Job Title Classification Strategies for the German Labor Market

Masterthesis

submitted by

Rahkakavee Baskaran

at the

Universität Konstanz

Department of Politics and Public Administration

Center for Data and Methods

1.Gutachter: Prof. Dr. Susumu Shikano

2.Gutachter: JunProf Juhi Kulshresthra

Konstanz, December 31, 2021

Contents

Abbreviations

1 Introduction

2 Related work

2.1 Textclassification

Text classification, a highly researched area, is the process of classifying text documents or text segments into a set of predefined classes. During the last decades, researches developed a various number of classifiers. As Kowsari et al. (2019) summarize in their survey of classifiers, we can group the approaches mainly into three groups. The first group contains traditional methods like Naive Bayes (NV), Support Vector Machines (SVM), K-nearest neighbors (KNN), Logistic Regressions (LR) or Decision Trees (DT) (Vijayan et al., 2017; Colas and Brazdil, 2006; Kowsari et al., 2019; Sebastiani, 2001). Deep learning methods like Convolutional Neural Networks (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.

2.2 Short Text classification

Another potential issue is the length of input documents for classification. Job titles are clearly short text with often not more than 50 characters. Short texts suffer from sparseness, few word co-occurrences, missing shared context, noisiness and ambiguity. Traditional methods, however, are based on word frequency, high word co-occurrence and context, which is why they often fail to achieve high accuracy for short texts (Song et al., 2014; Wang et al., 2017, 2014). In their overview, Song et al. (2014) present three approaches to solve this. First, since short text data often suffers from unlabeled data in the context of online text data, such as Twitter postings, they suggest using semi-supervised approaches. Second, they recommend to use ensemble learning methods, which focus on the sparseness of the data. Third, Song et al. (2014) propose feature dimensionality reduction and extraction of semantic relationship methods. Based on the latter more recent work on short text classification criticizes the use of the "Bag of Word" concept for feature representation as it only reflects the appearance of words in the text. Instead, they represent short texts with semantically similar and conceptual information

(Bouaziz et al., 2014; Wang et al., 2014; Chen et al., 2019). Another question concerning the representation of short texts is whether to represent them as dense or sparse vectors. In their comparison between tf-idf/counter vectorizer and the dense vectorizer word2vec and doc2vec, Wang et al. (2017) conclude that among the classifiers Naive Bayes, Logisitic Regression 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 outperform state-of-art deep learning methods, like CNN, for Chinese short texts. Concerning the classifiers, there is no consensus approach for short text classification. For traditional approaches Wang et al. (2017)'s results indicate that logistic regression and SVM perform best, while KNN seems to achieve best accuracy in Khamar (2013)'s work. Similar to job title specific work, more recent work prefers deep learning methods, mostly CNN (Chen et al., 2019).

2.3 Domain-specific

Each text classification task presents different challenges. One challenge is that domain-specific problems may arise. There is some work that deals with job classification in the English speaking job market. In terms of classifiers, the corresponding work can be categorised into traditional classifiers or deep learning methods. **?** for example, use a KNN classifier in combination with document embedding as feature selection strategy. **?** rely on traditional methods as well, by combining a SVM classifier and a KNN classifier for their job recommendation system. In contrast, the approaches of **?**, **?** and **?** are based on Deep Learning methods. From a higher perspective, there is another dividing line between the approaches. As mentioned earlier, job title normalization can be considered as a typical text classification task (**???**). **?** and **?**, however, formulate the task as a string representation approach of similar job titles.

2.4 Multiclass

A last challenge of text classification tasks comes with the number of classes. As **?** show in their classification of tissue, multiclass classification is more difficult than binary classification problems. Partly, because most of classification algorithms

were designed for binary problems (**?**). Approaches for multiclassification can be grouped into two types. Binary algorithms can handle multiclassification naturally. This is, for example, the case for Regression, DT, SVM, KNN and NV. The second type is the decomposition of the problem into binary classification tasks (for the different subtypes see **?**). The literature so far does not have a clear answer to solve multiclassification problems. Different approaches, like Boosting (**?**) or CNN (**?**) are applied. It is noticeable, however, that many works use variations of SVM (Guo and Wang, 2015; Tomar and Agarwal, 2015; Tang et al., 2019).

3 Data and taxonomy

3.1 KldB 2010 Taxonomy

The "Klassifikation der Berufe 2010 (KldB)" is structured hierarchically with 5 levels. On each level there is a different number of classes. On level 1 each class has an id of length one with a number from 0 to 9. Table 1 shows the 10 classes of level 1 with their class names. On level 2, then, each of the 10 classes are 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 the all 5 levels with an example of classes is given in table 2. Note that the example in table 2 does not show on level 2 to level 5 all classes. Thus on level 2 there exists also, e.g. the class id 41 with "Mathematik-, Biologie- Chemie- und Physikberufe", which in turn is divided into other classes on level 3 etc.. With this procedure this ultimatley leads to class ids of length 5 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 jobtitle belongs, for example, on level 3 to the class "Sofwareentwicklung" (Bundesagentur für Arbeit, 2011a,b; Paulus and Matthes, 2013)

The KldB contains of two dimension. The first dimension, the so-called "Berufsfachlichkeit" structures jobs according to their similarity in knowledge, activties and jobs. This is reflected in the first 4 levels. Considering the again the example from above and the job title "Fullstack PHP-Entwickler" it is reasonable to classify both on level 1 to "Naturwissenschaft, Geografie and Information", because both of them

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 (edited after (Bundesagentur für Arbeit, 2011b))

are related to computer science. It also make sense to classifiy them for example to 4341, because both are about sofware development. On level 5, then, a second dimension is introduced. the "Anforderungsniveau". This dimension gives information on the level of requirement for a job and 4 possible requirements. In table 3 they are summarized. 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 2 (Bundesagentur für Arbeit, 2011a,b; Paulus and Matthes, 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: Overview of KldB (edited after (Bundesagentur für Arbeit, 2011b))

With the KldB 2010, an useful and information-rich occupational classification was created for Germany that reflects the current trends in the labor market (Paulus and Matthes, 2013). One strength relies in the construction of the KldB. Instead of just including expert knowledge into the Taxonomy the developement process is based on systematical consideration of information about occupations, as well as statistical procedures for taxonomy developement. Furthermore, the taxonomy was reviewed qualitatively several times in relation to professions and revised. Considering the expressiveness, the KldB has some more benefits. Since the taxonmy is

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: Overview of Level of requirements on Level 5 (edited after (Bundesagentur für Arbeit, 2011b))

quite recent, it reflects new job classes and market trends very adequately. Further, by including the second dimension, the taxonomy provides a powerful tool to organize job titles into simple requirement classes. In addition, the taxonomy also distinguishes between managerial, supervisory, and professional employees, which is also valuable information. 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)

4 Method

4.1 Conceptual overview

4.2 Preprocessing

4.3 Vectorization Techniques

4.3.1 Count vectorizer

The count vectorizer is one of most simple techniques of converting texts to vectors. It belongs to the family of Bag of Words (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 important properties. First, they do not consider the order of the words, sequences or the grammar, which is why they also called BOW. Second, each word in the corpus is represented by it's own dimension. Thus the vectors contains a lot of zeros, especially for short texts, which is why they belong to the family of sparse vectors (Ajose-Ismail et al., 2020). Assuming, for example, a corpus contains of 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)

For the count vectorizer the values for the vector are generated by counting for each text the frequency of the words occuring. 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

The table 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 occurence of the words, but not the frequency (Kulkarni and Shivananda, 2021; Sarkar, 2016)

While being one of the most simple techniques, 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 a lot of information of the text is losen (Sarkar, 2016). In addition, count vectorizer does not take into account 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)

4.3.2 TFIDF vectorizer

Term Frequency - Inverse Document Frequency (TF-IDF) belongs like the count vectorizer, to the family of BOW and is as well a sparse vector. In contrast to 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 which occur often in a 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 occurence of a word over all documents. By taking the inverse of DF we get the IDF. Intuitvely the IDF should quantify how distinguishable a term is. If a term is frequent over all documents it is not usful for the distinction 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 one time over all documents, thus distinguish perfectly the document from other documents, $idf(t) = log(\frac{N}{1}) = 1$. (Sidorov, 2019) Note that there are slight adjustments to calculate the IDF. (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. The implementation of sklearn normalize in addition 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 compared to count vectorizer, as a BOW model it suffers from the same limitation as count vectorizer of not taking semantic, grammatic, sequences and context into account (Sarkar, 2016).

4.3.3 Word2Vec

In contrast to the sparse techniques method mentioned above, word embedding techniques are another popular approach for vectorization. Word embedding vectors are characterised by low dimensions, dense representation and a continous 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 computionally efficient word embedding implementation. The main idea of Word2Vec is based on the distributional hypothesis, which states that similar words appear often in similar context (Sahlgren, 2008). Thus, Word2Vec learns with the help of the context representations of words, which includes the semantic meaning and the context. In such a way it words which are similar are encoded similarily (Sarkar, 2016).

There exists 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). Since in this work a pretrained model from Google is used, which is trained with CBOW, in the following the focus is on CBOW.

CBOW Word2vec is a 2-layer neural network with a hidden layer and a output softmax layer, which is visualized in figure 1. The goal is to predict a word, the so-called target word, by using the target word's context words. The number of context words is defined by a certain window size. If the window size is 2, the 2 words before and the 2 two words after the target word are considered. Given a vocabulary V, which is a the unqiue set of the words from the corpus, each context word c is fed into the neural network, encoded with one-hot encoding of the length of V, building vector x_c. Thus in figure 1 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 1. Taking the dimension of V and 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 to the number of context words, the hidden layer h is calculated as follows

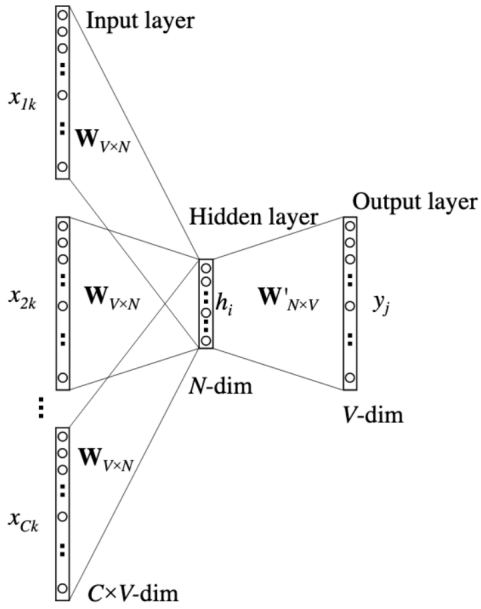

Figure 1: CBOW (Rong, 2014, 6)

(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 of the respective word value in the vector, which is 1, will be zero. Thus calculating h is just copying the k-th row of matrix W, thus a n-dimensional vector, which explains the second part of the equation. The hidden-layer matrix builds later the word embedding, which is why the decision of the size of the hidden layer defines the dimensions later 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. Caluculating the error and using backpropagation the weights are updated respectivley resulting then in a trained neural network. In practice, due to computional efficiency, instead of the softmax function Word2Vec is trained with hierarchical softmax or negative sampling. Both methods are efficient in the sense that they reduce the amount of weight matrix updates. [1]. (Rong, 2014; Simonton and Alaghband, 2017).

[1]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)

Based on the given theoretical insights, I train two Word2vec models. Both models use a pretrained model from Google and are fine tuned with different data. The first model is fine tuned with the complete dataset. The second model includes the additional knowledge. The model setting is as follows: I use CBOW Word2Vec models with 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 pretrained Google vectors, which have a size of 300. The minimal count of words is set to 1. The number of times the training data set is iterated, the epoch number, is set to 10. Lastly, the window size for the context is set to 5.

As last step the resulting word embeddings need to be processd in some way to get sentence embeddings for each job title. As the name already indicates Word2Vec cannot output sentence embeddings directly. In order to get the sentence embeddings the word vector embeddings of each job title are averaged.

4.3.4 Doc2vec

Doc2vec, also known as paragraph vectors or Distributed Memory Model of Paragraph Vectors is an extension method of Word2Vec, which outputs directly embeddings 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 bigger 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 and word vectors, like in Word2Vec, for that paragraph 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). Therby the paragraph vectors are trained as well with stochastic gradient descent and backpropagation. Similar to Word2Vec practical implementation use hierarchical softmax or negative sampling to fast up training time (Lau and Baldwin, 2016).

In figure 2 the algorithm is visualized. As an input the neural networks takes word vectors and a paragraph vector. While the word vectors are shared over all paragraphs, the paragprah vectors are unique for each paragraph. Those paragraphs are represented as unique vectors in a column of a matrix D. The word vectors are respresented in the matrix W as before. In order to predict the word both vectors are

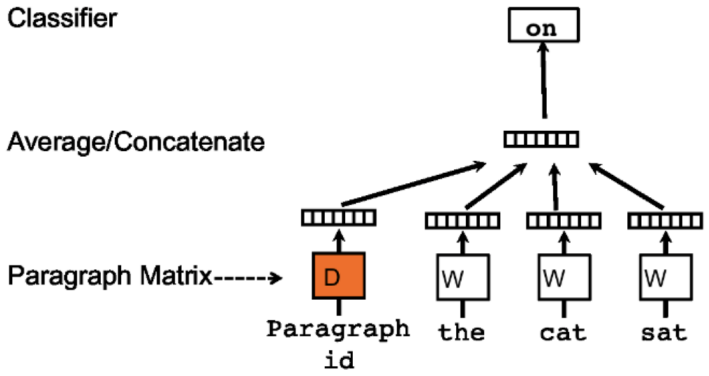

Figure 2: Doc2vec - Distributed Memory Model of Paragraph Vectors (Le and Mikolov, 2014, 3)

combined for example by averaging or concatenating. The Doc2vec implementation described by Le and Mikolov (2014) and also used in this work here concatenate 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 above in the Word2Vec part. In addition Doc2Vec takes the word order into account. At least in the same way of a large n-gram (Le and Mikolov, 2014).

Besides the model explained above, Doc2Vec comes also 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 disussion, I created two Distributed Memory Models of Paragraph Vectors. Instead of fine tuning a pretrained model, 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 10 epochs. Like Word2vec, the models are trained with negative sampling.

4.3.5 BERT

The last vectorization technique, Bidirectional Encoder Representations from Transformers (BERT), is the state-of-art language modeling technique developed by (Devlin et al., 2018) at Google. BERT stands out from other langugage models in several ways and outperforms other models for many Nature Language Processing (NLP) tasks. First, BERT uses bidirectional pretraining. Thus is does not only process from left-to-right or right-to-left, but it merge both of them. Second, it is possible to

fine tune the model for specific task without heavily-engineered and computionally costly architectures. Third compared to word2vec it is a context-dependent model. Thus, while word2vec would produce only one word embedding for "'Python" BERT can give based on the context different embeddings. Here "'Python" as a snake or as a programming language.

Architecture

BERT uses a multi-layer bidirectional Transformer encoder as the architecture. This transformer architecture was introduced by Vaswani et al. (2017). It consists of encoder and a decoder and make use of self-attention. Both, encoder and decoder stack include a number of identical layer, each of them including two sub-layers: a Multi-head attention and a feedforward network layer [2] It is out of the scope to elaborate the technical details and implementation of the attention mechanism, which is why in the following a simplified explanation of the attention mechanism 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 by relating the word to all other words, that "'it" belongs to "'dog" and not to "'food". 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 are applied in order to calculate an attention matrix Z [3]. BERT uses a multi-head-attention, which simply 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 WordPiece tokenizer and three special embedding layers, the token, the segement and the position embedding layer for each token, which is visualized in 3. A WordPiece tokenizes each word which exits in the vocabulary. If a word does not exists words are

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

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

splitas long a subword matches the vocabulary or a individual character is reached. Subwords are indicated by hashes. For the token embeddings the sentence is tokenized first, then a [CLS] is added at the beginning of the sentence and [SEP] token at the end. For example the job title ‘”java developer” and ‘” python developer”’ becomes to tokens as shown in the second row of 3. In order to distinguish between the two titles a segment embedding is added, indicating the sentences. Last 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).

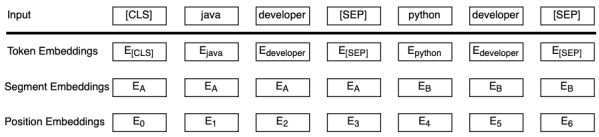

Figure 3: Input BERT (Devlin et al., 2018, 5)

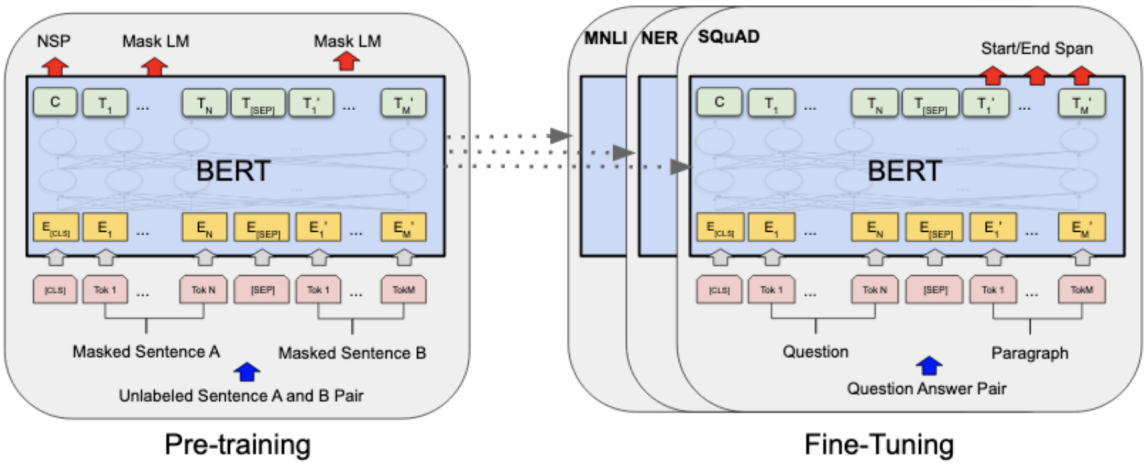

Figure 4: Overview BERT (Devlin et al., 2018, 3)

The BERT algorithm can be described in two steps. First the pretraining phase, which is illustrated on the left-hand side of the figure 4 and the fine-tuning phase, visualized on the right-hand side of figure 4. The pretraining phase consists of two jobs: Masked language modeling and next sentence prediction.

Pretraining

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 actually matched with a [mask] token, but also with random token or the real tokens [4]. 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 takes again two sentences, but predict whether the second sentence follows the first one. This helps to understand the relationship between the sentences. Each sentence pair is labelled 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 computionally expensive. Therefore, Devlin et al. (2018) initially developed different sizes of BERT like BERT-base and BERT-large. Besides those two models, there are plenty of pretrained BERT models for the German case, like BERT-base-german-cased or distilbert-base-german-cased [5]. 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 mode bert-base-german-dbmd-uncased seems to have the best result, which is why I use it for the fine tuning process. The model consists of 12 encoder layers, denoted by L, 12 attention heads, denoted by A and 768 hidden units, which results in total in 110 million parameters. It was trained with 16GB of German texts.

Fine-Tuning

The second phase, the fine-tuning, can be performed in different ways, also depending on the task. For text classifcation there are two main strategies. Either the

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

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

weights of the pretraind model are updated during the classifcation process. Or the pretrained model is first fine-tuned and then used as a feature extractor. Such it can be then in turn used for, for example, calculating similarities or as an input for classification algorithms.

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

The BERT classifier is fine tuned with the complete training data set. Practically this is done by converting the sentences of the dataset to the appropriate data format as described above and train 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 classifcation. In order to get the labels BERT uses a softmax function [6] (Sun et al., 2019). In the literature it is not well-understood so far, what exactly happens during the fine-tuning. An analysis of Merchant et al. (2020) indicates that the fine tuning process is relatively conservative in the sense that they affect only few layers and are specific for examples of a domain. Note that this analysis focussed on other nature language processing task than text classifcation. The set-up of the training is as follows: Testing different epoch numbers indicates that lower epoch size have 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 computionally efficient and practicable to implement. Thus, it faciliate to encode sentences directly into embeddings, which is why I use it for the BERT vectorizer. The model is constructed with the bert-base-german-case 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. 5 shows the architecture. The network takes pairs of sentences with a score as an input. Those scores indicate the similarity between the sentences. The network update the weights by passing the sentences through the network, calculating the cosine similarity and comparing to

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

the similarity score (Reimers and Gurevych, 2019). I create from the training data set and from the kldb taxonomy pairs of similar and dissimilar job titles or searchwords. Similar pairs are pairs from the same kldb class. As a score I choose 0.8. Dissimilar pairs are defined as pairs which are not from the same class. The score is 0.2. Building all combinations of titles and the searchwords for each class would results in a huge dataset. For example class 1 of the training data set has 2755 job titles. Thus we would have already have $\binom{2755}{2} = 3793635$ examples. Since this is computionally to expensive I randomly choose pairs of job titles. For the similar ones I choosed for job titles and for searchwords 3250 pairs. For the dissimilar pairs I choosed respectivley 1750 pairs. Thus, in total, I fine tuned the model with 10000 examples in 3 epochs.

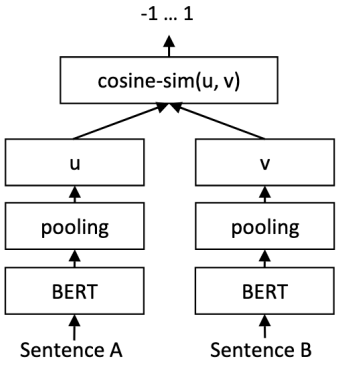

Figure 5: Sentence-BERT siamese architecture (Reimers and Gurevych, 2019, 3)

4.4 Dimensionality reduction

Dimensionality reduction techniques like Principal Component Analysis (PCA) play an important role in reducing computation time and saving computing resources (Ayesha et al., 2020). Figure 6 shows the running time of all three classifiers with different data sizes and with and without PCA dimensionality reducation.

The input of the classifiers are the word2vec word embeddings without the additional information. The bright lines show the running times without dimensionality reduction, while the dark colored lines report the running time with PCA transformation. It becomes clear that the runtime for the transformed embeddings are generally lower. While the magnitude of the differences is almost irrelevant for a data set of 500, the runtime of the non-transformed embeddings increases considerably with the size of the data set for all classifiers. This is most evident with

Random Forest (RF). Although the runtime of the transformed embeddings also increases for all classifiers, it does so at a much slower pace. Therefore, it can be concluded that the transformation clearly contributes to keeping the runtime lower for large data sets. As already described in chapter x, the training data set is fairly large, which is why it is reasonable to reduce the dimensions.

PCA, one of the most popular technique for dimensionality reduction, aims to reduce a high-dimensional feature space to a lower subspace while capturing the most important 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 in the first step the covariance matrix for the word embeddings, is obtained. The covariance matrix, denoted by $\mathbf{X}$, captures the linear relationships between the features of the word embeddings. In a next step the eigenvectors of $\mathbf{X}$ are calculated. The eigenvector of X defined as (Bro and Smilde, 2014):

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

where z is the eigenvector and λ the eigenvalue. In order decompose $\mathbf{X}$ to get the eigenvalue Singular Value Decomposition is applied. The eigenvalues are then sorted from highest to lowest and the most significant components n are kept. To transform the data a feature vector in generated. This vector contains of the most n significant eigenvalues. After transposing the mean-adjusted the word embedding and the feature vector, the embeddings can be transformed by multiplying both transposed vectors (Smith, 2002).

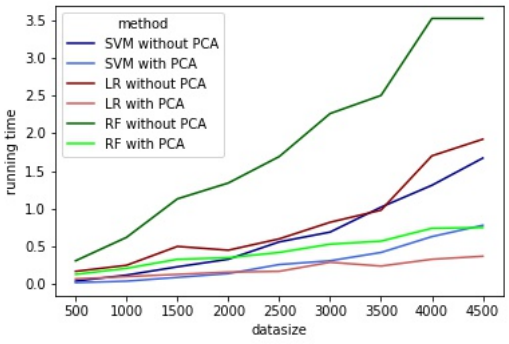

Figure 6: Running time of word2vec with different data sizes

4.5 Classifier

As pointed out in the literature review Naive Bayes (NB), Logistic Regression (LR) and Support Vector Machine (SVM) have several advantages for text classifcation tasks. In the following based on a theoretical discussion of each classifier, the exact modeling of the classifiers is justified.

4.5.1 Logistic Regression

4.5.2 Support Vector Machines

However, SVM also performed well for text classification. Especially for multiclass tasks, as mentioned in the literature review, often different versions of the algorithm are used and showed good performance (Aiolli and Sperduti, 2005; Angulo et al., 2003; Benabdeslem and Bennani, 2006; Guo and Wang, 2015; Mayoraz and Alpaydm, 1999; Tang et al., 2019; Tomar and Agarwal, 2015). In general SVM has several advantages for text classifcation. First, text classifcation usually has a high dimensional input space. SVM can handle these large features since they are able to learn independently of the dimensionality of the feature space. In addition SVMs are known to perform well for dense and sparse vectors, which is usually the case for text classification (Joachims, 1998). Empirical results, for example Joachims (1998) or Liu et al. (2010) confirm the theoretical expectations. It is, therefore, a reasonable option to use a basic version of the SVM algorithm as a baseline.

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, 2000, 138). In SVM this optimal hyperplane maximizes the margin, which is simply put the distance from the hyperplane to the closest points, so called Support Vectors, across both classes (Han et al., 2012). Formally, given a training data set 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 problem (primal) 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 optimzation function assumes that the data can be separated without errors. This is not always possible, which is why Cortes et al. (1995) introduce a soft margin SVM, which allows for missclassfication (Vapnik, 2000). 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., 2012):

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

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

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

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 to calculate the dot product in the original input space without knowing the exact mapping into the higher space (Han et al., 2012; Jordan et al., 2006).

In order to apply SVM to multiclass problems several approaches have been proposed. One stratetgy is to divide the multi-classifcation 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 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 in each classifier the data of two classes (Hsu and Lin, 2002). Besides dividing the multiclass problem into several binary problems, some researches propose approaches to solve the task in one single optimization problem, like Crammer and Singer (2001).

[7].

In order to find a strong classifier I checked SVM's with different parameters for the SVM, as well as different multiclass approaches. It appears that a SVM using a soft margin with a $C = 1$ and a one-vs-rest approach has the best results. I also test different kernels, like RBF Kernel or linear kernel. The linear kernel, formally $k(x, x') = x^T x'$, achieved the best results, which is why I choose it for the classifier.

4.5.3 Random Forest Classifier

In contrast to the previous two classifiers RF is a ensemble learning technique. The main idea of ensemble learning techniques is to create a number of learners, classifiers, and combine them. Those learners are for example descision tree or neural networks and are usually homogeneous, which means that each individual 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, the data sampling is important in sense of that it is not desirable to have same outputs for all classifiers. Thus ensemble techniques need diversity in the output, which means the outputs should be optimally independent and negatively correlated. There are well-established methods for achieving diversity. For example bagging techniques RF falls back to boostrap (Polikar, 2012). The second pillar rises 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 classifcation result and 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 as individual classfier decision trees. Before discussing RF in more detail within the three pillars described above, a brief discussion of 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 on the

[7]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)

bottom. For the root node and each of the intermediate nodes all possible splittings are checked and then are split according to the best feature. Each leaf nodes leads to one of the classification labels. Examples are then classified by traversing the tree from the top to the bottom and choosing at each intermediate node the branch which satisfy the attribute value for the example. 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 examples of the training set belong to the same class, or if there are no features left for splitting. This might end in tree with a high depth, which is why often pruning is applied to avoid overfitting of the tree Berthold et al. (2020).

There are two important points to discuss in constructing. First the types of splitting and second splitting criterion. There are mainly three types of splits: Boolean splits, nominal splits and continous splits. The latter chooses a particular value from the continous feature as the splitting value (Cutler et al., 2012; Berthold et al., 2020). For example considering a word embedding x with 300 dimension and a node t of a decision tree, which is split into a 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 0.336 at the dimension index 209 of the embedding vector are follow the branch to t_{left}, while all other examples follow the branch to t_{right}.

The splitting criterion is important to identify the best feature for splitting. Intiutively, the criterions should split the data in such a way that leaf nodes are created fast (Berthold et al., 2020). There are several measurements, so-called impurity measures to obtain 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.

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

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

p_i is th probability that a class n occurs in D. The more pure D is the lower the value of the gini value. To determine the best feature k, the dataset is partitioned based on the feature k. For continous features, as in word embeddings, this is done by continous 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 over all features K (Yuan et al., 2021, 3156):

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

.

Based on above theoretical explanations of the foundations of decision tree, researchers have developed several algorithms to train decision trees, like Iterative Dichotomiser 3, C4.5. or Classification and Regression Trees (CART), which is used in RF. CART produces depending on the target variable classification (for categorical variables) or regression trees (for numerical variables). It constructs only binary trees, thus each split is into two nodes. The algorithm uses as impurity measurement 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 train a number of independent classifiers. The sampling technique is sampling with replacement (bootstrapping). Bagging combines the individual models by using either majority voting or averaging. RF differentiates from the classic bagging method in the way that it also allows to choose a subset of features for each classifier from which the classifiers can select instead of allowing them to select from the complete 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 distributed 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 which is created for each k-th tree. It is important, that Θ_k

is independent of the vectors $\Theta_1 ... \Theta_{k-1}$, thus from all random vectors of the previous classifiers. Although the distribution of the random vectors remain. Combined with the training set, with $\mathbf{x}$ as the input vector a classifier $h(\mathbf{x}, \Theta_k)$ is constructed (Breiman, 2001). In practical implementation the random component Θ_k is not explicitly used. Instead it is rather used implicitly to generate two random strategies (Cutler et al., 2012). The first strategy is the bootstrapping. Thus drawing sample with replacement from the training data set. In order to estimating generalization error, correlation and variable importance Breiman (2001) applied out-of-bag estimation. Out-of-bag estimation leave out some portion of the training data in each bootstrap. The second strategy is to choose random feature for the splitting. Thus at each node from the set of features only a subset is used to split. While decision trees are often pruned to avoid overfitting, RF does without. The trees grow by applying CART-algorithm. RF uses as combination method for classifcation unweighted voting (Cutler et al., 2012).

Based on the above explanations the implemented RF has the following setting: The numbers of learners is 100. Gini is used as the splitting criterion. The maximal number of features is $\sqrt{\text{number of features}}$. Note that sklearn, which is used to implement RF here, uses an optimised algorithm of CART. (Pedregosa et al., 2011).

5 Result

5.1 Evaluation metrics

There exists several metrics for the evaluation of classification approaches in the literature (Fatourechi et al., 2008). The choice of appropriate measurements is a crucial step for obtaining a qualitative comparison in the performance between the baseline algorithms and the new approaches. Often researchers rely on popular metrics like overall accuracy (OA). However, especially for multiclass and inbalanced dataset tasks it is difficult to rely only on one measure like OA In order to select appropriate metrics for comparison in the following the most important metrics will be discussed focussing on multiclass classification and imbalanced data sets.

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

From the confusion matrix follows that $c_{i,j}$ defines examples which belong to class

j and are predicted as class i. Given that k is the current class, True positives (TP) is defined as $tp_k = c_{k,k}$, thus examples which are correctly predicted as the current class k. False negatives (FN) are defined as those examples which not belonging to the current class k, but are predicted as k. Formally $fn_k = \sum_{i=1,i\neq k}^{n} c_{i,k}$. Next, True negatives (TN), are examples belonging to the current class m, but are not predicted as m. Formally $tn_k = \sum_{i=1,i\neq k}^{n} \sum_{j=1,j\neq k}^{n} c_{i,j}$. Last, False positives (FP) are defined as examples not belonging to class k, but are predicted as such. Formally this can be expressed as: $fp_k = \sum_{i=1,i\neq k}^{n} c_{k,i}$ (Kautz et al., 2017)

As mentioned the OA is one of most common metric for performance evaluation. It represents how well the classifier classifies across all classes correctly. Formally, given that N is number of examples and K the number of all classes, this can be expressed as (Branco et al., 2017):

$$OA = \frac{1}{K} \sum_{i=1}^{K} \frac{tp_k + tn_k}{N}$$

Following the formula an accuracy of 1 means that all examples are correctly classified, while a 0 mean that each example is classified with the wrong class. (Berthold et al., 2020) Although OA is a widley used metric it is critized for favouring the majority classes, thus not reflecting minority classes appropriatly in unbalanced datasets (Berthold et al., 2020; Fatourechi et al., 2008)

Two more popular metrics are precision and recall. Precision respresents how well the classifier detects actual positive examples among the positive predicted examples. Recall, also called sensitivity, in contrast, represents how many examples are labelled as positive among the actual positive examples (Berthold et al., 2020). For the multiclass scenario, two different calculation approaches for each of the metrics are proposed: micro and macro average (Branco et al., 2017). In the macro approach first the metric is calculated for each class k against all other classes. The average of all of them is built. Formally:

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

Table 5: Confusion Matrix (edited after (Kautz et al., 2017, 113)

$$precision_{macro} = \frac{1}{K} \sum_{i=1}^{k} \frac{tp_i}{tp_i + fp_i}$$

$$recall_{macro} = \frac{1}{K} \sum_{i=1}^{k} \frac{tp_i}{tp_i + fn_i}$$

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

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

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

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

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

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

Apart from the trade-off between recall and precision, there is also a tradeoff between sensitivity and specificty (1- sensitivity). Using a receiver operating characteristics (ROC), which plots the specifity against the sensitivity the trade-off can be visualized for different thresholds. The area under the curve then can be used to obtain the performance of the classifier. A large area indicates a better classifier (Berthold et al., 2020; Espíndola and Ebecken, 2005).

As shown above, there are several metrics for evaluating the performance of a classifier, with the metrics having different focuses. Since the job title classification involves multiclass classification and the descriptive analysis show that the data is clearly unbalanced, at least for some classes in level 5, it is not reasonable to base the evaluation solely on the OA. Taking precision, recall and the harmonic mean into account would capture the performance of the minority classes as well. The ROC curve does gives, due to its visualization a good impression for the performance, but it is not feasible for high number of classes. Following this argumentation the

	LR	SVM	RF
CountVectorizer	0.71	0.68	0.64
TFIDF	0.71	0.69	0.64
Word2Vec	0.55	0.53	0.62
Doc2Vec	0.48	0.46	0.56
BERT	0.65	0.65	0.58

Table 6: Evaluation of Level 1 classification - Accuracy

	LR	SVM	RF
CountVectorizer	p: 0.74, r: 0.60, F1: 0.64	p: 0.73, r: 0.56, F1: 0.60	p: 0.67, r: 0.54, F1: 0.57
TFIDF	p: 0.75, r: 0.60, F1: 0.63	p: 0.74, r: 0.57, F1: 0.62	p: 0.65, r: 0.53, F1: 0.55
Word2Vec	p: 0.52, r: 0.40, F1: 0.42	p: 0.46, r: 0.41, F1: 0.41	p: 0.62, r: 0.54, F1: 0.56
Doc2Vec	p: 0.43, r: 0.34, F1: 0.35	p: 0.39, r: 0.33, F1: 0.33	p: 0.60, r: 0.41, F1: 0.43
BERT	p: 0.67, r: 0.57, F1: 0.60	p: 0.63, r: 0.56, F1: 0.58	p: 0.70, r: 0.46, F1: 0.50

Table 7: Evaluation of Level 1 classification - Macro

performance of the classifiers will be evaluated with accuracy, precision, recall, F-measure and Cohen's Kappa.

5.2 Experimental results

6 Conclusion and Limitations

7 sentences

Ajose-Ismail et al. (2020) concludes that for there are different results for different classifiers.

Singh (2022) The results of doc2vec and word2vec seem in the first place counterintuive. But short text high dimensional space. Also other papers like Singh (2022) show similar results for short texts.

	LR	SVM	RF
CountVectorizer	0.48	0.50	0.43
TFIDF	0.47	0.51	0.43
Word2Vec	0.28	0.14	0.35
Doc2Vec	0.19	0.20	0.31

Table 8: Evaluation of Level 3 classification - Accuracy

	LR	SVM	RF
CountVectorizer	p: 0.42, r: 0.27, F1: 0.3	p: 0.41, r: 0.35, F1: 0.36	p: 0.37, r: 0.25, F1: 0.28
TFIDF	p: 0.38, r: 0.23, F1: 0.26	p: 0.41, r: 0.35, F1: 0.36	p: 0.36, r: 0.25, F1: 0.27
Word2Vec	p: 0.15, r: 0.11, F1: 0.12	p: 0.07, r: 0.05, F1: 0.04	p: 0.23, r: 0.18, F1: 0.19
Doc2Vec	p: 0.1, r: 0.05, F1: 0.05	p: 0.13, r: 0.08, F1: 0.09	p: 0.23, r: 0.13, F1: 0.14

Table 9: Evaluation of Level 3 classification - Macro

	LR	SVM	RF
CountVectorizer	0.54	0.52	0.47
TFIDF	0.53	0.54	0.49
Word2Vec	0.34	0.28	0.39
Doc2Vec	0.24	0.28	0.36
BERT	0.51	0.49	0.45

	LR	SVM	RF
CountVectorizer	p: 0.63, r: 0.49, F1: 0.53	p: 0.57, r: 0.48, F1: 0.51	p: 0.55, r: 0.42, F1: 0.46
TFIDF	p: 0.66, r: 0.47, F1: 0.52	p: 0.59, r: 0.49, F1: 0.51	p: 0.58, r: 0.44, F1: 0.48
Word2Vec	p: 0.38, r: 0.26, F1: 0.28	p: 0.31, r: 0.23, F1: 0.24	p: 0.39, r: 0.34, F1: 0.35
Doc2Vec	p: 0.36, r: 0.14, F1: 0.15	p: 0.33, r: 0.23, F1: 0.25	p: 0.42, r: 0.29, F1: 0.31
BERT	p: 0.53, r: 0.44, F1: 0.46	p: 0.44, r: 0.45, F1: 0.44	p: 0.56, r: 0.38, F1: 0.42

	LR	SVM	RF
CountVectorizer	0.63	0.65	0.57
TFIDF	0.61	0.65	0.59
Word2Vec	0.44	0.38	0.49
Doc2Vec	0.27	0.35	0.45
BERT	0.62	0.63	0.55

	LR	SVM	RF
CountVectorizer	p: 0.71, r: 0.59, F1: 0.63	p: 0.67, r: 0.62, F1: 0.63	p: 0.63, r: 0.55, F1: 0.57
TFIDF	p: 0.72, r: 0.57, F1: 0.61	p: 0.67, r: 0.63, F1: 0.63	p: 0.63, r: 0.56, F1: 0.58
Word2Vec	p: 0.53, r: 0.38, F1: 0.42	p: 0.38, r: 0.32, F1: 0.31	p: 0.49, r: 0.46, F1: 0.46
Doc2Vec	p: 0.41, r: 0.15, F1: 0.17	p: 0.36, r: 0.3, F1: 0.31	p: 0.53, r: 0.37, F1: 0.41
BERT	p: 0.66, r: 0.58, F1: 0.60	p: 0.61, r: 0.62, F1: 0.60	p: 0.64, r: 0.49, F1: 0.53

	Accuracy	Precision	Recall	F1
BERT CLF Level 1	0.76	0.73	0.70	0.71
v BERT CLF Level 3	0.53	0.56	0.46	0.46
BERT CLF Level 5	0.60	0.59	0.54	0.53

References

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

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.

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

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.

Benabdeslem, K. and Bennani, Y. (2006). Dendogram based svm for multi-class classification. *Proceedings of the International Conference on Information Technology Interfaces, ITI*, pages 173–178.

Berthold, M. R., Borgelt, C., Höppner, F., Klawonn, F., and Silipo, R. (2020). *Guide to Intelligent Data Science*. Springer, 2 edition.

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

Bouaziz, A., Dartigues-Pallez, C., da Costa Pereira, C., Precioso, F., and Lloret, P. (2014). Short text classification using semantic random forest. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8646 LNCS:288–299.

Branco, P., Torgo, L., and Ribeiro, R. P. (2017). Relevance-based evaluation metrics for multi-class imbalanced domains. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10234:698–710.

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.

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.

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.

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

Colas, F. and Brazdil, P. (2006). Comparison of svm and some older classification algorithms in text classification tasks. *IFIP International Federation for Information Processing*, 217:169–178.

Cortes, C., Vapnik, V., and Saitta, L. (1995). Support-vector networks editor. *Machine Leaming*, 20:273–297.

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

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

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*.

Espíndola, R. P. and Ebecken, N. F. F. (2005). On extending f-measure and g-mean metrics to multi-class problems. *WIT Transactions on Information and Communication Technologies*, 35:25–34.

Fatourechi, M., Ward, R. K., Mason, S. G., Huggins, J., Schlögl, A., and Birch, G. E. (2008). Comparison of evaluation metrics in classification applications with imbalanced datasets. *Proceedings - 7th International Conference on Machine Learning and Applications, ICMLA 2008*, pages 777–782.

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.

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

Han, J., Kamber, M., and Pei, J. (2012). *Data Mining: Concepts and Techniques*. Elsevier Inc., 3 edition.

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

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. *European Conference on Machine Learning*, pages 137–142.

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

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*, 2:1916–1919.

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

Kowsari, K., Meimandi, K. J., Heidarysafa, M., Mendu, S., Barnes, L., and Brown, D. (2019). Text classification algorithms: A survey. *Information 2019*, 10:1–68.

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, 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*.

Liu, Z., Lv, X., Liu, K., and Shi, S. (2010). Study on svm compared with the other text classification methods. *2nd International Workshop on Education Technology and Computer Science, ETCS 2010*, 1:219–222.

Manning, C., Raghavan, P., and Schütze, H. (2008). *Introduction to information retrieval*. Cambridge University Press.

Mayoraz, E. and Alpaydm, E. (1999). Support vector machines for multi-class classification. *Lecture Notes in Computer Science*, 1607:833–842.

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

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

Pan, W., Narasimhan, H., Protopapas, P., Kar, P., and Ramaswamy, H. G. (2016). Optimizing the multiclass f-measure via biconcave programming. *IEEE 16th International Conference on Data Mining (ICDM)*, pages 1101–1106.

Paulus, W. and Matthes, B. (2013). Klassifikation der berufe : Struktur, codierung und umsteigeschlüssel. *FDZ Methodenreport*.

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.

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

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*.

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.

Schneider, K.-M. (2005). Techniques for improving the performance of naive bayes for text classification. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 682–693. Springer.

Sebastiani, F. (2001). Machine learning in automated text categorization. *ACM Computing Surveys*, 34:1–47.

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, L. (2022). Clustering text: A comparison between available text vectorization techniques. In *Soft Computing and Signal Processing*, pages 21–27. Springer.

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*, 9:635–643.

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

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. (2019). 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.

Vapnik, V. N. (2000). *The nature of statistical learning theory*. Springer.

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. R., and Parameswaran, L. (2017). A comprehensive study of text classification algorithms. pages 1109–1113. Institute of Electrical and Electronics Engineers Inc.

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

Wang, Y., Zhou, Z., Jin, S., Liu, D., and Lu, M. (2017). Comparisons and selections of features and classifiers for short text classification. *IOP Conference Series: Materials Science and Engineering*, 261:1–8.

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.

Xu, S. (2018). Bayesian naïve bayes classifiers to text classification:. *Journal of Information Science*, 44:48–59.

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

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

A Naive Bayes Classifier

A.1 Theory

The NB, a family of probabilistic classifiers, uses Bayes' rule in order to determine the most likely class for each document (Schneider, 2005). All NB classifiers rely

on the conditional independence assumptions which means, that "features are independent of each other, given the category variable" (Xu, 2018, 48). Depending on whether the features are discrete or continous, different distributions, so-called event models are proposed. While from a theoretical perspective for continous features Gaussian distribution is well-suited, for discrete features usually Bernoulli or multinomial distibutions are applied (Xu, 2018). Although, popular practical implementations, like the one from sklearn, allow as well fractional counts for multinomial distributions (Pedregosa et al., 2011). Trying different event models, the multinomial NB shows indeed for both Count Vectorizer and for TFIDF the best results, which is why I choose it as event model for the baseline.

The multinomial NB classifies according to the most likely class. Given that a document d has $t = 1, ..., k$ terms and can be assigned to $c = 1, ..., j$ classes, the probability of a term in a document given a class is calculated as (Manning et al., 2008):

$$P(t_k, c_j) = \frac{occurence(t_k, c_j) + 1}{\sum occurence(t, c_j) + |V|}$$

where $|V|$ is the cardinality of the vocabulary. In the denominator 1 is added, so-called Laplace smoothing, in order to avoid zeros, which is the case if the number of terms in a document for one class is zero. (Manning et al., 2008). Further given that $N_c = count(c_j)$ is the number of documents belonging to class c_j and N is number of documents the probability of c_j is defined as $\frac{N_c}{N}$. The probability of a document d belonging to a class c_j can then formulated as follows (Manning et al., 2008, 258):

$$P(c_j|d) \propto P(c_j) \prod_{i=1}^{k} P(t_i|c_j)$$

Then the most likely classes can be determined by (Manning et al., 2008):

$$\arg\max_{c \in C} P(c_j) \prod_{i=1}^{k} P(t_i|c_j)$$

A.2 Results