results are corroborated by Kamath et al. (2018). Furthermore, González-Carvajal and Garrido-Merchán (2020) compared the current state-of-the-art deep learning model BERT with traditional methods using the TF-IDF feature selection method and show clear outperformance by BERT.

Although deep learning models often outperform traditional methods in these comparative studies, not all classifiers achieve good results over all applications. In contrast to traditional methods, deep learning models also usually require millions of data to train an effective model (Chauhan and Singh, 2018). Thus, deep learning methods are not necessarily always the right choice. For example, Zhang et al. (2015), conducted experiments on character-level CNN and compared them to different traditional models, like BOW or Bag of Ngrams with TF-IDF and LR. For smaller and moderate size news datasets, the traditional methods, except for Bag of Means, performed well, and some of them outperformed CNN. For bigger datasets, CNN worked better. Another study from Yan et al.

(2018) relies on a siamese CNN deep learning approach with few-shot learning for short text classification using different Twitter benchmark data. They compared their results to some baseline deep learning methods and traditional methods, including SVM, ensemble techniques, and LR. Although the siamese CNN outperformed all the other methods clearly, some traditional methods outperformed the baseline deep learning methods for specific datasets.

Comparisons of ensemble techniques, traditional methods, and methods within the traditional methods indicate that not all classifications perform equally good or poor for all tasks. A comparative analysis with LR, RF and k-nearest neighbor, using TF-IDF, for news text classification indicates a good performance of LR and RF compared to KNN, wherby LR performed better then RF (Shah et al., 2020). A study from biomedical classification shows that among RF, NB with TF-IDF and SVM the best performance was achieved by SVM (Danso et al., 2014).

Although performance is essential, other criteria like the transparency of the algorithm, the interpretability, and efficiency in terms of the runtime are not irrelevant. Methods like NB or LR are much faster than neural networks or SVMs. In terms of transparency and interpretability, algorithms like Decision Tree or LR are more intuitive and easier to understand and interpret. In contrast, deep learning models and SVM rely on more complex computations. Especially deep learning lacks transparency (Maglogiannis, 2007). As for BERT, although it usually outperforms other machine learning algorithms for text classification, in general "there are more questions than answers about how BERT works" (Rogers et al., 2020, 853). It is, for example, not well-understood so far what exactly happens during the fine-tuning process of BERT (Merchant et al., 2020). In spite of the other qualification criteria, the main focus of this analysis is on performance.

Vectorization Methods Besides the classifier, it is also important for the performance with which inputs the classifiers are fed. Texts must be converted into a numerical representation to make them machine-readable (Singh and Shashi, 2019). The numerical vector representations of a text or document can be divided into sparse and dense vectors, also called word embeddings. Sparse vectors, relying on the BOW model, are high-dimensional vectors with many zeros, while word embedding techniques have a fixed-length representation (Almeida and Xexéo, 2019). Sparse vectors are for example TF-IDF or count vectorizer. Examples for word embedding techniques are word2vec, doc2vec, and BERT.

Unsuitable features considerably affect the performance of the classification algorithms

(Cahyani and Patasik, 2021). The correct selection of a feature extraction technique depends on many factors, like the length of the dataset or the specific domain (Arora et al., 2021). Empirically, this is reflected in the diverse and conflicting studies in the literature. Considering the sparse vectors, Wendland et al. (2021) compared for fake news data TF-IDF and count vectorization. While they found slightly better results with TF-IDF, the results of Wang et al. (2017b) show no difference between them. Some studies from different domains demonstrate the strength of word2vec compare to TF-IDF (Arora et al., 2021; Rahmawati and Khodra, 2016), while others show opposite results (Zhu et al., 2016; Cahyani and Patasik, 2021). Shao et al. (2018) conclude that they find no clear picture between BOW models and word2vec. Comparing doc2vec and word2vec, Lau and Baldwin (2016) found in general good performance of doc2vec. However, the authors admit that the qualitative differences between both techniques are not clear. Both Shao et al. (2018) and Wang et al. (2017b) obtain the worst results for doc2vec compared to word2vec and BOW vectorization techniques. BERT shows overall a good performance (González-Carvajal and Garrido-Merchán, 2020). Nevertheless, Miaschi and Dell'Orletta (2020) conclude from their research finding that word2vec and BERT code similarly for sentence-related linguistic features.

Challenges of Job Title Classification Each text classification task has different challenges. One challenge of the presented task is the number of classes. As Li et al. (2004) show in their classification of tissue, multi-class classification is more complex than binary classification problems. Partly because most classification algorithms were designed for binary problems (Aly, 2005). Approaches for multi-class classification can be grouped into two types. Either binary algorithms can handle multi-class classification naturally, or the problem is decomposed into binary classification tasks (for the different subtypes, see Aly (2005)). The literature so far does not have a clear answer to solve multi-class classification problems. Different approaches, like boosting (Schapire and Singer, 2000) or CNN (Farooq et al., 2017) are applied. It is apparent, however, that many works use variations of SVM (Guo and Wang, 2015; Tomar and Agarwal, 2015; Tang et al., 2019a).

Another important issue for text classification is the length of input documents. Job titles are short texts with often not more than 50 characters. Short texts suffer from sparseness, few word co-occurrences, missing shared context, noisiness, and ambiguity. These attributes make it challenging to construct features that capture the meaning of the text. Furthermore, traditional methods are based on word frequency, high word cooccurrence, and context, which is why they often fail to achieve high accuracy for short texts (Song et al., 2014; Wang et al., 2017b, 2014; Alsmadi and Gan, 2019). Besides this, short texts are also often characterized as having a lot of misspelling and informal writing. The last two attributes of short texts are indeed a problem, e.g., for Twitter data, which is a popular topic for short text data (Karimi et al., 2013; Sriram et al., 2010; Yan et al., 2018). However, these attributes play little or no role in the job title classification, especially compared to the other stated issues, since job postings are usually reviewed and controlled thoroughly before release. Due to this, only research concerning the first mentioned attributes is considered in the following.

A popular approach proposed by many researchers for improving short text classification is to add additional knowledge to the features. Many studies of Twitter short texts demonstrate the power of this approach. Karimi et al. (2013), for example, use a BOW approach for disaster classification of Twitter posts. They experiment with different features enriched by additional information. While, for example, generic features like the number of hashtags improved classification, other information like incident-specific features only helped in specific settings. All in all, the use of BOW with specific features delivers quite good performance. Similarly, Sriram et al. (2010) achieved a good performance for Twitter short text messages, using features extracted manually from the short text itself.

Criticism of the BOW approach for short text classification is raised by Wang et al. (2014) since BOW results usually in high dimensional data. They state that this is much more harmful to short texts because they are short and sparse. They propose a "bag of concept" approach using a knowledge base. The knowledge base is used to learn concepts for each category and find a set of relevant concepts for each short text. Following this, Wang et al. (2017a) use an enriching concept for short text classification. They use a concept vector combined with a taxonomy knowledgebase which indicates how much a text is related to the concept. Those are merged with the word embeddings. In addition, they add character-level features.

In general, in short text classification, the question arises whether to represent the features as dense or sparse vectors. Based on their comparison of TF-IDF and the count vectorizer against the dense vectorizer word2vec and doc2vec, Wang et al. (2017b) conclude that among the classifiers NB, LR and SVM, the sparse vectorizers achieve the highest accuracy. Chen et al. (2019), conversely, see limitations in sparse representation as it cannot capture the context information. In their work, they integrate sparse and dense representation into a deep neural network with knowledge-powered attention, which outperforms state-of-art deep learning methods, like CNN, for Chinese short texts.

In contrast to the above mentioned approaches, Sun (2012)pursues a completely different strategy. Instead of enriching the features, he focused them on specific keywords. In order to select the essential keywords, he used TF-IDF approaches, some with a clarity function for each word. Using LR he accomplished persuasive results for his classification.

Instead of feature enrichment according to Song et al. (2014), some researchers also apply feature dimensionality reduction and extraction of semantic relationship techniques using, for example, Latent Dirichlet approaches.

Concerning the classifiers, there is no consensus approach for short text classification. For traditional approaches, the results of Wang et al. (2017b) indicate that LR and SVM perform best, while Khamar (2013) displays that k-nearest neighbor has the best achieved accuracy. Song et al. (2014) propose to use ensemble techniques. In combination with enriched features Bouaziz et al. (2014), for example, achieve better results as for LR with ensemble techniques. Similar to job title-specific work, more recent work prefers deep learning methods, mostly CNN (Chen et al., 2019).

Implications of State of Research There are three consequences emerging from the above-discussed literature. Firstly, appropriate feature selection or vectorization plays a decisive role in the performance. For that reason, I implement several feature extraction techniques covering sparse and dense vectors. For sparse vectors, I transform the data with count vectorizer and TF-IDF vectorizer. Word2vec, doc2vec, and BERT embedding built the group of dense vectorization techniques. The discussion of the different techniques will be the first pillar of the comparison.

Secondly, relying on one classification does not seem to be a reasonable option. Instead, experimenting and exploring different traditional, deep learning, and ensemble classifiers allows for identifying the best classifier based on the task and the data (Maglogiannis, 2007). Therefore, I use four classifiying techniques. Two of them are traditional methods. The literature shows that SVM and LR are well compatible with other techniques, which is why I choose both of them. I also include RF as an ensemble technique. As the last method, I implement a BERT deep learning model since it is the state-of-art method for text classification. The evaluation of different classifications builds the second pillar of the comparison.

Thirdly, from the two challenges presented, the focus of this paper is on the classification of short texts. The literature on short text classification reveals two issues. First, there are different results on whether sparse or dense techniques are better suited. Testing different sparse and dense vectorization techniques allows adressing this issue. Second, most of the solutions include additional knowledge. Therefore, I introduce a second model for each word2vec and doc2vec with additional knowledge from the taxonomy. The model comparison between these models forms the last pillar of the comparison.

3 Job Title Data and Taxonomy

The training data consists of two datasets.[1] The classes are extracted from the first dataset, referred to below as the KldB dataset. The KldB dataset contains all information of the KldB taxonomy. The second dataset, called the job title dataset, contains the necessary data from the job titles. In the first part of this chapter, a brief explanation about the taxonomy structure and both datasets is given. The second part consists of a descriptive analysis of the class distribution of the data.

3.1 KldB 2010

The KldB dataset is structured hierarchically with five levels, with each level containing a different number of classes. The classifiers are trained for level 1 and level 3. In the following, these classes are also referred to as *kldb*. On level 1, each class has an id of length 1 with a number from 0 to 9. Table 1 shows the ten classes of level 1 with their class names. On level 2, each of the ten classes is divided into one or more subclasses having a class id of length 2, with the first digit indicating the class of level 1 and the second digit the class of level 2. An overview of all five levels with an example of classes is given in table 2. This procedure ultimately leads to class ids of length five on level 5. An occupation can be classified on every level in the taxonomy. Considering the classes of the example in table 2, the job title “java developer” could be classified on level 5 to the class 43412. From this id, it is also derivable that the job title belongs, for example, on level 3 to the class “Sofwareentwicklung” (Bundesagentur für Arbeit, 2011a,b; Paulus et al., 2013)

Furthermore, the KldB has two dimensions. The first dimension, the so-called “Berufsfachlichkeit”, structures jobs according to their similarity in knowledge, activities, and

[1]Both datasets are provided by cause&effect DFSG UG for this study. They are not publicly available in this format.

IDs KldB 2010	Berufsbereich (Level 1)
1	Land-, Forst- und Tierwirtschaft und Gartenbau
2	Rohstoffgewinnung, Produktion und Fertigung
3	Bau, Architektur, Vermessung und Gebäudetechnik
4	Naturwissenschaft, Geografie und Informatik
5	Verkehr, Logistik, Schutz und Sicherhe
6	Kaufmännische Dienstleistungen, Warenhandel, Vertrieb, Hotel und Tourismus
7	Unternehmensorganisation, Buchhaltung, Recht und Verwaltung
8	Gesundheit, Soziales, Lehre und Erziehung
9	Sprach-, Literatur-, Geistes-, Gesellschafts- und Wirtschaftswissenschaften, Medien, Kunst, Kultur und Gestaltung
0	Militär

Table 1: Overview of classes Level 1 - Berufsbereiche (adapted from (Bundesagentur für Arbeit, 2011b))

jobs, reflected in the first four levels. Considering the job titles “fullstack php-entwickler” and “java developer”, it is reasonable to classify both on level 1 to ”Naturwissenschaft, Geografie and Information” because they are related to computer science. It also makes sense to classify them, for example, to 4341 as both are concerned with software development. On level 5, then, a second dimension is introduced - the ”Anforderungsniveau”. This dimension gives information on the level of requirement for a job summarized in table 3. From the class ID of job title “Java Developer”, we can see that the job has been assigned to the second requirement level since the last digit is a two (Bundesagentur für Arbeit, 2011a,b; Paulus et al., 2013).

Name	Level	Number of classes	Example
Berufsbereiche	1	10	4: Naturwissenschaft, Geografie und Informatik
Berufshauptgruppen	2	37	43: Informatik-, Informations- und Kommunikationstechnologieberufe
Berufsgruppen	3	144	434: Sofwareentwicklung
Berufsuntergruppen	4	700	4341: Berufe in der Sofwareentwicklung
Berufsgattungen	5	1286	43412: Berufe in der Sofwareenetwicklung - fachlich ausgerichtete Tätigkeiten

Table 2: Levels of KldB with examples for each level (adapted from (Bundesagentur für Arbeit, 2011b))

The KldB 2010 thus represents a valuable and information-rich occupational classification scheme for Germany, that also reflects the current trends in the labor market (Paulus et al., 2013). It can capitalize, above all, on its sophisticated construction process. Instead of just including expert knowledge into the taxonomy, the development process is based on systematical consideration of occupations and statistical procedures for taxon-

level of requirement	class ID	name long	name short
1	xxxx1	Helfer- und Anlerntätigkeit	Helfer
2	xxxx2	fachlich ausgerichtete Tätigkeiten	Fachkraft
3	xxxx3	komplexe Spezialstentätigkeiten	Spezialist
4	xxxx4	hoch komplexe Tätigkeiten	Experte

Table 3: Level of requirements on level 5 (adapted from (Bundesagentur für Arbeit, 2011b))

omy development. Furthermore, the taxonomy was reviewed qualitatively several times concerning professions.

Considering the expressiveness, the KldB has some further advantages. Since the taxonomy is relatively recent, it reflects new job classes and market trends adequately. On top of that the taxonomy provides a powerful tool to organize job titles into simple requirement classes by including the second dimension. In addition, it also distinguishes between managerial, supervisory, and professional employees, which is a valuable information as well. Finally, the taxonomy also convinces with the possibility to switch to "International Standard Clasification of Occupations (ISCO)" through its IDs and thus to normalize jobs to a global standard (Bundesagentur für Arbeit, 2011b).

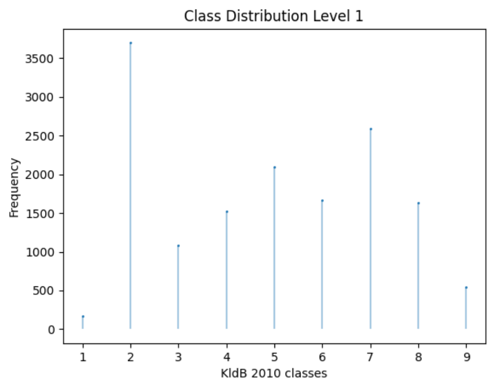
The KldB dataset contains different information related to the structure described above. Some search words are given besides the class label, the level, and the title, on level 5 for each *kldb*. There are two types of keywords: Firstly, job title search words that match the respective kldb; secondly, actual search words by which the associated kldb can be inferred. Therefore, these search words are beneficial knowledge for training classification algorithms because they contain *kldb* specific words that are also often present in the job titles. The search words will be termed additional knowledge in the rest of the work.

3.2 Job Title Data

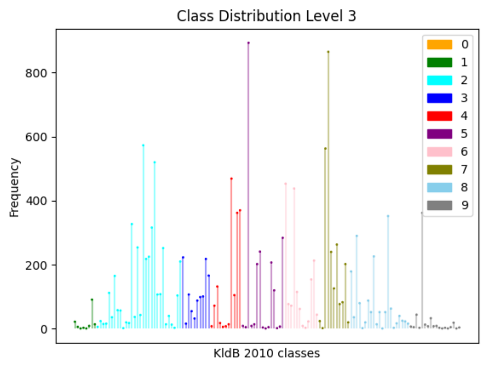
Job titles can be scraped from the Federal Employment Agency's job board. Employers must provide additional data for each job posting, including the job title, as well as an internal documentation code that indicates a class in the KldB taxonomy. There is an option to provide alternative documentation codes if more than one *kldb* is believed to fit to the job in question. The data contains job postings between June and November 2021. The documentation code is an internal number of the Federal Employment Agency, which can be uniquely assigned to a *kldb*, which, as already mentioned, is specified in the taxonomy data for each *kldb*. A snippet example of the scraped data as well as the matched training data is given is provided in the appendix.[2]

Initially, the training dataset contained a total of 269788 examples. However, during the training phase, it became apparent that there are problems, concerning the runtime and memory, especially for the SVM classifier. Due to limited resources, a sample with the same distribution must be taken from the complete dataset. A sample size of 15000

[2]There are two versions of the raw data since the Federal Employment Agency changed during the scraping phase the data structure. Both versions are given in the appendix.



(a) Class distribution level 1



(b) Class distribution level 3

Figure 1: Class distribution of the training data

examples proved to be feasible. The data is divded into training and test data, with 25% of the examples assigned to the test data.[3]

Figures 1a and 1b show the distribution of the classes in level 1 and level 3.[4] Both figures show the absolute number of examples for each *kldb* in the respective levels. In figure 1b the number of examples is colored with the belonging *kldb* level 1 class to give a better overview of the 144 *kldb* on level 3. In general, from both levels, it is clear that the data is not distributed equally. For the class distribution of level 1 on Figure 1a, class 1 and class 9 have really few examples, while class 2 has considerably more examples than all other classes. Although the data is imbalanced, all classes have at least some examples to get meaningful performance measurements. At level 3, the uneven distribution is even more apparent. Compared to level 1, there are many classes with only one example, which is problematic for obtaining interpretable measurements. This issue will be discussed in the results section. Note that *kldb* 0 is missing for level 1. Thus, it cannot be trained with the classifier. 0 stands for "Militär". Jobs in this category are generally not posted frequently on the Employment Agency Job Board page, so it was not possible to get examples for this *kldb*. On level 3 eight classes do not have any examples.

[3]Instead of splitting the training data once, the literature advises using cross-validation (Refaeilzadeh et al., 2009). Cross-validation involves splitting the data and validating it in more than one round to ensure that performance is not random. For computational reasons, however, I do not apply this method. However, the code to perform cross-validation is included in the repository. It also outputs the confidence intervals for cross-validation.

[4]The class distribution of the long dataset is given in the appendix.

4 Method

4.1 Overview of the Pipeline

The classification process is divided into several steps. Figure 2 gives an overview of the procedure. In the first step, the KldB and the job title data are focused on the necessary variables. This is done by matching the *kldb* ids from the KldB dataset with the internal documentation id from the job title data. In the following steps, the targeted data is preprocessed. This preprocessed data is then transformed into numerical vectors. For this purpose, different vector representations are created using five vectorization techniques.

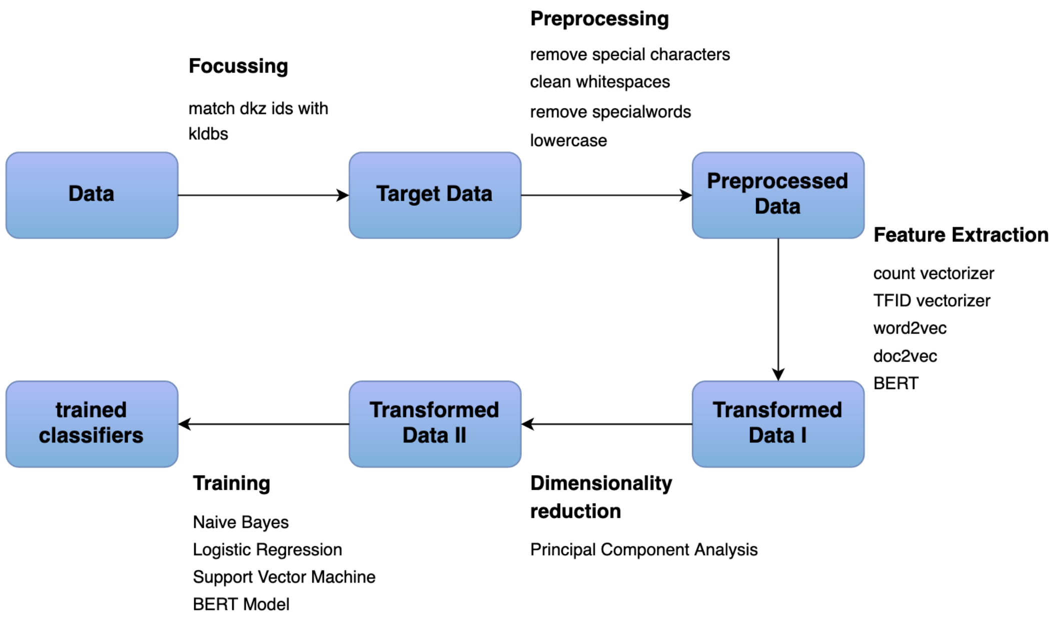


Figure 2: Training pipeline

The transformed data is then reduced to lower dimensions using principal component analysis. The resulting data is the input for the classifier.In the last step, the four classifiers are trained. Note that the BERT deep learning algorithm follows a different pipeline, which is why the algorithm is presented separately. Here, the data is put into the deep learning model directly after preprocessing, and the model is trained. The targeting of the data has already been described in the previous chapter. The procedure and the associated methods are explained in detail in the following.

4.2 Preprocessing

Having an important influence on the performance of a classifier (Uysal and Gunal, 2014; HaCohen-Kerner et al., 2020; Gonçalves and Quaresma, 2005), preprocessing is a crucial step that has to be taken before vectorization. There are several standardized practices for preprocessing like tokenization, stemming, stop word removal, and lowercasing (Alsmadi and Gan, 2019). The methods of preprocessing are justified below.

The first preprocessing step is removing special characters, which is necessary because most titles contain slashes and brackets to distinguish gender forms in occupational titles. Some job titles contain emojis, such as asterisks, which need to be removed because they are not related to specific job titles.

Capitalized words are usually converted to lowercase to treat them equally for the classification. However, lowercase can be at the same time harmful for the performance if the interpretation changes with converting. The word 'US", for example, which refers to the United States of America is equal to the pronoun "us" if converted to lowercase. This can affect the performance if the differentiations play a role in the classification (Kowsari et al., 2019). Conversion makes sense for occupational names in German since capitalization plays a major role in German. In turn, I do not expect the conversion to cause problems often, as described in the example above, since occupational titles usually have specific names, and their lowercase variants do not often lead to other words or word classes. Thus, in a second step, titles are all converted to lowercase.

As in every text classification task, stop word removal is a common praxis. All stop words, which are listed in the German stop words list of the Nature Language Toolkit package (Bird et al., 2009), are removed. In addition, the data contained other peculiarities that justify the removal of other words besides stop words. Job titles often contain words, such as "employee" or "effective immediately", that do not contain important information and are not specific to particular job titles. In order to identify such words, the frequencies of each word across all documents are calculated. These frequencies are then used to identify words, such as "employee", that are common but not relevant. As a final preprocessing step, the identified words are deleted from all job titles.

4.3 Vectorization Techniques

Count Vectorizer

The count vectorizer is one of the simplest methods for converting texts into vectors. It belongs to the family of BOW models. BOW models are based on vectors, where each dimension is a word from a vocabulary or corpus. The corpus is built by all words across all documents. BOW models have two essential properties. Firstly, they do not consider the order of the words, sequences, or grammar, which is why they are also called BOW. Secondly, each word in the corpus is represented by its own dimension. Thus, the vectors contain many zeros, especially for short texts, which is why they belong to the family of sparse vectors (Ajose-Ismail et al., 2020). Assuming that a corpus contains 1000 words, meaning each text is built only with these 1000 words, the vector for each text has a length of 1000, thus producing sparse, high-dimensional vectors (Kulkarni and Shivananda, 2021; Sarkar, 2016)

The values for the vector are generated by counting for each text the frequency of the words occurring. Considering a corpus including only the three words “java”, “developer” and “python”, the titles “java developer” and “python developer” would be encoded as follows:

	java	developer	python
java developer	1	1	0
python developer	0	1	1

Table 4: Encoding with count vectorizer

As can be seen 4 results in the vectors $(1, 1, 0)$ and $(0, 1, 1)$. Note that if the title “java developer” contains, for example, two times the word “java”, then the vector would change to (2,1,0). But since this is not a likely case for short text and especially job titles, the count vectorization here is for the most titles similar to one-hot vector encoding, which only considers the occurrence of the words, but not the frequency (Kulkarni and Shivananda, 2021; Sarkar, 2016)

While it is one of the most simple techniques, the count vectorizer has several limitations. The main downside is that it does not consider information like the semantic meaning of a text, the order, sequences, or the context. In other words, much information of the text is lost (Sarkar, 2016). In addition, the count vectorizer does not consider the importance of words in terms of a higher weighting of rare words and a lower weighting of frequent words across all documents (Suleymanov et al., 2019).

TF-IDF Vectorizer

Like the count vectorizer TF-IDF belongs to the family of BOW and is a sparse vector as well. In contrast to the count vectorizer, it considers the importance of the words by using the Inverse Document Frequency (IDF). The main idea of TF-IDF is to produce high values for words that often occur in documents but are rare over all documents. The Term Frequency (TF) represents the frequency of a word t in a document d and is denoted by $tf(t,d)$. The Document Frequency (DF), denoted by df, quantifies the occurrence of a word over all documents. By taking the inverse of DF, we get the IDF. Intuitively, the IDF should quantify how distinguishable a term is. If a term is frequent over all documents, it does not help distinguish between documents. Thus, the IDF produces low values for common words and high values for rare terms and is calculated as follows (Sidorov, 2019; Kuang and Xu, 2010):

$$idf(t) = log\frac{N}{df}$$

where N is the set of documents. The log is used to attenuate high frequencies. If a term occurs in every document, so $df = N$ the IDF takes a value of 0 ($log(\frac{N}{N})$) and if a term is occuring only once in all existing documents, thus distinguishing perfectly from other documents, IDF becomes 1 ($log(\frac{N}{1})$ (Sidorov, 2019). Note that the calculation of IDFs can be slightly adjusted (Robertson, 2004). The implementation of sklearn package, which is used in this work, uses an adapted calculation (Pedregosa et al., 2011):

$$idf(t) = log\frac{1+n}{1+df(t)} + 1$$

Given the $idf(t)$ and tf the TF-IDF can be obtained by multiplying both metrics. In addition sklearn normalizes the resulting TF-IDF vectors v by the Euclidean norm (Pedregosa et al., 2011):

$$v_{norm} = \frac{v}{||v||_2}$$

Although TF-IDF considers the importance of words, as a BOW model it suffers from the same limitation as the count vectorizer does by not taking semantic, grammatic, sequences, and context into account (Sarkar, 2016).

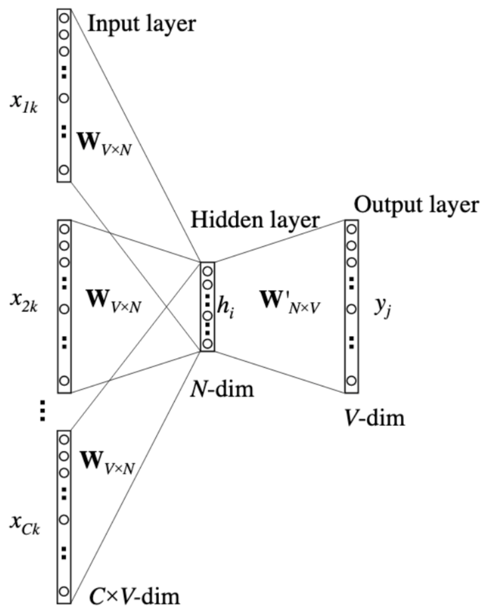


Figure 3: CBOW (Rong, 2014, 6)

Word2vec

In contrast to the sparse techniques mentioned above, word embedding techniques are another popular approach for vectorization. Word embedding vectors are characterized by low dimensions, dense representation, and continuous space. They are usually trained with neural networks (Li et al., 2015; Jin et al., 2016).

Word2vec, introduced by Mikolov et al. (2013), is one computationally efficient word embedding implementation. The main idea of word2vec is based on the distributional hypothesis, which states that similar words often appear in similar context (Sahlgren, 2008). Thus, word2vec learns with the help of the context representations of words, which include the semantic meaning and the context. In such a way, similar words are encoded similarly (Sarkar, 2016).

There exist two variants of word2vec. The first variant is based on BOW, the so-called Continous Bag of Words (CBOW), which predicts a word based on surrounding words. In contrast, the second variant, Skip-Gram, predicts the context words from a word (Ajose-Ismail et al., 2020; Sarkar, 2016). This study uses a pre-trained model from Google, which is trained with CBOW. Thus, in the following, the focus is on CBOW.

CBOW word2vec is a 2-layer neural network with a hidden layer and an output softmax layer, which is visualized in figure 3. The goal is to predict a word, the so-called target word, by using the target word's context words. A certain window size defines the number of context words. If the window size is 2, the two words before and after the target word are considered. Given a vocabulary V, which is the unique set of words from the corpus, each context word c is fed into the neural network, encoded with a one-hot encoding of the length of V, building vector x_c. Thus, in figure 3 x_{1k}, for example, could be a one-hot encoded vector of the word before the target word. The weights between the input layer and the hidden layer are shown in figure 3. Taking the dimension of V and the hidden layer size N results in the $V \times N$ matrix W. Given that each row v_w in W represents the weights of the associated word from the input and C equals the number of context words, the hidden layer h is calculated as follows (Rong, 2014):

$$h = \frac{1}{C} W^T (x_1 + x_2 + ... + x_c) = \frac{1}{C} (v_{w_1} + v_{w_2} + ... + v_{w_C})$$

Since the context words are encoded with one-hot vector encoding, all except the respective word value in the vector, which is 1, will be 0. Thus, calculating h is just copying the k-th row of matrix W, thus an n-dimensional vector, which explains the second part of the equation. The hidden-layer matrix later builds the word embedding, which is why the decision on the size of the hidden layer later defines the dimensions for the word embedding vector (Rong, 2014)

From the hidden to the output layer, a $N \times V$ weight matrix is used to calculate scores for each word in V. A softmax function is used to get the word representation. Calculating the error and using backpropagation, the weights are updated respectively, resulting in a trained neural network. Due to computational efficiency, word2vec is trained with hierarchical softmax or negative sampling instead of the softmax function. Both methods are efficient because they reduce the amount of weight matrix updates (Rong, 2014; Simonton and Alaghband, 2017).[5]

Based on the given theoretical insights, I trained two word2vec models. Both models use a pre-trained model from Google and are fine-tuned with different data. The first model is fine-tuned with the complete dataset. The second model includes additional knowledge. The set-up is as follows: I use CBOW word2vec models with a negative sampling technique. The hidden layer size and thus the word embedding vectors is 300, since the vectors have to have the same size as the pre-trained Google vectors. The minimal count of words is set to 1. The number of times the training dataset is iterated, the epoch number, is set to 10. Lastly, the window size for the context is set to 5.

[5]Since the focus is relying here on the word embeddings from the hidden layer and not the trained neural network itself, no further mathematical details will be given concerning the updating. For a detailed derivation see (Rong, 2014)

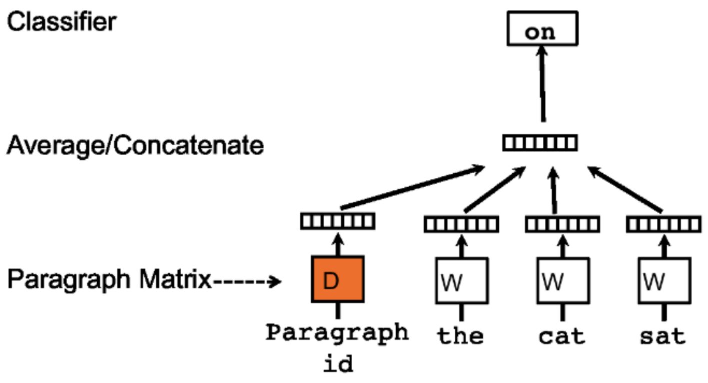


Figure 4: Doc2vec - Distributed Memory Model of Paragraph Vectors (Le and Mikolov, 2014, 3)

As the last step, the resulting word embeddings need to be processed in some way to get sentence embeddings for each job title. Word2vec cannot output sentence embeddings directly, which is why the word vector embeddings of each job title are averaged.

Doc2vec

Doc2vec, also known as paragraph vectors or Distributed Memory Model of Paragraph Vectors, is an extension method of word2vec, which outputs embeddings directly for each document (Lau and Baldwin, 2016). It was proposed by Le and Mikolov (2014). Doc2vec can be used for a variable length of paragraphs. Thus, it is applicable for more extensive documents, but also for short sentences like job titles (Le and Mikolov, 2014).

The main idea is, like for word2vec, to predict words in a paragraph. To do so a paragraph vector of that paragraph and word vectors are concatenated. The paragraph vector "acts as memory that remembers what is missing from the current contest - or the topic of the paragraph" (Le and Mikolov, 2014, 3). Thereby, the paragraph vectors are trained with stochastic gradient descent and backpropagation. Similar to word2vec's practical implementation, doc2vec uses hierarchical softmax or negative sampling to speed up the training time (Lau and Baldwin, 2016).

In figure 4 the algorithm is visualized. The neural networks take word vectors and a paragraph vector as input. While the word vectors are shared over all paragraphs, each paragraph's paragraph vectors are unique. The paragraphs are represented as unique vectors in a column of a matrix D. The word vectors are expressed in the matrix W as before. In order to predict the word, both vectors are combined, for example, by averaging or concatenating. The doc2vec implementation described by Le and Mikolov (2014) and also used in this paper here concatenates the vectors. Formally, this only changes the calculation of h (Lau and Baldwin, 2016).

Since doc2vec is a word embedding method, it has the same advantages mentioned for word2vec. In addition, doc2vec takes the word order into account - at least in the same way a large n-gram would (Le and Mikolov, 2014). Besides the model explained above, doc2vec also comes in a second variant, the so-called Distributed Bag of Words of Paragraph model, which ignores the word order. It is not clear which model performs better, although the inventor of doc2vec propose the first version (Lau and Baldwin, 2016).

Based on this discussion, I created two Distributed Memory Models of Paragraph Vectors. Since there are no popular pre-trained models for doc2vec, I trained two custom models, one with the training data and one including the additional knowledge. I set the vector size to 300 with a window size of 5, a minimal count of 1, and trained ten epochs. Like word2vec, the models are trained with negative sampling.

BERT

BERT uses a Multi-Layer Bidirectional Transformer Encoder as the architecture. This transformer architecture was introduced by Vaswani et al. (2017). It consists of an encoder and a decoder and makes use of self-attention. Both encoder and decoder stack include an identical layer. In addition, each of them includes two sub-layers: A multi-head attention and a feedforward network layer.[6] It is out of the scope of this paper to elaborate on the technical details of the attention mechanism, which is why a simplified explanation is given.

The self-attention mechanism improves the representation of a word, represented by a matrix X by relating it to all other words in the sentence. In the sentence "A dog ate the food because it was hungry" (Ravichandiran, 2021, 10) the self-attention mechanism, for example, could identify that "it" belongs to "dog" and not to "food" by relating the word to all other words. In order to compute the self-attention of a word, three additional matrices, the query matrix Q, the key matrix K, and a value matrix V are introduced. Those matrices are created by introducing weights for each of them and multiplying those weights with X. Based on those matrices, the dot product between the Q and the K matrix, a normalization and a softmax function is applied in order to calculate an attention matrix

[6] A simplified visualization with a two-layer encoder, as well the architecture of a can be found in the Appendix.

Z.[7] BERT uses a multi-head-attention, which means that multiple Z attention matrices instead of a single one are used.

BERT takes one sentence or a pair of sentences as input. To do so, it uses a WordPiece tokenizer and three special embedding layers, the token, the segment, and the position embedding layer for each token, which is visualized in 5. A WordPiece tokenizes each word that exists in the vocabulary. If a word does not exist, words are split as long as a subword matches the vocabulary or an individual character is reached. Hashes indicate subwords. For the token embeddings, the sentence is tokenized first, then a [CLS] is added at the beginning of the sentence and a [SEP] token at the end. The job titles “java developer” and “python developer”, for example, become tokens as shown in the second row of 5. In order to distinguish between the two titles, a segment embedding is added, indicating the sentences. Lastly, the BERT model takes a position embedding layer, which indicates the order of the words. For each token, those layers are summed up to get the representation for each token (Devlin et al., 2018; Ravichandiran, 2021).

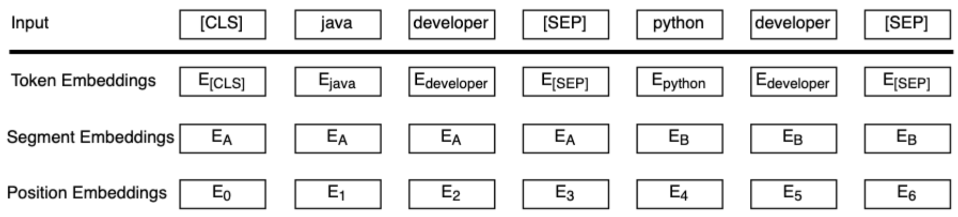


Figure 5: Input of BERT (Devlin et al., 2018, 5)

The BERT algorithm can be described in two steps. First, the pretraining phase, which is illustrated on the left-hand side of figure 6 and the fine-tuning phase, which is visualized on the right-hand side.

Pretraining The pretraining phase consists of two jobs: Masked language modeling and next sentence prediction. Masked language modeling means that a percentage of the input tokens are masked at random. For example, the job title “python developer” could be masked as follows: [[CLS] python [MASK] [SEP]]. Since in fine-tuning tokens are not masked, a mismatch would occur between fine-tuning and pretraining, which is why not all of the masked tokens are matched with a [mask] token, but also with a random token or

[7]For a step-by-step calculating see (Ravichandiran, 2021)

the real tokens.[8] Instead of predicting the complete sentence, BERT trains to predict the masked tokens. The prediction is performed with a feed-forward network and a softmax activation (Devlin et al., 2018; Ravichandiran, 2021).

The second task retakes two sentences but predicts whether the second sentence follows the first one, which helps understand the relationship between the sentences. Each sentence pair is labeled with either isNext or NotNext. By using the [CLS] token, which has the aggregating representation of all tokens, a classification task of whether a sentence pair is isNext or NotNext can be carried out (Ravichandiran, 2021; Devlin et al., 2018).

The pretraining of BERT is, in contrast to the fine-tuning process, computationally expensive. There are plenty of pre-trained BERT models for the German case, like "bert-base-german-cased" or "distilbert-base-german-cased". [9] In an evaluation of German pre-trained language models, Aßenmacher et al. (2021) conclude that the "bert-base-german-dbmd-uncased" algorithm works quite well. Following their results and own tests on different models, "bert-base-german-dbmd-uncased" and "bert-base-german-cased" seem to have the best results, which is why they are used for the fine-tuning process. Both models consist of 12 encoder layers, 12 attention heads, and 768 hidden units, resulting in 110 million parameters. The first model was trained with 16GB of German texts, the second model with 12GB of German texts.

Fine-Tuning The second phase, fine-tuning, can be performed differently, depending on the task. Either the weights of the pre-trained model are updated during the classification

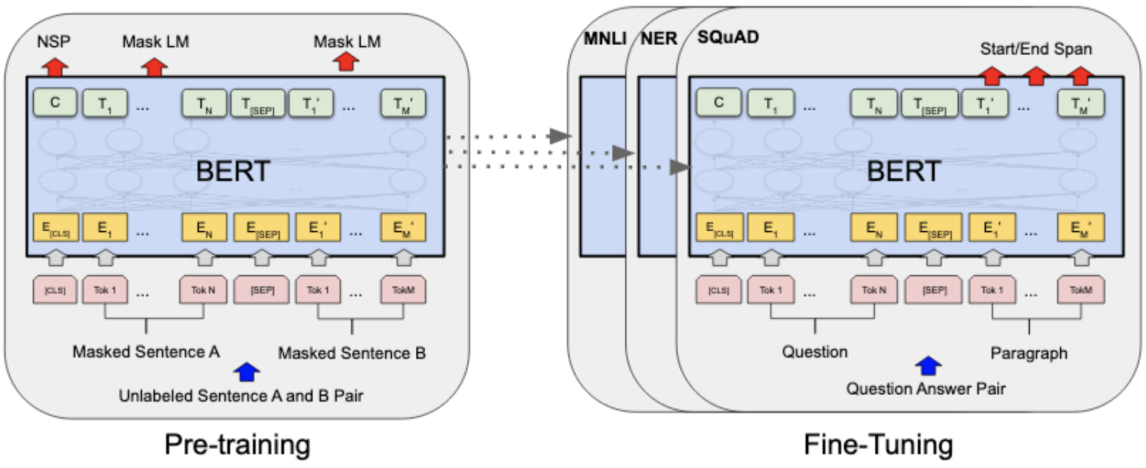


Figure 6: Overview BERT (Devlin et al., 2018, 3)

[8] There are specific rules of how to mask. See Devlin et al. (2018) for detailed implementation

[9] All german BERT models are open source and are accessible through the transformers library (Wolf et al., 2020)

process, or the pre-trained model is first fine-tuned and then used as a feature extractor.

I train two models with BERT. While the first model includes a classification layer, in the following named BERT deep learning model, the second model applies BERT as a feature extraction method, in the following named BERT vectorizer.

The BERT deep learning model is fine-tuned with the complete training dataset. Practically, this is done by converting the sentences of the dataset into the appropriate data format as described above and training it with the supervised dataset on some epochs, which then outputs the labels. From a theoretical point of view, the last hidden state of the [CLS] token with the aggregation of the whole sentence is used for the classification. In order to get the labels, BERT uses a softmax function (Sun et al., 2019).[10] As already stated in the literature, it is not well-understood what exactly happens during the fine-tuning. An analysis of Merchant et al. (2020) for other NLP tasks than text classification indicates that the fine-tuning process is relatively conservative in the sense that they affect only a few layers and are specific for examples of a domain. The training set-up is as follows: Testing different epoch numbers indicates that smaller epoch size has better results for the model, which is why I fine-tune in 6 epochs. For the optimization, an adam algorithm, a gradient-based optimizer (Kingma and Ba, 2014), with a learning rate of $1e^{-5}$ is used.

In order to get sentence embeddings, different strategies, like averaging the output layer of BERT or using the [CLS] token, are applied. Another method, developed by Reimers and Gurevych (2019) is sentence-BERT, which is computationally efficient and practicable to implement. Since it facilitates encoding sentences directly into embeddings, I use it for the BERT vectorizer. The model is constructed with the "bert-base-german-dbmd-uncased“ model and a pooling layer. The pooling layer is added to output the sentence embeddings. The fine-tuning process uses a siamese network architecture to update the weights. Figure 7 shows the architecture. The network takes pairs of sentences with a score as an input. Higher scores indicate higher similarity between the sentences. The network updates the weights by passing the sentences through the network, calculating the cosine similarity, and comparing to the similarity score (Reimers and Gurevych, 2019).

From the job title dataset and the *kldb* dataset, I create pairs of similar and dissimilar job titles or search words. Similar pairs are pairs by being the same *kldb* class. As a score, I choose 0.8. Dissimilar pairs are defined as pairs that are not from the same class. The score is 0.2. Building all combinations of titles and the search words for each class

[10]The explanation of the softmax function follows in the chapter of the classifiers

would result in a huge dataset. For example, class 1 of the training dataset has 2755 job titles, which results in $\binom{2755}{2} = 3793635$ pairs for class 1. Since this is computationally too expensive, I randomly choose pairs of job titles. For level 1, the following samples are drawn: From the job title data for each class, I used 3250 pairs. The same is true for the searchwords of each class. For unsimilar pairs, I used 1750 job title pairs, not from the same class. The same principle applies for the search words. This results in a total number of $(3250 + 3250) \times 9 + 2 \times 1750 = 62000$ pairs for the finetuning of level 1.[11] For level 3, the procedure is the same, only the numbers differ. For similar pairs of job titles and searchwords I used 400 for each class and for the unsimilar I used 6000 pairs. Classes with only one example are not considered because building pairs is impossible. This gives a total number of $(400 + 400) \times (136 - 7) + 2 \times 1500 = 61200$ pairs.

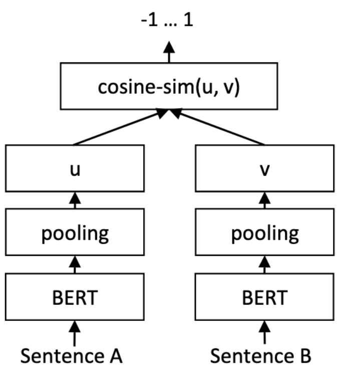


Figure 7: Sentence-BERT siamese architecture (Reimers and Gurevych, 2019, 3)

4.4 Dimensionality Reduction

Dimensionality reduction techniques based for example on Principal Component Analysis (PCA) play an essential role in reducing computation time and saving computing resources (Ayesha et al., 2020). Figure 8 shows the runtime of all three classifiers with different data sizes and both with and without PCA.

The input for the classifiers are the word2vec word embeddings without the additional information. The bright lines show the runtimes without dimensionality reduction, while the dark-colored lines report the runtime with PCA transformation. It becomes clear that the runtime for the transformed embeddings is generally lower. The magnitude of the differences is almost irrelevant for a dataset of 500. However, for the non-transformed

[11]The determination of the exact number of pairs is exploratory, and run time and performance were taken into account.

embeddings, the runtime increases considerably with the dataset size for all classifiers. This is most evident with RF. With increasing dataset size the runtime of the transformed embeddings also increases for all classifiers. However, it does so at a much slower pace. Therefore, it can be concluded that the transformation contributes to keeping the runtime lower for large datasets. As already described in chapter 3, the training dataset is pretty large, so it is reasonable to reduce the dimensions.

PCA, one of the most popular techniques for dimensionality reduction, aims to reduce a high-dimensional feature space to a lower subspace while capturing the essential information (Tipping and Bishop, 1999; Bisong, 2019). The main idea is to use linear combinations of the original dimensions, so-called principal components, to reduce the dimensional space (Bro and Smilde, 2014; Geladi and Linderholm, 2020).

Conceptually, the covariance matrix for the word embeddings is obtained in the first step. The covariance matrix, denoted by **X**, captures the linear relationships between the features of the word embeddings. In the next step, the eigenvectors of **X** are calculated. The eigenvector of X is defined as (Bro and Smilde, 2014):

$$\mathbf{X}z = \lambda z$$

where z is the eigenvector and λ the eigenvalue. In order to decompose **X** to get the eigenvalue, Singular Value Decomposition is applied. The eigenvalues are sorted from highest to lowest, and the most significant components n are kept. In order to choose n, the variance explained by the principal components is considered. The explained variance is set to 0.95. In order to transform the data, a feature vector is generated. This vector contains the most n significant eigenvalues. After transposing the mean-adjusted word embedding and the feature vector, the embeddings can be transformed by multiplying

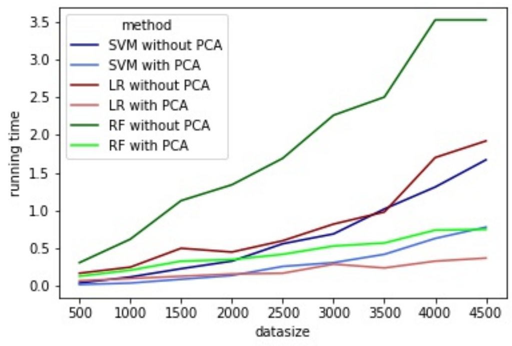


Figure 8: Running time of word2vec with different data sizes

both transposed vectors (Smith, 2002).

4.5 Classifiers

In the following, based on a theoretical discussion of each classifier, the exact modeling is justified. The focus and depth of the explanations of the properties of the classifiers, such as decision function or regularization, depend on the settings chosen for the classification algorithms. Therefore, not all properties are described to the same extent for all classifiers. The BERT deep learning model is already described in chapter 4.3.

4.5.1 Logistic Regression

Multinomial Logistic Regression (MLR), a generalized linear model, is one of the most used analytical tools in social and natural science for exploring the relationships between features and categorical outcomes. For solving classification problems, it learns weights and a bias (intercept) from the input vector. Figure 9 illustrates the idea of the calculation of MLR. To classify examples, first, the weighted sum of the input vector is calculated. For multi-classification, the weighted sum has to be calculated for each class. Thus, given a $f \times 1$ feature vector x with $[x_1, x_2, ..., x_f]$, a weight vector w_k with k indicating the class k of the set of classes K, a bias vector b_k, the weighted sum is defined as the dot product of w_k and **x** plus the b_k. Representing the weight vectors of each class in a $[K \times f]$ matrix **W**, the weighted sum can be formally expressed as $\mathbf{W}x + b$. In Figure 9, the blue lines, for example, are a row in **W** and are the weight vectors related to a class labeled with 1 (Jurafsky and Martin, 2021).

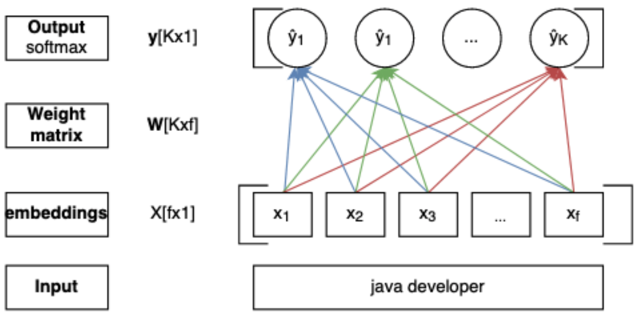


Figure 9: MRL (edit after (Jurafsky and Martin, 2021, p.))

In a second step, the weighted sums are mapped to a value range of $[0, 1]$ to classify the input. While binary logistic regression uses a sigmoid function to do so, MLR needs a generalized sigmoid function. This generalization is called the softmax function, which outputs probabilities for each of the classes, which is why MLR is often called softmax regression in the literature. These probabilities models for each class $p(y_k = 1|x)$.

Similar to the sigmoid function, but applicable for multiple values, the softmax function maps each value of an input vector z with $[z_1, z_2, ..., z_K]$ to a value of the range of $[0, 1]$. Thus, outputting a vector of length z and all values together summing up to 1. Formally it is defined as:

$$softmax(z_i) = \frac{exp(z_i)}{\sum_{j=1}^{K} exp(z_j)} \; 1 \leq i \leq K$$

Then the output vector y can be calculated by

$$\hat{y} = softmax(\mathbf{W}x + b)$$

The goal of weight and bias training is to "maximize the log probability of the true y labels" of the input data. This is commonly done by minimizing a generalized cross-entropy loss function for MLR. Different methods exist for solving the optimization problem, like stochastic gradient descent or limited-memory Broyden-Fletcher-Goldfarb Shannon solver. The latter converges rapidly and is characterized by a moderate memory, which is why it can converge faster for high-dimensional data (Fei et al., 2014; Pedregosa et al., 2011).

For MLR it is common to add a regularization parameter. This avoids overfitting and ensures that the model is more generalizable to unseen data. The idea is to penalize weights with good classification but using high weights more than weights with good classification but smaller weights. There are two popular penalty terms, the L1 and the L2 penalty. While for L1, the absolute values of the weights are summed and used as the penality term, L2 regularizes with a quadratic function of the weights. With the regularization, a parameter C is introduced to control the strength of the regularization. C is a positive value, and if smaller, it regularizes stronger (Jurafsky and Martin, 2021).

The following setting will be used for the MLR: A MLR with a L2 penalty with a C value of 1 is used. Since the input vectors, especially for count vectorizer and for TF-IDF are high-dimensional, the limited-memory Broyden-Fletcher-Goldfarb-Shannon solver is set for solving. For some training, converge problems appeared, which is why the maximal iteration of the classifier is set to 10000.

4.5.2 Support Vector Machines

The general idea of a SVM is to map "the input vectors x into a high-dimensional feature space Z through some nonlinear mapping chosen a priori [...], where an optimal separating hyperplane is constructed" (Vapnik, 1999, 138). This optimal hyperplane maximizes the margin, which is the distance from the hyperplane to the closest points, so-called Support Vectors, across both classes (Han et al., 2011). Formally, given a training dataset with n training vectors $x_i \in R^n, i = 1, ..., n$ and the target classes $y_1, ...y_i$ with $y_i \in \{-1, 1\}$, the following quadratic programming primal problem has to be solved in order to find the optimal hyperplane:

$$\min_{w,b} \frac{1}{2} w^T w$$

$$\text{subject to } y_i(w^T \phi(x_i) + b) \geq 1$$

where $\phi(x_i)$ transforms x_i into a higher dimensional space, w corresponds to the weight and b is the bias (Chang and Lin, 2001; Jordan et al., 2006). The given optimization function assumes that the data can be separated without errors. Since this is not always possible, Cortes and Vapnik (1995) has introduced a soft margin SVM, which allows for misclassification (Vapnik, 1999). By adding a regularization parameter C with $C > 0$ and the corresponding slack-variable ξ the optimization problem changes to (Chang and Lin, 2001; Han et al., 2011):

$$\min_{w,b} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \xi_i$$

$$\text{subject to } y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i,$$

$$\xi_i \geq, i = 1, ..., n$$

Introducing Lagrange multipliers α_i and converting the above optimization problem into a dual problem, the optimal w meets (Chang and Lin, 2001; Jordan et al., 2006):

$$w = \sum_{I=1}^{n} y_i \alpha_i \phi(x_i)$$

with the decision function (Chang and Lin, 2001):

$$\text{sgn } (w^T \phi(x) + b) = sgn(\sum_{i=1}^{n} y_i \alpha K(x_i, x) + b)$$

$K(x_i, x)$ corresponds to a Kernel function, which allows calculating the dot product in the original input space without knowing the exact mapping into the higher space (Han et al., 2011; Jordan et al., 2006).

In order to apply SVM to multi-class problems, several approaches have been proposed. One strategy is to divide the multi-classification problem into several binary problems. A common approach here is the one-against-all method. In this method, as many SVM classifiers are constructed as there are classes k. The k-th classifier assumes that the examples with the k label are positive labels, while all the other examples are treated as negative. Another popular approach is the one-against-one method. In this approach $k(k-1)/2$ classifiers are constructed, allowing to train the data of two classes within each classifier (Hsu and Lin, 2002). Besides, some researchers, like Crammer and Singer (2001), propose approaches that seek to solve the task in a single optimization problem.[12].

In order to find a robust classifier, I checked SVMs with different parameters, as well as different multi-class approaches. I found that a SVM using a soft margin with a $C = 1$ and a one-vs-rest approach yielded the best results. I also tested different kernels, like RBF Kernel or linear kernel. The latter, formally $k(x, x') = x^T x'$, achieved the best results, which is why I chose it for the classifier.

4.5.3 Random Forest Classifier

In contrast to the previous two classifiers, RF is an ensemble learning technique. The main idea of ensemble learning techniques is to create several learners and combine them. Those learners are, for example, a Decision Tree or neural networks and are usually homogeneous, meaning that each learner is based on the same machine learning algorithm. The different ensemble techniques are built on three pillars: the data sampling technique, the strategy of the training, and the combination method (Polikar, 2012; Zhou, 2009).

The first pillar, data sampling, is important because it is not desirable to have the same outputs for all classifiers. Thus, ensemble techniques require diversity in the output, i.e. the outputs should be optimally independent and negatively correlated. Several methods for achieving diversity are already well established. Bagging techniques, for example, rely on bootstrap. (Polikar, 2012). The second pillar raises the question of which techniques should be applied to train the learners of the method. The most popular strategies for the training are bagging and boosting (Polikar, 2012). The last pillar is about the combining method. Each classifier of the method will output an individual classification result, and

[12]For a detailed overview of all different methods and the method of Crammer and Singer (2001) see Hsu and Lin (2002); Crammer and Singer (2001)

those results have to be combined in some way to achieve an overall result. There are plenty of methods like majority voting or borda count (Polikar, 2012).

RF uses Decision Trees as its individual classifier. Before discussing RF in more detail within the three pillars described above, a brief discussion of the Decision Tree is given in order to understand the mechanism and training procedure of the classifiers.

The main idea of the Decision Tree algorithm is to "break up a complex decision into a union of several simple decisions" (Safavian and Landgrebe, 1991, 660) by using trees, with a root node on the top, intermediate nodes, and leaf nodes at the bottom. All possible splittings are checked and split according to the best feature for the root node and each intermediate node. Each leaf node leads to one of the classification labels. Examples are then classified by traversing the tree from top to bottom and choosing the branch that satisfies the attribute value for the example at each intermediate node. The construction of a Decision Tree is a recursive procedure (Berthold et al., 2020; Xia et al., 2008; Cutler et al., 2012). The algorithm stops for a specific node if all training set examples belong to the same class or if there are no features left for splitting. This might end in a tree with a high depth, which is why pruning is often applied to avoid overfitting (Berthold et al., 2020).

Regarding the construction of the tree, the type of splitting and the splitting criterion has to be discussed. There are mainly three types of splits: boolean splits, nominal splits, and continuous splits. The latter chooses a particular value from the continuous feature as the splitting value (Cutler et al., 2012; Berthold et al., 2020). For example, considering a word embedding x with 300 dimensions and a node t of a Decision Tree, which is split into the nodes t_{left} and t_{right}. The node t could have the split $x[209] <= 0.336$. Examples with a value smaller than or equal to 0.336 at the dimension index 209 of the embedding vector follow the branch to t_{left}, while all other examples follow the branch to t_{right}.

The splitting criterion is essential to identify the best feature for splitting. Intuitively, the criteria should split the data in such a way that leaf nodes are created quickly (Berthold et al., 2020). Several measurements can be applied to determine the best split for each node, like Gini Impurity or Information Gain. Since RF uses Gini Impurity, only this criterion will be discussed in detail below.

The gini value indicates the purity of a dataset D with n classes. It is defined as follows (Yuan et al., 2021, 3156):

$$Gini(D) = 1 - \sum_{i=1}^{n} p_i^2$$

where p_i is the probability that a class n occurs in D. The more pure D is, the lower the value of the Gini value. In order to determine the best feature k, the dataset is partitioned based on feature k. For continuous features, as in word embeddings, this is done by a continuous split. Defining V as the total number of subsets and D^v as one of the subsets, the Gini impurity for a feature k can be calculated as follows Yuan et al. (2021):

$$\text{Gini index}(D, k) = \sum_{v=1}^{V} \frac{|D^v|}{|D|} Gini(D^v)$$

Conceptually the Gini Index is the weighted average of the Gini value for each subset of D based on a feature k. Thus, subsets with more samples are weighted more in the Gini index. The optimal feature k^* is then determined by minimizing the Gini impurity overall features K (Yuan et al., 2021, 3156):

$$k^* = arg \min_{k \in K} \text{Gini index}(D, k)$$

Based on the above theoretical explanations of the foundations of the Decision Tree, researchers have developed several algorithms to train Decision Trees, like Iterative Dichotomiser 3, C4.5, or Classification and Regression Trees (CART). Depending on the target CART produces classification (for categorical variables) or regression trees (for numerical variables). It only construct binary trees. The algorithm uses Gini Index, and it can handle numerical and categorical input (Brijain et al., 2014).

RF belongs to the family of bagging ensemble techniques. Bagging selects a single algorithm and trains several independent classifiers. The sampling technique is sampling with replacement (bootstrapping). Bagging combines the individual models by using either majority voting or averaging. RF differs from the classic bagging method by also allowing the selection of a subset of features for each classifier from which the classifiers can select, rather than allowing them to select from the entire range of features. (Polikar, 2012; Zhou, 2009; Berthold et al., 2020).

Formally Breiman (2001), who introduced mainly the RF algorithm, defines the classifier as follows:

> “A random forest is a classifier consisting of a collection of tree-structured classifiers $\{h(\mathbf{x}, \Theta_k), k = 1, ...\}$ where the Θ_k are independent identically dis-

tributed random vectors and each tree casts a unit vote for the most popular class at the input $\mathbf{x}$." (Breiman, 2001, 6).

Θ_k is a random vector created for each k-th tree. Θ_k must be independent of the vectors $\Theta_1 ... \Theta_{k-1}$ and thus of all random vectors of the previous classifiers. Nonetheless, the distribution of the random vectors remains. Combined with the training set, with $\mathbf{x}$ as the input vector a classifier $h(\mathbf{x}, \Theta_k)$ is constructed (Breiman, 2001). The random component Θ_k is not explicitly used in practical implementation. Instead, it is rather used implicitly to generate two random strategies (Cutler et al., 2012). The first strategy is bootstrapping, i.e. drawing a sample with replacement from the training dataset. In order to estimate the generalization error, the correlation and the variable importance, Breiman (2001) applied out-of-bag estimation. Out-of-bag estimation leaves out some portion of the training data in each bootstrap. The second strategy is to choose a random feature for the splitting. Thus, at each node from the set of features, only a subset is used to split. RF does not use pruning. The trees grow by applying CART-algorithm. As combination method for classification RF uses unweighted voting.(Cutler et al., 2012).

Based on the above explanations, the implemented RF has the following setting: The number of learners is 100. Gini is used as the splitting criterion. The maximal number of features is square root of the number of features. Note that sklearn, which is used to implement RF here, uses an optimized algorithm of CART (Pedregosa et al., 2011).

5 Result

5.1 Evaluation Metrics

Appropriate measurements are crucial for being able to meaningfully compare the performance of different algorithms. There exist several metrics for the evaluation of classification approaches in the literature (Fatourechi et al., 2008). In the following, the selected measurements for performance measurement will be justified.

Most metrics rely on a confusion matrix. For the multi-class case, this confusion matrix is defined as follows (Kautz et al., 2017):

Derived from the confusion matrix one can see that $c_{i,j}$ defines examples which belong to *class j* and are predicted as *class i*. Based on the confusion matrix, the true positives of a current class m can be defined as $tp_m = c_{m,m}$, i.e. examples correctly predicted as the current *class m*. The false negatives are defined as those examples which do not belong to the current *class m*, but are predicted as k, formally written as $fn_m = \sum_{i=1,i\neq m}^{n} c_{i,m}$. Next, the true negatives are examples belonging to the current *class m*, but are not predicted as m. This can be represented through the formula $tn_m = \sum_{i=1,i\neq m}^{n} \sum_{j=1,j\neq m}^{n} c_{i,j}$. Last, false positives are defined as examples not belonging to *class m*, but are predicted as such. Formally this can be expressed as: $fp_m = \sum_{i=1,i\neq m}^{n} c_{m,i}$ (Kautz et al., 2017)

The Overall Accuracy (OA) is one of the most common metrics for performance evaluation. It represents how well the classifier classifies across all classes correctly. Given that N is the number of examples, formally, the OA can be expressed as:

$$OA = \frac{\sum_{i=1}^{m} tp_i}{N}$$

An accuracy of 1 means that all examples are correctly classified. 0 means that each example is classified with the wrong class (Berthold et al., 2020). Although OA is a widely used metric, it is criticized for favoring the majority classes and therefore not adequatley reflecting minority classes in unbalanced datasets (Berthold et al., 2020; Fatourechi et al., 2008).

Two more popular metrics are precision and recall. Precision represents how well the classifier detects actual positive examples among the positive predicted examples. Recall, also called sensitivity, in contrast, represents how many examples are labeled as positive among the actual positive examples (Berthold et al., 2020). For the multi-class scenario, two different calculation approaches for each of the metrics are proposed: micro and macro average (Branco et al., 2017). In the macro approach the metric for each *class m* is first calculated against all other classes. Then, the average of all of them is computed. Given that K is the total number of classes the macro precision and recall are defined as:

$$precision_{macro} = \frac{1}{M} \sum_{i=1}^{m} \frac{tp_i}{tp_i + fp_i}$$

	positive examples			
positive prediction	$c_{1,1}$	$c_{1,2}$	$\cdots$	$c_{1,n}$
	$c_{2,1}$	$c_{i,j}$		
	$\vdots$		$\ddots$	$\vdots$
	$c_{n,1}$		$\cdots$	$c_{n,n}$

Table 5: Confusion Matrix (adapted from (Kautz et al., 2017, 113))

$$recall_{macro} = \frac{1}{M}\sum_{i=1}^{m}\frac{tp_i}{tp_i + fn_i}$$

In contrast to the macro apporach, the micro approach aggregates the values, which can be formally expressed as follows:

$$precision_{micro} = \frac{\sum_{i=1}^{m} tp_i}{\sum_{i=1}^{m} tp_i + fp_i}$$

$$recall_{micro} = \frac{\sum_{i=1}^{m} tp_i}{\sum_{i=1}^{m} tp_i + fn_i}$$

There is a trade-off between precision and recall (Buckland and Gey, 1994). The F-score captures precision and recall by taking the harmonic mean between the two. It is calculated as follows (Branco et al., 2017; Narasimhan et al., 2016):

$$F_{micro} = 2 \cdot \frac{precision_{micro} \cdot recall_{micro}}{precision_{micro} + recall_{micro}}$$

$$F_{macro} = 2 \cdot \frac{precision_{macro} \cdot recall_{macro}}{precision_{macro} + recall_{macro}}$$

A closer look at the formula of the micro scores shows that actually, the micro-precision score and the recall score are the same since aggregating the false negatives and aggregating the false positive results in the same number. If the precision and recall are the same, it follows from the F-measure calculation that it has to be as well equal. And even further, the micro score is reducing to the accuracy, thus suffering from the same problem as accuracy (Grandini et al., 2020).

Since the job title classification involves multi-class classification and the descriptive analysis has shown that the data is imbalanced, it is not reasonable to base the evaluation solely on the OA. Showing that micro scores of precision, recall, and F1-scorereduce to accuracy, it is crucial to take the macro scores into account to capture the performance of the minority classes well.

5.2 Experimental Results

As explained above, the results are compared from three perspectives: Vectorization techniques, classification algorithms, and enrichment with additional knowledge. Table 6 and 7 show the results of level 1 for all vectorization methods. Each row denotes a vectorization technique. Each column represents a classifier. Table 6 reports the accuracy while table 7 reports the macro precision (p), recall(r) and F1-score (F1). Word2vec and doc2vec without additional knowledge training are marked with *I*, the ones with additional knowledge by *II*. The results of the BERT deep learning model are reported separately in table 8.

	LR	**SVM**	**RF**
count vectorizer	0.72	0.69	0.65
TF-IDF	0.72	0.69	0.65
word2vec_I	0.54	0.53	0.61
word2vec_II	0.54	0.52	0.62
doc2vec_I	0.48	0.46	0.56
doc2vec_II	0.45	0.42	0.53
BERT	0.78	0.78	0.77

Table 6: Evaluation of level 1 classification - accuracy

Comparing the accuracy of the vectorization techniques, with approx. 78% accuracy the BERT vectorizer outperforms the other methods. A look at the macro table 7 for BERT confirms the high performance with relatively similar macro scores over all classifiers. Only in combination with RF, the recall and thus the F1-score is lower with BERT compared to the other classifiers. Further, the two sparse techniques count vectorizer and TF-IDF performed quite well, especially compared to the word embedding techniques word2vec and doc2vec regardless of having additional knowledge or not for LR and SVM. Related to LR and SVM, the worst performance is achieved by doc2vec. However, word2vec also performs considerably below the sparse vectors and BERT. For example, word2vec_I has 18% less accuracy than the count vectorizer for LR. This picture is confirmed when looking at the macro table. The difference between word2vec and doc2vec on one side compared to the sparse vectors and the BERT vectorizer on the other side shows up more strongly. Again for LR, in comparison with word2vec_I, the count vectorizer performed 24% better, measured by the F1-score.

A different picture is given for RF. The differences between the sparse methods and

	LR	**SVM**	**RF**
count vectorizer	p: 0.76, r: 0.61, F1: 0.66	p: 0.72, r: 0.58, F1: 0.63	p: 0.66, r: 0.53, F1: 0.57
TF-IDF	p: 0.77, r: 0.61, F1: 0.65	p: 0.73, r: 0.58, F1: 0.62	p: 0.67, r: 0.53, F1: 0.57
word2vec_I	p: 0.58, r: 0.39, F1: 0.42	p: 0.46, r: 0.40, F1: 0.41	p: 0.58, r: 0.52, F1: 0.54
word2vec_II	p: 0.59, r: 0.41, F1: 0.45	p: 0.48, r: 0.41, F1: 0.43	p: 0.59, r: 0.53, F1: 0.55
doc2vec_I	p: 0.51, r: 0.33, F1: 0.35	p: 0.41, r: 0.33, F1: 0.34	p: 0.59, r: 0.40, F1: 0.43
Doc2vec_II	p: 0.54, r: 0.30, F1: 0.32	p: 0.37, r: 0.30, F1: 0.31	p: 0.55, r: 0.38, F1: 0.40
BERT	p: 0.76, r: 0.76, F1: 0.76	p: 0.75, r: 0.77, F1: 0.76	p: 0.78, r: 0.72, F1: 0.74

Table 7: Evaluation of level 1 classification - macro

	accuracy	**precision macro**	**recall macro**	**F1 macro**
BERT clf level 1	0.76	0.71	0.71	0.71
BERT clf level 3	0.44	0.20	0.18	0.17

Table 8: Evaluation of level 1 and 3 BERT deep learning model

the word2vec techniques is much smaller and for the macro scores almost vanished, while doc2vec performs lower. Finally, compared to BERT vectorizer, all other vectorizers are performing lower, taking the macro score as a performance metric instead of accuracy. Note that doc2vec has ill-defined scores for LR and RF and word2vec for SVM since they have no predictions for some classes. Considering the formula of the macro score above, if a class has zero examples, this class is set to zero but is included in the average. Hence, the macro scores for these vectorization techniques combined with the respective classifiers must be interpreted with caution.

The analysis of the classification algorithms confirms the vague impression in the literature. While LR and SVM have an excellent performance for the dense vectorization techniques, RF shows the best performance for the two word embedding techniques word2vec and doc2vec. For BERT, on the other hand, all classification algorithms turn out to be strong. The BERT deep learning model performed lower than the BERT vectorizer but better than the other vectorization techniques.

Concerning the last analysis dimension, the focus relies on word2vec_II and doc2vec_II, which contain additional knowledge for the embeddings.[13] The accuracy score in table 7 of word2vec_II shows no improvement by adding knowledge. For doc2vec, the results are even slightly worse, which is also reflected in the macro results. The macro results for word2vec show slightly better performance adding knowledge for SVM and LR, but for RF this improvement has almost disappeared.

Table 9 and 10 contain the results of level 3. Again the results of the BERT deep learning model are reported separately in 8. A problem is that many classes only have one example. Thus, there are often zero predictions for many classes' overall classifiers and vectorization methods. This leads to ill-defined macro scores. At the same time, the accuracy still suffers from the mentioned problems of favoring classes. Nevertheless, the performance of the classifiers can be compared, since all suffer more or less equally from the classes without predictions. However, not too much meaning should be given to the

[13]The BERT model contains as well additional knowledge since search words pairs also were used for the fine-tuning. However, the model comparison is difficult since BERT deep learning model was trained differently and allows no direct comparison

exact percentages.

	LR	**SVM**	**RF**
count vectorizer	0.50	0.52	0.44
TF-IDF	0.48	0.53	0.45
word2vec_I	0.30	0.15	0.36
word2vec_II	0.30	0.20	0.35
doc2vec_I	0.19	0.19	0.30
doc2vec_II	0.16	0.16	0.27
BERT	0.50	0.46	0.45

Table 9: Evaluation of level 3 classification - Accuracy

	LR	**SVM**	**RF**
count vectorizer	p: 0.41, r: 0.27, F1: 0.30	p: 0.40, r: 0.34, F1: 0.35	p: 0.36, r: 0.25, F1: 0.28
TF-IDF	p: 0.40, r: 0.23, F1: 0.27	p: 0.39, r: 0.33, F1: 0.35	p: 0.39, r: 0.26, F1: 0.29
word2vec_I	p: 0.18, r: 0.12, F1: 0.12	p: 0.08, r: 0.06, F1: 0.05	p: 0.24, r: 0.19, F1: 0.20
word2vec_II	p: 0.25, r: 0.14, F1: 0.17	p: 0.13, r: 0.11, F1: 0.10	p: 0.27, r: 0.20, F1: 0.22
doc2vec_I	p: 0.11, r: 0.05, F1: 0.05	p: 0.12, r: 0.09, F1: 0.09	p: 0.22, r: 0.13, F1: 0.14
doc2vec_II	p: 0.09, r: 0.04, F1: 0.04	p: 0.11, r: 0.07, F1: 0.08	p: 0.19, r: 0.11, F1: 0.12
BERT	p: 0.58, r: 0.51, F1: 0.53	p: 0.52, r: 0.48, F1: 0.48	p: 0.48, r: 0.33, F1: 0.36

Table 10: Evaluation of level 3 classification - macro

In general, the performance of level 3 is lower than level 1, which is due to the higher number of classes and lower number of examples in the classes. In contrast to level 1, there is no noticeable difference in accuracy between BERT and the sparse vectors. However, looking at the macro scores, BERT has a much higher recall, and thus a higher F1-score than the other techniques. In general, while the macro results for all other techniques reveal much worse performance compared to the accuracy score, BERT is stable overall metrics. Thus, again it can be concluded that BERT outperforms the other classifiers. Both sparse vectors performed relative equally. Comparing the sparse vectors against the word embeddings word2vec and doc2vec, the latter are the inferior. Same as for level 1, doc2vec has the worst performance.

Comparing the classifiers for level 3 shows the same picture for sparse vectors and word embeddings. Sparse vectors and BERT vectorizer perform better with LR and SVM, while the word embeddings have the best result with RF. In contrast to level 1, the best performance for BERT is achieved by LR on level 3, especially compared to the deep learning model, which performed much lower than the BERT vectorization and looking at the macro results even much worse than the sparse techniques. However, concerning the results of BERT of LR and SVM, a peculiarity should be noted. During training, there was no convergence of SVM with BERT in a real runtime. Therefore, a maximal iteration of 10000 was set. This might have affected the performance of SVM. Thus, one should be careful to rank LR as considerably better than SVM.

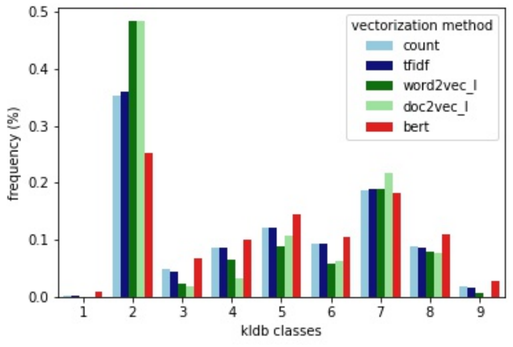
Including additional knowledge in word2vec reveals no noticeable improvement for LR and RF. For SVM, there seems to be a slight improvement. The trend of macro results across all classifiers seems to confirm the improvement, although, for RF, the improvement is not too great and should not be given too much attention due to the ill-defined scores. The same problem applies to doc2vec. Roughly compared, there is no difference between doc2vec_I and doc2vec_II. However, the somewhat poorer performance of doc2vec_II becomes apparent with some classification algorithms.

5.3 Deep Dive into the Results

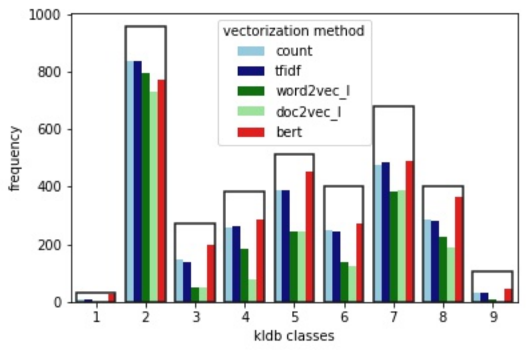
Looking at the results, we see some trends emerging. Comparing the vectorization techniques in both level 1 and level 3, the BERT vectorizer has the best performance. The sparse ones perform better than the embedding techniques word2vec and do2vec for LR and SVM, while there is little difference for RF. Doc2vec performs the worst in general. Concerning the classifiers, RF has lower performance overall but a better performance for word2vec and doc2vec compared to their results for LR and RF. While the deep learning model for level 1 has a comparable but lower performance than BERT vectorization, it slips down to the performance of sparse vectors at level 3. Finally, the additional knowledge seems to lead to slightly better results for word2vec except for RF for level 1. Doc2vec, in contrast, often shows slightly better results without adding the knowledge. Another noticeable result is that macro results are often worse than the accuracy implied, except for the BERT vectorizer. All these trends require explanation. To get a better insight into the black box of classifiers and vectorization techniques, it is helpful to look at the actual predictions of the classifiers.

Possible Explanations for Level 1 Results

For this purpose, the predictions of the test dataset are used. In the first step, the predictions for all methods and classifiers are obtained. Then the distribution of the class labels is analyzed to find possible patterns. The left-hand side of figure 10 shows the share of prediction labels of LR for all vectorization techniques except the ones with additional knowledge. For all methods, class 2 exhibit the highest prediction shares. Interestingly,



Share of predictions labels for level 1 - LR



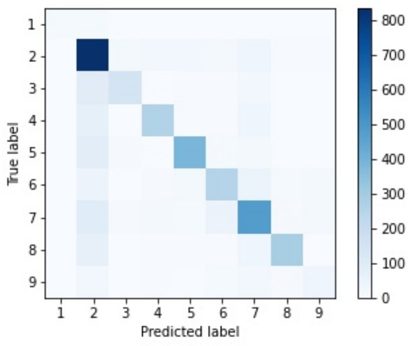
Frequency of correct predictions for each labels and frequency of true labels - LR

Figure 10: Confusion matrices - LR

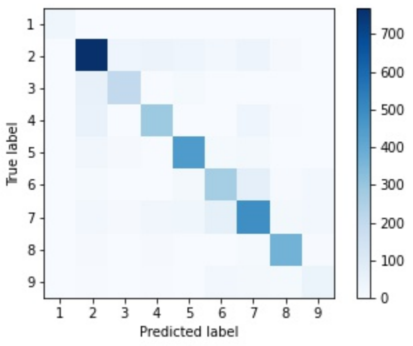
the share of the two embedding techniques, word2vec and docv2vec, is about the same but much higher than for other methods. Likewise, the sparse vectors form a group and have a substantially higher share than BERT. If we look at the proportions of BERT, the labels are more evenly distributed. There are two possible explanations for this. Either class 2 is represented in a high proportion in the test dataset, and word2vec and doc2vec have led to better recognition, or the classifiers are more or less biased towards class 2, depending on the method. The class distribution in Figure 1a shows that the data is imbalanced in favor of class 2. In addition the macro scores, which capture the imbalanced data better, are lower for most models . Thus, it is likely that the classifiers are biased.

To shed light on this issue, one can look at the correct predictions. The right-hand side of figure 10 shows the correct predictions for each method for LR, as well as the number of true labels for each class. At first glance, it looks like the classifiers predicted class 2 very well, especially compared to BERT. However, for the other labels, the two embedding techniques, doc2vec and word2vec, have a much worse performance than the other methods, indicating a bias. The situation is similar for the sparse vectors compared to BERT. Obviously, the excellent performance for class 2 is not due to the good differentiation of class 2, but to the fact that simply a very high proportion of predictions lay in class 2.

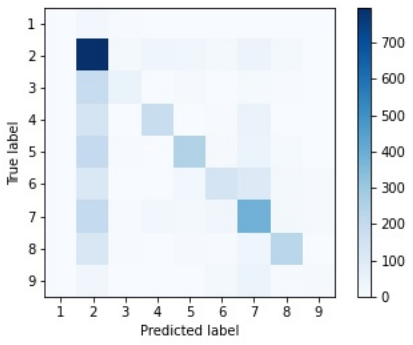
To add further evidence for the presence of the bias, it is interesting to look at the predictions of class label 2 in comparison to the true labels. For this purpose, the confusion matrix as explained in 5 can be considered. Figure 11 shows the confusion matrices for all methods without additional knowledge. Also, TF-IDF is left out because the sparse vectors behave relatively similarly. Thus, it is enough to check one of the techniques. The



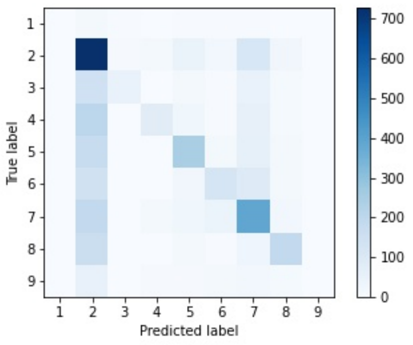
Count Vectorizer



BERT

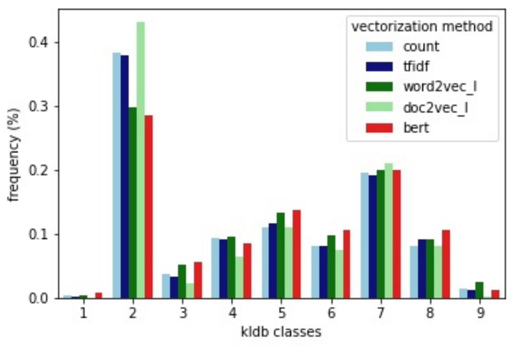


word2vec_I

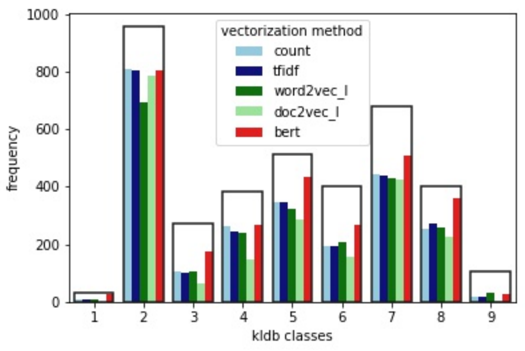


doc2vec_I

Figure 11: Confusion matrices - LR



Share of predictions labels for level 1 - RF



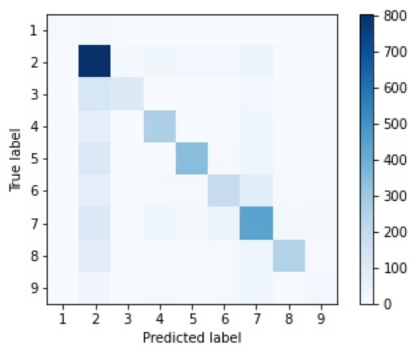
Frequency of correct predictions for each labels and frequency of true labels - RF

Figure 12: Confusion matrices level 1- RF

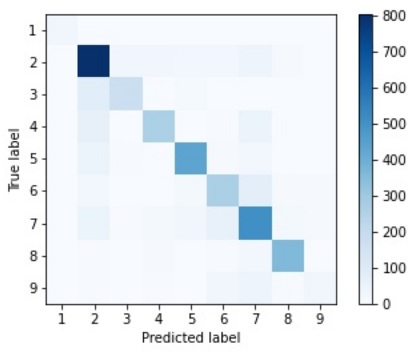
x-axis represents the predicted labels, and the y-axis the actual labels. For doc2vec and word2vec, a vertical line is visible at the predicted label 2. This line shows that both methods often classified labels as 2, although they belong to one of the other labels. At the same time, they did not do so for the other labels. For doc2vec, this vertical line is even more apparent. The count vectorizer also has a stronger vertical line compared to BERT, while BERT seems to have treated all labels equally. Figure 11 thus again provides strong evidence for the bias.[14]

Investigating SVM, it shows the same picture as LR. The corresponding plots are in the appendix. In contrast, RF behaves differently. Figure 12 shows the share of predictions on the left-hand side and the frequency of correct predictions on the right-hand side. While the sparse techniques almost maintain their share of label predictions, word2vec shows a significant drop in the share of predictions for class 2. It is also lower for doc2vec, but not to the same extent. Looking at the right-hand side figure, we can conclude that the sparse techniques and word2vec have the same number of predictions for almost all labels and are closer to BERT. Just like using LR,doc2vec classifies again very well for label 2 because the share of predictions for 2 is generally higher, but it underestimates the other labels. From Figure 18, no clear bias can be inferred for the sparse techniques and word2vec. However, there seems to be no difference between the methods. The confusion matrices show no difference between word2vec and count vectorizer, but there is slight favoritism for label 2. Meanwhile, doc2vec shows a clear favoring of label 2. BERT remains unchanged.

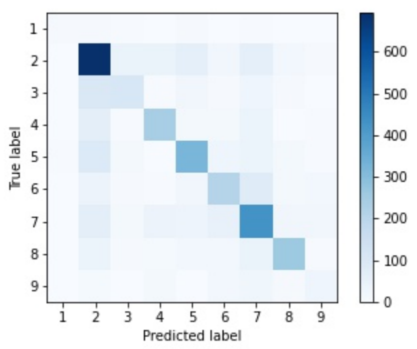
[14]Detecting a labeling bias for a classification algorithm requires systematic analysis, as proposed by Jiang and Nachum (2020). Since this is beyond the scope of this paper, the confusion matrix, which gives a good indication of whether a possible bias exists, is used here.



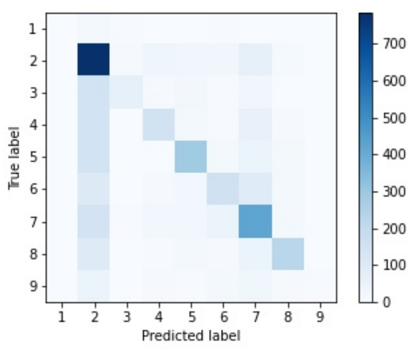
Count Vectorizer



BERT



word2vec_I



doc2vec_I

Figure 13: Confusion matrices level 1 - RF

Based on the previous analyses, the question arises as to how the results obtained relate to the performances in tables 6 and 7. In light of the literature on fundamentally imbalanced data, the analysis provides a possible explanation for the disparity in performance. In their research, Paduraru and Breaban (2019) compared imbalanced text data with several vectorization and classification techniques. They conclude for smaller datasets that sparse techniques are not as biased as more complex techniques, like doc2vec. Furthermore, the classifiers play a role as well. Decision Trees can handle imbalanced data better than linear techniques like SVM and LR. Systematic reviews and empirical studies confirm the better handling of imbalanced data by Decision Trees and RF (Kaur et al., 2019; Muchlinski et al., 2016; Krawczyk, 2016).

In line with this literature, the analysis above shows that for the linear classifiers the sparse vectors are not as affected by the bias as word2vec and doc2vec. This is reflected in the scores of LR and of SVM in Table 6 and 7 and may explain the better performance compared to word2vec and doc2vec. Besides a strong bias, the worst performance of doc2vec could be attributed to the fact that doc2vec was not trained from a pretrained model. According to the fact that BERT is not affected by the bias, it performs better. Furthermore, although the bias in RF is present, all methods are affected by it to a similar extent, except for doc2vec. This explains the decrease of the differences in scores between the sparse techniques and word2vec and the persistent poor performance of doc2vec. At the same time, it also explains that word2vec and doc2vec perform better overall with RF because RF can handle imbalanced data better. Since both methods are more affected by the bias overall, this has positively affected their performance. The outstanding performance of BERT with RF cannot be explained by the analysis above.

Besides this, the analysis does not explain the generally better performance of LR and SVM compared to RF for the dense techniques and BERT. One would expect RF to perform better overall based on the imbalanced data. However, SVM is said to handle large features better since they can learn independently of the dimensionality of the feature space. They are known to perform well for sparse vectors with a lot of relevant features, which is usually the case for text classification (Joachims, 1998). In addition for multiclass tasks, as mentione d in the literature review, often different versions of SVM are used and showed good performance (Aiolli et al., 2005; Angulo et al., 2003; Bennani and Benabdeslem, 2006; Guo and Wang, 2015; Mayoraz and Alpaydin, 1999; Tang et al., 2019b; Tomar and Agarwal, 2015). LR is preferred mainly for the sake of interpretability. A clear theoretical argument for the better performance of LR in this application cannot be found. Ultimately, the results confirm previous findings in the literature. The performance of SVM and especially of LR might be due to the fact that LR and SVM simply work better with the data and the feature. Considering the deep learning BERT confusion matrix in figure 14, there seems to be no bias problem, which leaves the question of why does BERT vectorizer with LR work better? One explanation is the difference in the size of the data. The BERT vectorizer was trained with much more data. The deep learning model probably performs better with the bigger dataset. However, the interpretability and transparency are lower compared to the BERT vectorizer. As explained in the methods section, I controlled which pairs were given as input for the fine-tuning of the BERT vectorizer and with what similarity, making it much more intuitive than the deep learning model that extracts features automatically.

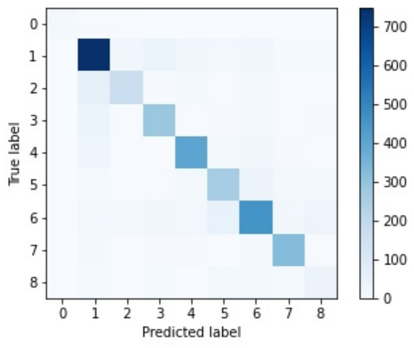
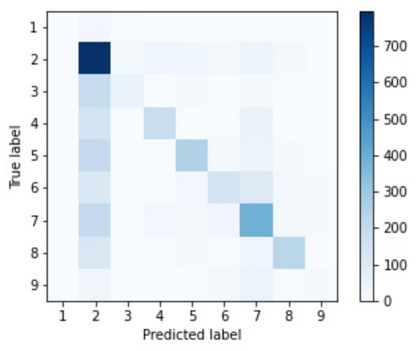


Figure 14: Covariance Matrix of BERT - Level 1)

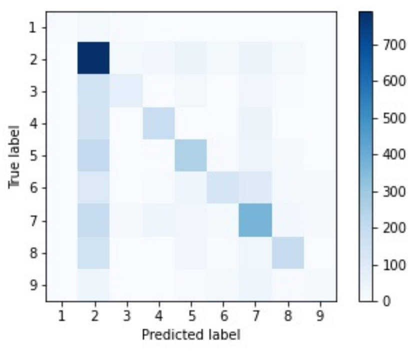
From the last comparison perspective, it is still unexplained why doc2vec_II performed lower, and word2vec_II performed slightly better. 15 and 16 report the corresponding confusion matrices for LR and RF respectively. The share of predictions and the correct predictions are reported in the appendix. Considering word2vec, there are no considerable differences. Both models seem to be biased to the same extent using LR. The same holds for RF. Thus, the slight variation between both models might be due to the additional knowledge. In contrast, doc2vec_II has more predictions with label 2 belonging to other labels. Therefore, the additional knowledge in the doc2vec model might be somewhat more biased, or adding the additional knowledge is harmful to the doc2vec vectorizer.

Possible Explanations for Level 3 Results

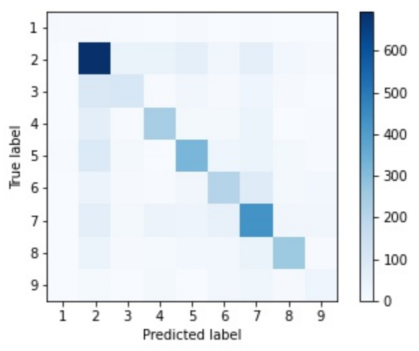
Looking at figure 1a and the fact that level 3 reveals quite the same trends for the classifiers and methods as level 1, it is reasonable to assume that the classifiers of level



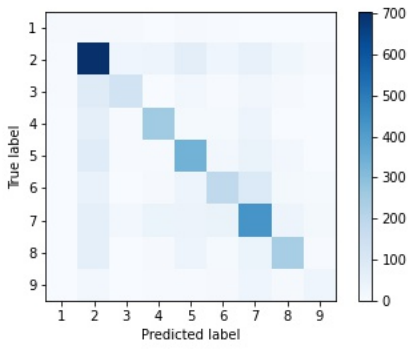
word2vec_I LR



word2vec_II LR

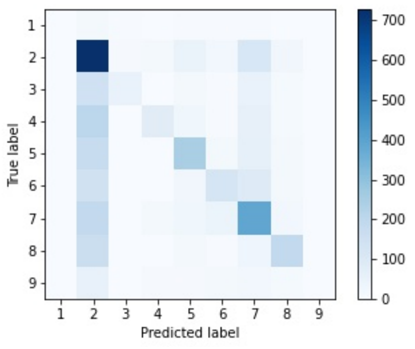


word2vec_I RF

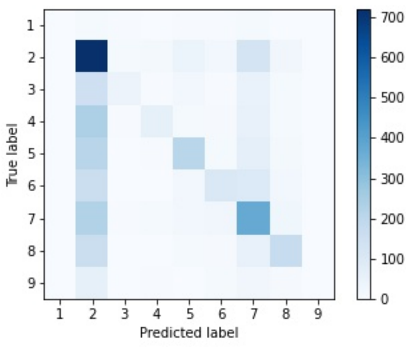


word2vec_II RF

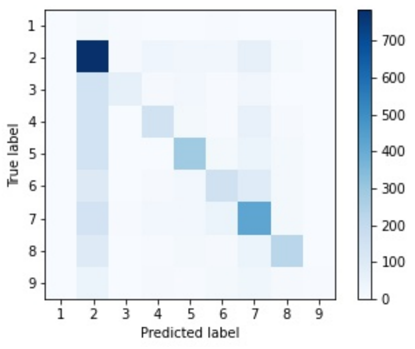
Figure 15: Confusion matrices word2vec with and without additional knowldege level 1



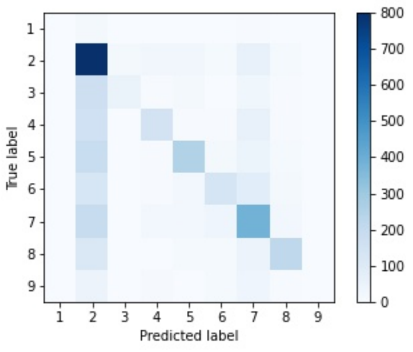
doc2vec_I LR



doc2vec_II LR



doc2vec_I RF



doc2vec_II RF

Figure 16: Confusion matrices doc2vec with and without additional knowldege level 1

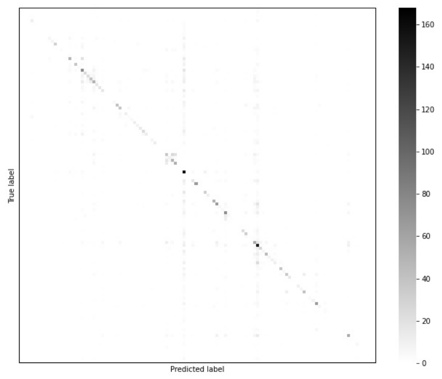
3 suffer from the same problem. However, the considerably higher number of classes makes it challenging to examine the bias visually, as was done for level 1 for the first two analyses. Instead, only the confusion matrices are studied.

According to the trends of level 3 and the explanations for level 1, it can be expected that the classifiers LR and SVM in combination with the sparse techniques as well as word2vec and doc2vec show a tendency towards imbalanced labels. This bias should be more apparent with doc2vec and word2vec. BERT, in comparison, should be less biased or not biassed at all. Figure 17 shows the respective confusion matrices. For readability, the class labels are removed. The predicted labels for word2vec and doc2vec indeed show clearly two vertical lines, which indicates a bias. While the predictions for the count vectorizer for one label seem to be biased as well, BERT again does not suffer from the imbalance of the training data. Further comparing the confusion matrices of level 1 over all classifiers to level 3 LR matrices, except for BERT, the diagonal line is not visible. This observation reflects the poor performance of the sparse techniques, word2vec and doc2vec and the excellent performance of BERT.

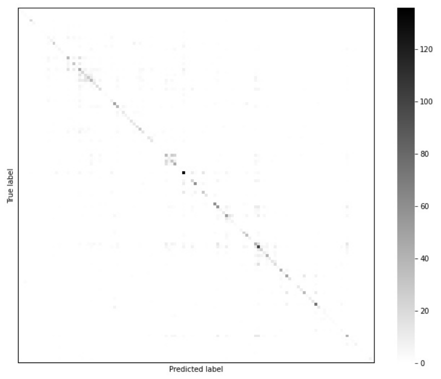
Similar to level 1, the performance for RF is, due to the better handling of imbalance, better for word2vec and doc2vec, which is reflected in their confusion matrices in figure 18. While the performance of word2vec and doc2vec are closer to the sparse techniques for RF than for the linear techniques, the confusion matrices do not clearly show differences. In addition, it is interesting that BERT has a considerably better performance with RF, while the confusion matrices do not have substantial differences in terms of the bias. Thus, the differences between the results for RF remain unexplained by the analysis.

As for level 1, the generally lower performance for RF for BERT and sparse vectors cannot be explained by the analysis and is probably because RF is not as suitable for the application as LR and SVM. More interesting is the low performance of the deep learning model. Figure 19, however, shows no differences to the BERT vectorizer confusion matrix in 18. As mentioned above, for level 1, the deep learning model trained with much less data might have a greater impact on a classification problem with much more labels. The pairs for the vectorization are also specifically tailored for level 3. According to the literature, this might be a better strategy than using a deep learning model with automatic feature selection for a small dataset.

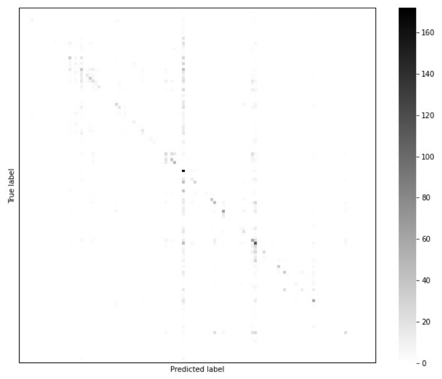
The additional knowledge models behave similarly to level 1. The confusion matrices do not provide any remarkable differences. Adding knowledge seems to improve the performance of word2vec somewhat, while it seems to be detrimental to doc2vec.



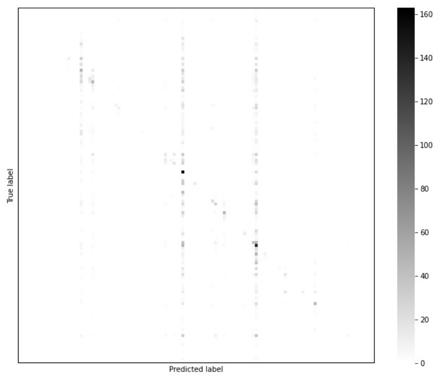
Count Vectorizer



BERT

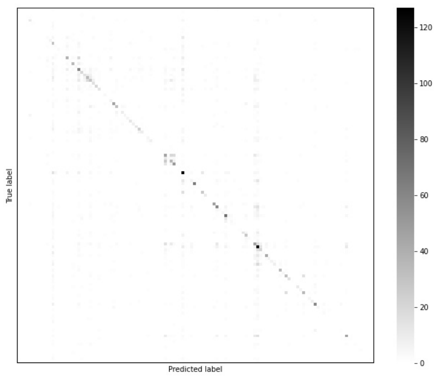


word2vec_I

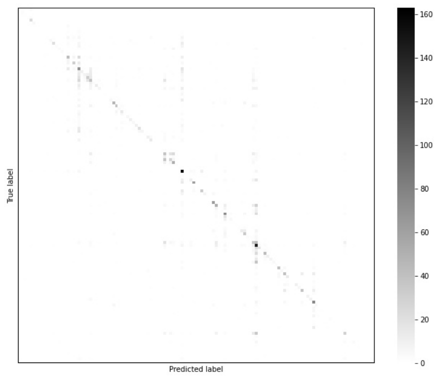


doc2vec_I

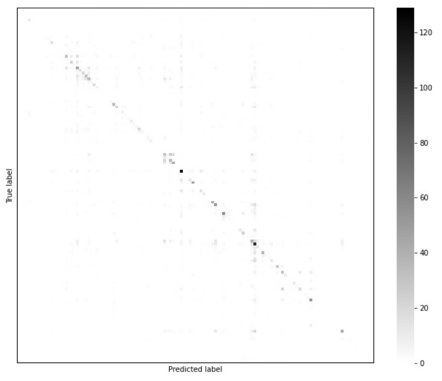
Figure 17: Confusion matrices level 3- LR



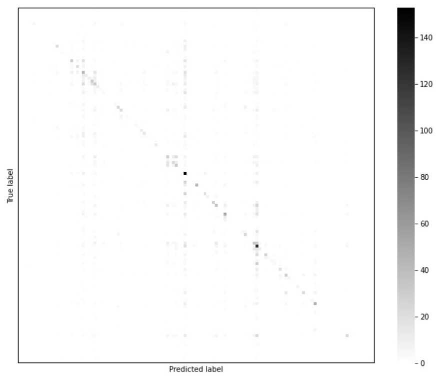
Count Vectorizer



BERT



word2vec_I



doc2vec_I

Figure 18: Confusion matrices level 3 - RF

6 Conclusion and Limitations

The starting point of this work was to examine the classification of job titles of German job postings with the taxonomy KldB 2010. Based on the literature on text classification and the challenges of short text classification, the analysis was conducted along three pillar: application of different vectorization techniques, training of different classification algorithms, and treatment of short job titles by adding additional knowledge.

The results revealed some interesting trends. The BERT vectorizer in combination with LR has shown the best performance. Due to imbalanced data, the other classification algorithms might have developed a label bias in different settings, both for level 1 and level 3. RF can reduce this bias for doc2vec and word2vec, but not to the extent that it overpowers the sparse vectors or BERT results, especially in combination with the linear classification algorithms. The deep learning model of BERT performs, in general and especially for level 3, worse than the BERT vectorizer. Adding additional knowledge does not lead to clear improvement. The performance of word2vec seems to improve slightly, whereas that of doc2vec deteriorates slightly.

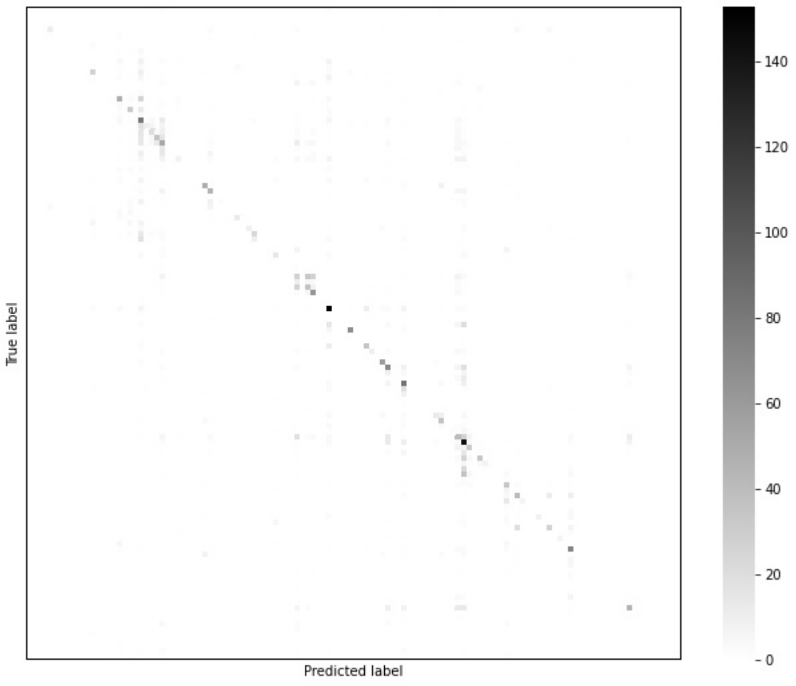


Figure 19: Confusion matrix BERT deep learning model - Level 3

In conclusion, this paper suggests that the BERT vectorizer is, due to the stable and excellent performance against imbalanced data and compared to the other classification algorithms, a promising method for classifying job titles into the *kldb* classes, especially in combination with LR. The possibility of controlling which pairs are used as input for the fine-tuning gives the BERT vectorizer the advantage of being interpretable and transparent. Furthermore, there is much space for improvements with this vectorizer method. Future research could combine pairs in other ways, for instance by including the label names of the *kldb* or by combining search words for different levels with different probabilities. This would result in vast datasets with millions of pairs for the fine-tuning process. However, corresponding computing resources must then also be available for the training process.

This paper is the first step towards establishing a job title classification system for German job postings with the KldB by providing initial results and first analysis. The results on BERT and the problem of biased classifiers are leading the way for future research. Nevertheless, some limitations require future research to develop a high-performance classification algorithm. The first limitation concerns the dataset. As detailed in the discussion of the results, the data imbalance has been problematic for most methods. This was reflected in the performance, especially for level 3, where many classes have few to no examples. Due to this fact, it was also not possible to train all taxonomy classes - 8 out of the 144 classes on level 3 were not trained. For level 1, class 0 could not be trained. In addition, this problem led to ill-defined metrics, which in part only allowed a cautious interpretation of the performance. Further, only a sample of the dataset could be used in this work due to limited computational resources. A larger dataset is likely to improve the results, especially those of the deep learning model. Additionally, future research should investigate how to cope with the imbalanced data in order to compare the vectorization methods without the bias problem. Another strategy that should be pursued, is to enrich the data manually with examples for minority classes. Such efforts are also vital to ensure that all classes are trained.

The second limitation addresses the question of generalization. A frequent problem is overfitting the classifier to the trainings data. The data was divided into training and test data to ensure that the performance was measured on unseen data. However, since not all classes are trained, the classifiers cannot classify titles belonging to those missing classes, restricting the generalizability of the algorithms. Furthermore, the classifiers are not multilingual and therefore cannot be generalized to job postings in other languages.

Lastly, there is certainly room for improving the usage of additional knowledge. As BERT is a promising technique, one could combine and compare different BERT algorithms with additional knowledge. For example, Ostendorff et al. (2019) use an approach of enriching BERT with knowledge graph embeddings. This is a relevant topic for future work.

In addition to the above limitations, in general, the complexity of structuring and normalization job titles limits the ability to improve the performance of the job title classification algorithms. Considering the occupation software developer, figure 20 shows the *kldb* that are assigned to the title with the terms “softwareentwickler” or “softwareentwicklerin” in the training dataset. Most of the titles belong to “434”, but approx. 30% are assigned to other *kldb*. Comparing the titles in these different *kldbs*, most of them do not contain keywords other than “Softwareentwickler” or “Softwareentwicklerin” that would allow distinguishing the *kldbs*. There are even titles with exactly the same wording in the different *kldbs*. Thus, the data is ambiguous for some occupations and *kldb* making it impossible for a classifier to differentiate between the *kldb*. This reveals the complexity of the problem and the limitations of improving in the performance.

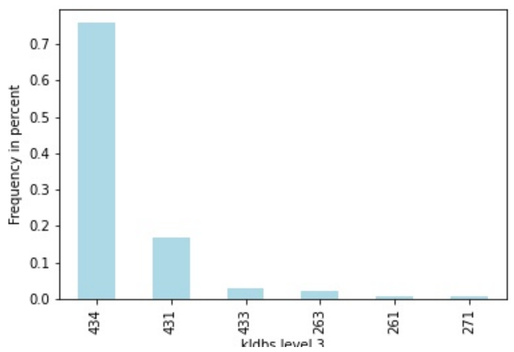


Figure 20: Share of *kldb* for the occupation ‘softwareentwickler”

The problem just highlighted raises the question of whether the limitation is merely a problem of the quality of the dataset. This can be clarified by analyzing the alternative *kldb* that employers can provide in case of uncertainty. The heatmap in 21 shows on the horizontal line the *kldb* which were used for the training and on the vertical line the alternative *kldb*. Considering again the main *kldb* 431, the vertical line that emanates from the *kldb* indicates that it is not clear at all what *kldb* a title might belong in the real world.[15] This poses the question of whether it is adequate to develop a classifier with only

[15]Note that this analysis relies on the complete dataset, not on the short one. This is due to illustrative purposes to highlight the problem better. However, since a sample with the same distribution was drawn,

one class as an output. At the same time, one should elaborate on whether a multilabel classification algorithm is practical for downstream tasks in this domain. In addition, there are concerns about appropriate evaluation measurements. This may constitute the object of future studies.

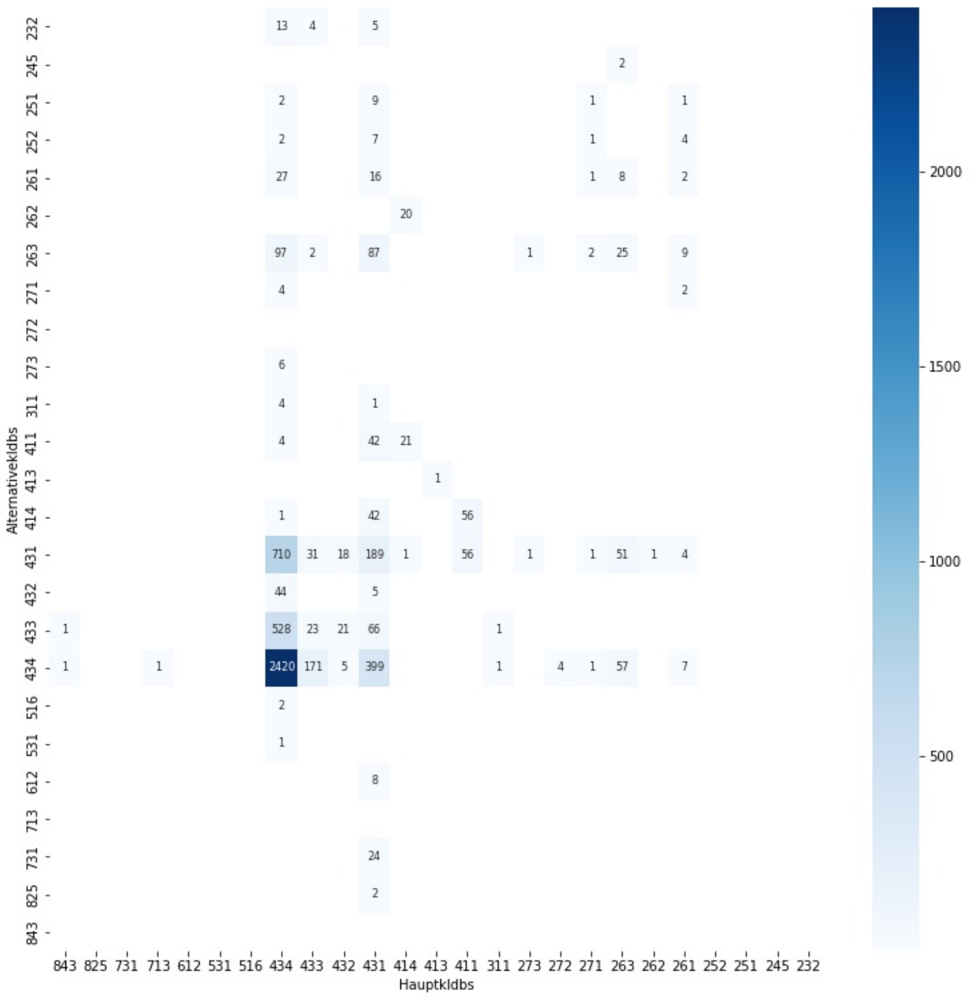


Figure 21: Co-occurence matrix for the occupation ‘softwareentwickler”

the logic is the same.

References

Aiolli, F., Sperduti, A., and Singer, Y. (2005). Multiclass classification with multi-prototype support vector machines. *Journal of Machine Learning Research*, 6(5):817–850.

Ajose-Ismail, B., Abimbola, O. V., and Oloruntoba, S. (2020). Performance analysis of different word embedding models for text classification. *International Journal of Scientific Research and Engineering Development*, 3(6):1016–1020.

Almeida, F. and Xexéo, G. (2019). Word embeddings: A survey. *arXiv preprint arXiv:1901.09069*.

Alsmadi, I. and Gan, K. H. (2019). Review of short-text classification. *International Journal of Web Information Systems*.

Aly, M. (2005). Survey on multiclass classification methods. *Neural Netw*, 19:1–9.

Angulo, C., Parra, X., and Catala, A. (2003). K-svcr. a support vector machine for multi-class classification. *Neurocomputing*, 55(1-2):57–77.

Arora, M., Mittal, V., and Aggarwal, P. (2021). Enactment of tf-idf and word2vec on text categorization. In *Proceedings of 3rd International Conference on Computing Informatics and Networks: ICCIN 2020*, pages 199–209. Springer Singapore.

Aßenmacher, M., Corvonato, A., and Heumann, C. (2021). Re-evaluating germeval17 using german pre-trained language models. *arXiv preprint arXiv:2102.12330*.

Ayesha, S., Hanif, M. K., and Talib, R. (2020). Overview and comparative study of dimensionality reduction techniques for high dimensional data. *Information Fusion*, 59:44–58.

Bennani, Y. and Benabdeslem, K. (2006). Dendogram-based svm for multi-class classification. *Journal of Computing and Information Technology*, 14(4):283–289.

Berthold, M. R., Borgelt, C., Höppner, F., Klawonn, F., and Silipo, R. (2020). *Guide to Intelligent Data Science*. Springer.

Bird, S., Klein, E., and Loper, E. (2009). *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Bisong, E. (2019). *Building machine learning and deep learning models on Google Cloud Platform*. Springer.

Bouaziz, A., Dartigues-Pallez, C., Costa Pereira, C. d., Precioso, F., and Lloret, P. (2014). Short text classification using semantic random forest. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 288–299. Springer.

Boydston, P. S. and Hirst, E. S. J. (2020). Public perceptions and understanding of job titles related to behavior analysis. *Behavior Analysis in Practice*, 13(2):394–401.

Branco, P., Torgo, L., and Ribeiro, R. P. (2017). Relevance-based evaluation metrics for multi-class imbalanced domains. pages 698–710.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Brijain, M., Patel, R., Kushik, M., and Rana, K. (2014). A survey on decision tree algorithm for classification.

Bro, R. and Smilde, A. K. (2014). Principal component analysis. *Analytical methods*, 6(9):2812–2831.

Buckland, M. and Gey, F. (1994). The relationship between recall and precision. *Journal of the American society for information science*, 45(1):12–19.

Bundesagentur für Arbeit, B., editor (2011a). *Klassifikation der Berufe 2010 Band 1: Systematischer und alphabetischer Teil mit Erläuterungen.*

Bundesagentur für Arbeit, B. (2011b). Klassifikation der berufe 2010 (kldb 2010) – aufbau und anwenderbezogene hinweise. Technical report.

Cahyani, D. E. and Patasik, I. (2021). Performance comparison of tf-idf and word2vec models for emotion text classification. *Bulletin of Electrical Engineering and Informatics*, 10(5):2780–2788.

Chang, C.-C. and Lin, C.-J. (2001). Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2.3:1–27.

Chauhan, N. K. and Singh, K. (2018). A review on conventional machine learning vs deep learning. In *2018 International Conference on Computing, Power and Communication Technologies (GUCON)*, pages 347–352. IEEE.

Chen, J., Hu, Y., Liu, J., Xiao, Y., and Jiang, H. (2019). Deep short text classification with knowledge powered attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6252–6259.

Colas, F. and Brazdil, P. (2006). Comparison of svm and some older classification algorithms in text classification tasks. pages 169–178.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Crammer, K. and Singer, Y. (2001). On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of machine learning research*, 2(Dec):265–292.

Cutler, A., Cutler, D. R., and Stevens, J. R. (2012). Random forests. In *Ensemble machine learning*, pages 157–175. Springer.

Danso, S., Atwell, E., and Johnson, O. (2014). A comparative study of machine learning methods for verbal autopsy text classification. *arXiv preprint arXiv:1402.4380*.

Decorte, J.-J., Van Hautte, J., Demeester, T., and Develder, C. (2021). Jobbert: Understanding job titles through skills. *arXiv preprint arXiv:2109.09605*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Farooq, A., Anwar, S., Awais, M., and Rehman, S. (2017). A deep cnn based multi-class classification of alzheimer's disease using mri. In *2017 IEEE International Conference on Imaging systems and techniques (IST)*, pages 1–6. IEEE.

Fatourechi, M., Ward, R. K., Mason, S. G., Huggins, J., Schloegl, A., and Birch, G. E. (2008). Comparison of evaluation metrics in classification applications with imbalanced datasets. pages 777–782.

Fei, Y., Rong, G., Wang, B., and Wang, W. (2014). Parallel l-bfgs-b algorithm on gpu. *Computers & graphics*, 40:1–9.

Geladi, P. and Linderholm, J. (2020). 2.03 - principal component analysis. In Brown, S., Tauler, R., and Walczak, B., editors, *Comprehensive Chemometrics (Second Edition)*, pages 17–37. Elsevier, Oxford.

Gonçalves, T. and Quaresma, P. (2005). Evaluating preprocessing techniques in a text classification problem. *São Leopoldo, RS, Brasil: SBC-Sociedade Brasileira de Computação.*

González-Carvajal, S. and Garrido-Merchán, E. C. (2020). Comparing bert against traditional machine learning text classification. *arXiv preprint arXiv:2005.13012.*

Grandini, M., Bagli, E., and Visani, G. (2020). Metrics for multi-class classification: an overview. *arXiv preprint arXiv:2008.05756.*

Guo, H. and Wang, W. (2015). An active learning-based svm multi-class classification model. *Pattern recognition*, 48(5):1577–1597.

HaCohen-Kerner, Y., Miller, D., and Yigal, Y. (2020). The influence of preprocessing on text classification using a bag-of-words representation. *PloS one*, 15(5):e0232525.

Han, J., Pei, J., and Kamber, M. (2011). *Data mining: concepts and techniques.* Elsevier.

Hassan, A. and Mahmood, A. (2017). Efficient deep learning model for text classification based on recurrent and convolutional layers. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1108–1113. IEEE.

Hsu, C.-W. and Lin, C.-J. (2002). A comparison of methods for multiclass support vector machines. *IEEE transactions on Neural Networks*, 13(2):415–425.

Javed, F., Luo, Q., McNair, M., Jacob, F., Zhao, M., and Kang, T. S. (2015). Carotene: A job title classification system for the online recruitment domain. In *2015 IEEE First International Conference on Big Data Computing Service and Applications*, pages 286–293. IEEE.

Javed, F., McNair, M., Jacob, F., and Zhao, M. (2016). Towards a job title classification system. *arXiv preprint arXiv:1606.00917.*

Jiang, H. and Nachum, O. (2020). Identifying and correcting label bias in machine learning. In *International Conference on Artificial Intelligence and Statistics*, pages 702–712. PMLR.

Jin, P., Zhang, Y., Chen, X., and Xia, Y. (2016). Bag-of-embeddings for text classification. In *IJCAI*, volume 16, pages 2824–2830.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. pages 137–142.

Jordan, M., Kleinberg, J., and Schölkopf, B. (2006). *Pattern Recognition and Machine Learning*. Springer.

Jurafsky, D. and Martin, J. H. (2021). Speech and language processing - an introduction to natural language processing, computational linguistics, and speech recognition. "`https://web.stanford.edu/~jurafsky/slp3/old_dec20/ed3book_dec302020.pdf`",.

Kamath, C. N., Bukhari, S. S., and Dengel, A. (2018). Comparative study between traditional machine learning and deep learning approaches for text classification. In *Proceedings of the ACM Symposium on Document Engineering 2018*, pages 1–11.

Karimi, S., Yin, J., and Paris, C. (2013). Classifying microblogs for disasters. In *Proceedings of the 18th Australasian Document Computing Symposium*, pages 26–33.

Kaur, H., Pannu, H. S., and Malhi, A. K. (2019). A systematic review on imbalanced data challenges in machine learning: Applications and solutions. *ACM Computing Surveys (CSUR)*, 52(4):1–36.

Kautz, T., Eskofier, B. M., and Pasluosta, C. F. (2017). Generic performance measure for multiclass-classifiers. *Pattern Recognition*, 68:111–125.

Khamar, K. (2013). Short text classification using knn based on distance function. *International Journal of Advanced Research in Computer and Communication Engineering*, 2(4):1916–1919.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kowsari, K., Jafari Meimandi, K., Heidarysafa, M., Mendu, S., Barnes, L., and Brown, D. (2019). Text classification algorithms: A survey. *Information*, 10(4):150.

Krawczyk, B. (2016). Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, 5(4):221–232.

Kuang, Q. and Xu, X. (2010). Improvement and application of tf• idf method based on text classification. In *2010 International Conference on Internet Technology and Applications*, pages 1–4. IEEE.

Kulkarni, A. and Shivananda, A. (2021). *Natural language processing recipes.* Springer.

Lau, J. H. and Baldwin, T. (2016). An empirical evaluation of doc2vec with practical insights into document embedding generation. *arXiv preprint arXiv:1607.05368.*

Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR.

Li, L., Peltsverger, S., Zheng, J., Le, L., and Handlin, M. (2021). Retrieving and classifying linkedin job titles for alumni career analysis. In *Proceedings of the 22st Annual Conference on Information Technology Education*, pages 85–90.

Li, T., Zhang, C., and Ogihara, M. (2004). A comparative study of feature selection and multiclass classification methods for tissue classification based on gene expression. *Bioinformatics*, 20(15):2429–2437.

Li, Y., Xu, L., Tian, F., Jiang, L., Zhong, X., and Chen, E. (2015). Word embedding revisited: A new representation learning and explicit matrix factorization perspective. In *Twenty-Fourth International Joint Conference on Artificial Intelligence.*

Maglogiannis, I. G. (2007). *Emerging artificial intelligence applications in computer engineering: real word ai systems with applications in ehealth, hci, information retrieval and pervasive technologies*, volume 160. Ios Press.

Malherbe, E., Diaby, M., Cataldi, M., Viennet, E., and Aufaure, M.-A. (2014). Field selection for job categorization and recommendation to social network users. In *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)*, pages 588–595. IEEE.

Marinescu, I. and Wolthoff, R. (2020). Opening the black box of the matching function: The power of words. *Journal of Labor Economics*, 38(2):535–568.

Martin-Caughey, A. (2021). What's in an occupation? investigating within-occupation variation and gender segregation using job titles and task descriptions. *American Sociological Review*, 86(5):960–999.

Mayoraz, E. and Alpaydin, E. (1999). Support vector machines for multi-class classification. In *International Work-Conference on Artificial Neural Networks*, pages 833–842. Springer.

Merchant, A., Rahimtoroghi, E., Pavlick, E., and Tenney, I. (2020). What happens to bert embeddings during fine-tuning? *arXiv preprint arXiv:2004.14448.*

Miaschi, A. and Dell'Orletta, F. (2020). Contextual and non-contextual word embeddings: an in-depth linguistic investigation. In *Proceedings of the 5th Workshop on Representation Learning for NLP*, pages 110–119.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781.*

Muchlinski, D., Siroky, D., He, J., and Kocher, M. (2016). Comparing random forest with logistic regression for predicting class-imbalanced civil war onset data. *Political Analysis*, 24(1):87–103.

Narasimhan, H., Pan, W., Kar, P., Protopapas, P., and Ramaswamy, H. G. (2016). Optimizing the multiclass f-measure via biconcave programming. pages 1101–1106.

Neculoiu, P., Versteegh, M., and Rotaru, M. (2016). Learning text similarity with siamese recurrent networks. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 148–157.

Ostendorff, M., Bourgonje, P., Berger, M., Moreno-Schneider, J., Rehm, G., and Gipp, B. (2019). Enriching bert with knowledge graph embeddings for document classification. *arXiv preprint arXiv:1909.08402.*

Padurariu, C. and Breaban, M. E. (2019). Dealing with data imbalance in text classification. *Procedia Computer Science*, 159:736–745.

Paulus, W., Matthes, B., et al. (2013). Klassifikation der berufe: Struktur, codierung und umsteigeschlüssel. Technical report, Forschungsdatenzentrum der Bundesagentur für Arbeit im Institut für Arbeitsmarkt- und Berufsforschung.

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

Polikar, R. (2012). Ensemble learning. In *Ensemble machine learning*, pages 1–34. Springer.

Rahmawati, D. and Khodra, M. L. (2016). Word2vec semantic representation in multilabel classification for indonesian news article. In *2016 International Conference On Advanced Informatics: Concepts, Theory And Application (ICAICTA)*, pages 1–6. IEEE.

Ravichandiran, S. (2021). *Getting Started with Google BERT: Build and train state-of-the-art natural language processing models using BERT.* Packt Publishing.

Refaeilzadeh, P., Tang, L., and Liu, H. (2009). Cross-validation. *Encyclopedia of database systems*, 5:532–538.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Robertson, S. (2004). Understanding inverse document frequency: on theoretical arguments for idf. *Journal of documentation*.

Rogers, A., Kovaleva, O., and Rumshisky, A. (2020). A primer in bertology: What we know about how bert works. *Transactions of the Association for Computational Linguistics*, 8:842–866.

Rong, X. (2014). word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*.

Safavian, S. R. and Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674.

Sahlgren, M. (2008). The distributional hypothesis. *Italian Journal of Disability Studies*, 20:33–53.

Sarkar, D. (2016). *Text Analytics with python.* Springer.

Schapire, R. E. and Singer, Y. (2000). Boostexter: A boosting-based system for text categorization. *Machine learning*, 39(2):135–168.

Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47.

Shah, K., Patel, H., Sanghvi, D., and Shah, M. (2020). A comparative analysis of logistic regression, random forest and knn models for the text classification. *Augmented Human Research*, 5(1):1–16.

Shao, Y., Taylor, S., Marshall, N., Morioka, C., and Zeng-Treitler, Q. (2018). Clinical text classification with word embedding features vs. bag-of-words features. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2874–2878. IEEE.

Sidorov, G. (2019). *Syntactic n-grams in computational linguistics.* Springer.

Simonton, T. M. and Alaghband, G. (2017). Efficient and accurate word2vec implementations in gpu and shared-memory multicore architectures. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7. IEEE.

Singh, A. K. and Shashi, M. (2019). Vectorization of text documents for identifying unifiable news articles. *Int. J. Adv. Comput. Sci. Appl*, 10.

Slamet, C., Andrian, R., Maylawati, D. S., Darmalaksana, W., Ramdhani, M., et al. (2018). Web scraping and naïve bayes classification for job search engine. In *IOP Conference Series: Materials Science and Engineering*, volume 288, page 012038. IOP Publishing.

Smith, B. N., Hornsby, J. S., Benson, P. G., and Wesolowski, M. (1989). What is in a name: The impact of job titles on job evaluation results. *Journal of Business and Psychology*, 3(3):341–351.

Smith, L. I. (2002). A tutorial on principal components analysis.

Song, G., Ye, Y., Du, X., Huang, X., and Bie, S. (2014). Short text classification: A survey. *Journal of multimedia*, 9(5):635–643.

Sriram, B., Fuhry, D., Demir, E., Ferhatosmanoglu, H., and Demirbas, M. (2010). Short text classification in twitter to improve information filtering. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 841–842.

Suleymanov, U., Kalejahi, B. K., Amrahov, E., and Badirkhanli, R. (2019). Text classification for azerbaijani language using machine learning and embedding.

Sun, A. (2012). Short text classification using very few words. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 1145–1146.

Sun, C., Qiu, X., Xu, Y., and Huang, X. (2019). How to fine-tune bert for text classification? In *China National Conference on Chinese Computational Linguistics*, pages 194–206. Springer.

Tang, L., Tian, Y., and Pardalos, P. M. (2019a). A novel perspective on multiclass classification: Regular simplex support vector machine. *Information Sciences*, 480:324–338.

Tang, L., Tian, Y., and Pardalos, P. M. (2019b). A novel perspective on multiclass classification: Regular simplex support vector machine. *Information Sciences*, 480:324–338.

Tipping, M. E. and Bishop, C. M. (1999). Mixtures of probabilistic principal component analyzers. *Neural computation*, 11(2):443–482.

Tomar, D. and Agarwal, S. (2015). A comparison on multi-class classification methods based on least squares twin support vector machine. *Knowledge-Based Systems*, 81:131–147.

Uter, W. (2020). *Classification of Occupations*, pages 61–67. Springer International Publishing.

Uysal, A. K. and Gunal, S. (2014). The impact of preprocessing on text classification. *Information processing & management*, 50(1):104–112.

Vapnik, V. (1999). *The nature of statistical learning theory.* Springer science & business media.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Vijayan, V. K., Bindu, K., and Parameswaran, L. (2017). A comprehensive study of text classification algorithms. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1109–1113. IEEE.

Wang, F., Wang, Z., Li, Z., and Wen, J.-R. (2014). Concept-based short text classification and ranking. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 1069–1078.

Wang, J., Abdelfatah, K., Korayem, M., and Balaji, J. (2019). Deepcarotene-job title classification with multi-stream convolutional neural network. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 1953–1961. IEEE.

Wang, J., Wang, Z., Zhang, D., and Yan, J. (2017a). Combining knowledge with deep convolutional neural networks for short text classification. In *IJCAI*, volume 350.

Wang, Y., Zhou, Z., Jin, S., Liu, D., and Lu, M. (2017b). Comparisons and selections of features and classifiers for short text classification. In *Iop conference series: Materials science and engineering*, volume 261, page 012018. IOP Publishing.

Wang, Z. and Qu, Z. (2017). Research on web text classification algorithm based on improved cnn and svm. In *2017 IEEE 17th International Conference on Communication Technology (ICCT)*, pages 1958–1961. IEEE.

Wendland, A., Zenere, M., and Niemann, J. (2021). Introduction to text classification: Impact of stemming and comparing tf-idf and count vectorization as feature extraction technique. In *European Conference on Software Process Improvement*, pages 289–300. Springer.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Xia, F., Zhang, W., Li, F., and Yang, Y. (2008). Ranking with decision tree. *Knowledge and information systems*, 17(3):381–395.

Yan, L., Zheng, Y., and Cao, J. (2018). Few-shot learning for short text classification. *Multimedia Tools and Applications*, 77(22):29799–29810.

Yuan, Y., Wu, L., and Zhang, X. (2021). Gini-impurity index analysis. *IEEE Transactions on Information Forensics and Security*, 16:3154–3169.

Zhang, X., Zhao, J., and LeCun, Y. (2015). Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28:649–657.

Zhou, Z.-H. (2009). Ensemble learning. *Encyclopedia of biometrics*, 1:270–273.

Zhu, W., Zhang, W., Li, G.-Z., He, C., and Zhang, L. (2016). A study of damp-heat syndrome classification using word2vec and tf-idf. In *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1415–1420. IEEE.

Zhu, Y., Javed, F., and Ozturk, O. (2017). Document embedding strategies for job title classification. In *The Thirtieth International Flairs Conference*.

A Data

A.1 Data snippet raw data

```json
{
  "hashId": "-IgNS05-jeri5aZhe0_VK35Y0-6xQAoADg3b0MyraTI=",
  "hauptberuf": "Telefonist/in",
  "freieBezeichnung": "Telefonist / Telefonistin m/w/d",
  "referenznummer": "14469-20210617140207-S",
  "mehrereArbeitsorteVorhanden": false,
  "arbeitgeber": "aventa Personalmanagement GmbH",
  "arbeitgeberHashId": "MYRG2meMKxCjrQ9Cpl8JwgEDPbM133Z9iRCkolaOONo=",
  "aktuelleVeroeffentlichungsdatum": "2021-06-29",
  "eintrittsdatum": "2021-06-29",
  "logoHashId": "wMN78p7yNK_C0aJDJ77l63RVH3DCEzwJGxZk1ZzsUrY=",
  "angebotsart": "ARBEIT",
  "hauptDkz": "7389",
  "alternativDkzs": [
    "35082"
  ],
  "angebotsartGruppe": "ARBEIT",
  "anzeigeAnonym": false,
  "arbeitsort": {
    "plz": "10407",
    "ort": "Berlin",
    "region": "Berlin",
    "land": "Deutschland",
    "koordinaten": {
      "lat": 52.5335379,
      "lon": 13.4462856
    }
  },
    "_links": {
      "details": {
        "href": "http://jobboerse.arbeitsagentur.de/vamJB/stellenangebotAnzeigen.html?bencs=xZ8NQKDByg2g6avJgLLIrGwqlXZQi1GKNAI%2BzAoCWJ5RD6egZDnwqMFj%2B4AnUX6XN5nyEJ7NKSdBBr1EvlmnVw%3D%3D"
      },
      "arbeitgeberlogo": {
        "href": "https://api-con.arbeitsagentur.de/prod/jobboerse/jobsuche-service/ed/v1/arbeitgeberlogo/wMN78p7yNK_C0aJDJ77l63RVH3DCEzwJGxZk1ZzsUrY="
      },
      "jobdetails": {
        "href": "https://api-con.arbeitsagentur.de/prod/jobboerse/jobsuche-service/pc/v1/jobdetails/-IgNS05-jeri5aZhe0_VK35Y0-6xQAoADg3b0MyraTI="
      }
    }
  },
```

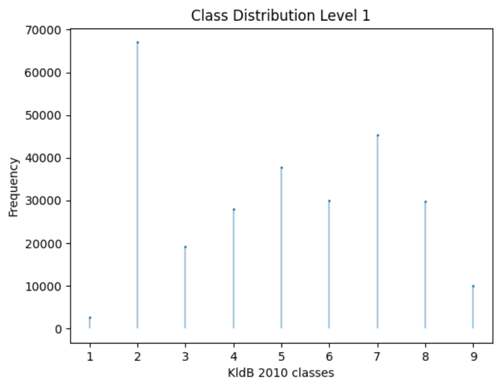
A.2 Trainingsdata snippet (without preprocessing) - Level 1

```
{'id': '2', 'title': 'Maschinenbediener (m/w/d)'}
{'id': '7', 'title': 'Controlling'}
{'id': '5', 'title': 'Lagermitarbeiter (m/w/d)'}
{'id': '2', 'title': 'Reifenmonteur (m/w/d) Facharbeiter'}
{'id': '5', 'title': 'Kommissionierer (m /w /d)'}
{'id': '7', 'title': 'Sachbearbeiter (m/w/d) im Einkauf Weimar'}
{'id': '5', 'title': 'Schubmaststapler Fahrer (m/w/d)'}
{'id': '3', 'title': 'Bauhelfer Elektroinstallation (m/w/d)'}
{'id': '7', 'title': 'Telefonist / Telefonistin m/w/d'}
{'id': '9', 'title': 'Telefonische Kundenbetreuung (m/w/d)'}
```

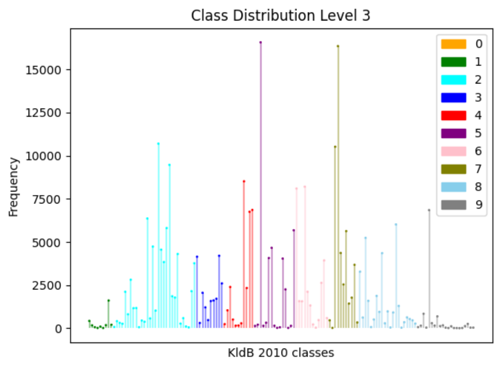
A.3 Trainingdata snippted (preprocessed) - Level 1

```
[{'id': '2', 'title': 'maschinenbediener'},
 {'id': '7', 'title': 'controlling'},
 {'id': '5', 'title': 'lagermitarbeiter'},
 {'id': '2', 'title': 'reifenmonteur facharbeiter'},
 {'id': '5', 'title': 'kommissionierer'},
 {'id': '7', 'title': 'sachbearbeiter einkauf weimar'},
 {'id': '5', 'title': 'schubmaststapler fahrer'},
 {'id': '3', 'title': 'bauhelfer elektroinstallation'},
 {'id': '7', 'title': 'telefonist telefonistin'},
 {'id': '9', 'title': 'telefonische kundenbetreuung'}]
```

A.4 Class distribution of level 1 und level 3



(a) Class distribution level 1



(b) Class distribution level 3

Figure 22: Class distribution of training data

B Results