Multiclass Text Classification on Unbalanced, Sparse and Noisy Data

Tillmann Dönicke[1], **Florian Lux**[2], and **Matthias Damaschk**[3]

University of Stuttgart
Institute for Natural Language Processing
Pfaffenwaldring 5 b
D-70569 Stuttgart
{doenictn[1],luxfn[2],damascms[3]}@ims.uni-stuttgart.de

Abstract

This paper discusses methods to improve the performance of text classification on data that is difficult to classify due to a large number of unbalanced classes with noisy examples. A variety of features are tested, in combination with three different neural-network-based methods with increasing complexity. The classifiers are applied to a songtext–artist dataset which is large, unbalanced and noisy. We come to the conclusion that substantial improvement can be obtained by removing unbalancedness and sparsity from the data. This fulfils a classification task unsatisfactorily—however, with contemporary methods, it is a practical step towards fairly satisfactory results.

1 Introduction

Text classification tasks are omnipresent in natural language processing (NLP). Various classifiers may perform better or worse, depending on the data they are given (eg. Uysal and Gunal, 2014). However, there is data where one would expect to find a correlation between (vectorised) texts and classes, but the expectation is not met and the classifiers achieve poor results. One example for such data are songtexts with the corresponding artists being classes. A classification task on this data is especially hard due to multiple handicaps:

First, the number of classes is extraordinarily high (compared to usual text classification tasks). Second, the number of samples for a class varies between a handful and more than a hundred. And third, songtexts are structurally and stylistically more diverse than, e.g., newspaper texts, as they may be organised in blocks of choruses and verses, exhibit rhyme, make use of slang language etc. (cf. Mayer et al., 2008). In addition, we try to predict something latent, since there is no direct mapping between artists and their songtexts. A lot of the texts are not written by the singers themselves, but by professional songwriters (Smith et al., 2019, p. 5). Hence the correlation that a classifier should capture is between songtexts that the writers think to fit a specific artist and the artist. All these points make the task difficult; still, it is a task needed for nowadays' NLP systems, e.g., in a framework that suggests new artists to a listener based on the songtexts s/he likes. Thus, to tackle the challenges given is a helpful step for any task in the field of NLP that might come with similarly difficult data.

For the artist classification, we investigate three neural-network-based methods: a one-layer perceptron, a two-layer Doc2Vec model and a multilayer perceptron. (A detailed description of the models shall follow in section 2.) Besides the model, the representation of the instances in a feature space is important for classification, thus we also aim to find expressive features for the particular domain of songtexts. (See section 4 for a list of our features.)[1]

2 Methods

The following sections shall provide an overview of our classifiers in order of increasing complexity.

2.1 Perceptron

A perceptron is a very simple type of neural network that was first described in Rosenblatt (1958). It contains only one layer, which is at the same time the input and output layer. The input is a feature vector $\vec{x}$ extracted from a data sample x. Every possible class $y \in Y$ is associated with a weight vector $\vec{w}_y$. A given sample x is classified as $\hat{y}_x$ as follows:

[1]The code will be made available at https://github.com/ebaharilikult/DL4NLP.

Proceedings of the First NLPL Workshop on Deep Learning for Natural Language Processing, pages 58–65

$$\hat{y}_x = \arg\max_{y \in Y} \vec{x} \cdot \vec{w}_y \quad (1)$$

During training, every training sample is classified using the perceptron. If the classification is incorrect, the weights of the incorrectly predicted class are decreased, whereas the weights of the class that would have been correct are reinforced.

Additions to this basic system include shuffling of the training data, batch learning and a dynamic learning rate. The shuffling of the training data across multiple epochs shall prevent the order of the samples from having an effect on the learning process. Batch learning generally improves the performance of a perceptron (e.g. McDonald et al., 2010); hereby all updates are jointly performed after each epoch instead of each sample. This removes a strong preference for the last seen class if the information in the features of multiple samples overlaps greatly. A dynamic learning rate improves the convergence of the weights. It gives updates that happen later during training less impact and allows a closer approximation to optimal weights.

2.2 Doc2Vec

Doc2Vec (Le and Mikolov, 2014) is a two-layer neural network model which learns vector representations of documents, so-called paragraph vectors. It is an extension of Word2Vec (Mikolov et al., 2013) which learns vector representations of words. Thereby, context words, represented as concatenated one-hot vectors, serve as input and are used to predict one target word. After training, the weight matrix of the hidden layer becomes the word matrix, containing the well-known Word2Vec word embeddings.

The extension in Doc2Vec is that a unique document/paragraph id is appended to each input n-gram, as if it was a context word. Since paragraph ids have to be different from all words in the vocabulary, the weight matrix can be separated into the word matrix and the paragraph matrix, the latter containing document embeddings.[2]

In a document classification task, labels instead of paragraph ids are used. By doing so, the Doc2Vec model learns vectors of each label instead of each document. If one wants to predict the label of an unseen document, a vector representation for this document needs to be inferred first. Therefore, a new column/row is added to the paragraph matrix. Then the n-grams of the document are iteratively fed to the network (as in training). However, the word matrix as well as the weight matrix of the output layer are kept fixed, and so only the paragraph matrix is updated. The resulting paragraph vector for the unseen document is finally compared to the paragraph vectors representing labels; and the label of the most similar vector is returned as the prediction.

2.3 MLP

A multi-layer perceptron (Beale and Jackson, 1990), also referred to as feed forward neural network, is a deep neural network consisting of multiple hidden layers with neurons which are fully connected with the neurons of the next layer, and an output layer with as many neurons as classes. The number of layers and the number of neurons in each hidden layer depends on the classification tasks and are therefore hyperparameters. For multiclass classification, the softmax function is used in the output layer. During training, the backpropagation learning algorithm (Rumelhart et al., 1985) based on the gradient descent algorithm, updates the weights and reduces the error of the chosen cost function, such as mean squared error or cross-entropy.

To prevent overfitting in neural networks, dropout (Srivastava et al., 2014) is commonly used. It omits hidden neurons with a certain probability to generalise more during training and thus enhances the model.[3]

3 Data

We use a dataset of 57,647 English songtexts with their corresponding artist and title, which has been downloaded from Kaggle[4]. The data was shuffled uniformly and 10% were held out for validation and test set, respectively.

There are 643 unique artists in the data. Table 1 shows the distribution of artists and songtexts for

[2]The described version of Word2Vec and Doc2Vec is commonly referred to as "continuous bag of words" (DBOW) model. If the input and output are swapped, i.e. a single word (Word2Vec) or document (Doc2Vec) is used to predict several context words, the architecture is called "skip-gram" (SG) or "distributed memory" (DM) model, respectively.

[3]It should be noted here that advanced deep learning models such as CNNs and RNNs exist and have been successfully used in text classification tasks (Lee and Dernoncourt, 2016), but have not been used in the context of this work and are therefore not explained in detail.

[4]`https://www.kaggle.com/mousehead/songlyrics`

Dataset	Artists	Songs	Avg. songs
Training	642	46,120	71.8 (44.0)
Validation	612	5,765	9.4 (5.7)
Test	618	5,765	9.3 (6.0)

Table 1: Number of unique artists (classes), number of songtexts and average number of songtexts per artist (standard deviation in parentheses) for each dataset.

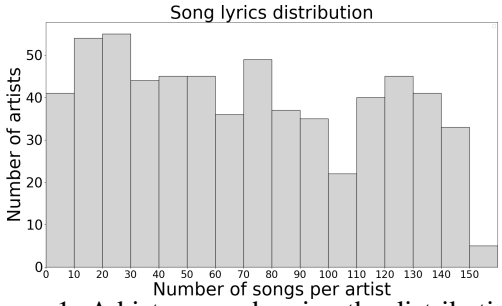


Figure 1: A histogram showing the distribution of songtexts in the training set.

each subset. The training set contains most of the unique artists (642) whereas less artists appear in the validation (612) and test set (618). Also, the standard deviation of average songs per artists is relatively high (i.e. more than half the average) which indicates that the number of songs per artists is spread over a large range.

The distribution of songs per artists in the training set can be seen in Figure 1. It shows similar counts for classes (artists) with many and classes with only a few samples (songtexts), i.e. unbalanced training data. The bandwidth goes from one artist with 159 songs to four artists with only one song which also illustrates the sparsity for some classes.

Besides the issues caused by the distribution of songtexts per artist, the quality of the texts is less than perfect. Nonsense words and sequences, such as *tu ru tu tu tu tu*, as well as spelling variations, such as *Yeaaah*, *Hey-A-Hey* and *aaaaaaaallllllright!*, are very common.

4 Feature Extraction

For both the (single-layer) perceptron and the multi-layer perceptron we use the same preprocessing and features which are described below. (The Doc2Vec model uses its own integrated tokeniser and Word2Vec-based features.)

The songtexts are tokenised by whitespace. Within a token, all non-initial letters are lowercased and sequences of repeating letters are shortened to a maximum of three. To further reduce noise, punctuation is removed. The texts are tagged with parts-of-speech (POS) using the Apache OpenNLP maximum entropy tagger[5].

Stylometric features Generic information about the text, i.e. the number of lines, the number of tokens, the number of tokens that appear only once, the number of types and the average number of tokens per line.

Rhyme feature Number of unique line endings (in terms of the last two letters), normalised by the number of lines. This should serve as a simple measure for how well the lines rhyme.

Word count vectors Every unique word in the training corpus is assigned a unique dimension. In each dimension, the number of occurrences of the word in the sample are denoted. Term-frequency (tf) weighting is implemented, but can be switched on and off. As a minimal variant, only nouns can be taken into account; in this case we speak of noun count vectors.

POS count vectors The same as word count vectors after replacing all words with their POS tag.

Word2Vec embeddings 300-dimensional embeddings created from the Google news corpus.[6] The embedding of a text is hereby defined as the average of the embeddings of all the words in the text. As a minimal variant, only nouns can be taken into account.

Bias A feature with a constant value of 1 (to avoid zero vectors as feature vectors).

5 Experiments

This section lists the concrete parameter settings of the methods described in section 2. Since our models can only predict classes which have been encountered during training, only the 612 artists occurring in all subsets are kept for all evaluations. For another series of experiments, only the 40 unique artists with more songs than 140 in the training set are kept to reduce the impact of unbalancedness and sparsity (numbers in Table 2).

[5] `https://opennlp.apache.org/docs/1.8.0/manual/opennlp.html\#tools.postagger`

[6] GoogleNews-vectors-negative300.bin.gz from `https://code.google.com/archive/p/word2vec/`

Dataset	Artists	Songs	Avg. songs
Training	40	5,847	146.2 (4.6)
Validation	40	646	16.2 (3.6)
Test	40	714	17.9 (5.0)

Table 2: Number of unique artists (classes), number of songtexts and average number of songtexts per artist (standard deviation in parentheses) for each dataset, keeping only the 40 unique artists with more songs than 140 in the training set.

5.1 Experimental Settings

Perceptron We train and test two versions of the perceptron. The minimal version (Perceptron) only uses noun count vectors without tf-weighting and the bias. The maximal version (Perceptron+) uses all features. All add-ons described in section 2.1 (shuffling, batch learning, dynamic learning rate) are used for both versions since they led to an increasing performance on the validation set in preliminary tests. For the decay of the learning rate, a linear decrease starting from 1 and ending at $\frac{1}{\text{number of epochs}}$ is chosen.

Doc2Vec For the Doc2Vec implementation, we use deeplearning4j[7] version 0.9. The hyperparamters are a minimum word frequency of 10, a hidden layer size of 300, a maximum window size of 8, and a learning rate starting at 0.025 and decreasing to 0.001. Tokenisation is performed by the incorporated UIMA tokeniser. The model is trained for 100 epochs with batch learning.[8]

MLP The implementation of MLP and MLP+ is done with Keras (Chollet et al., 2015). MLP+ uses all feature groups from section 4 and one bias feature for every group. The MLP+ model is shown in Figure 2. Each feature group uses multiple stacked layers that are then merged with a concatenation layer. The sizes of the dense layers are manually selected by trial and error. Several dropout layers with a constant probability of 0.2 are included. In contrast, MLP uses only the noun count vectors (as Perceptron) and thus only one input branch.

For both models, Adadelta, an optimisation function with adaptive learning rate, a batch size of 32 and categorical cross-entropy as the loss function are used. For the activation functions, rectified linear units are used in the hidden layer and softmax in the output layer. The model trains for 250 epochs and stores the weights that led to the best accuracy on the validation set through a checkpoint mechanism.

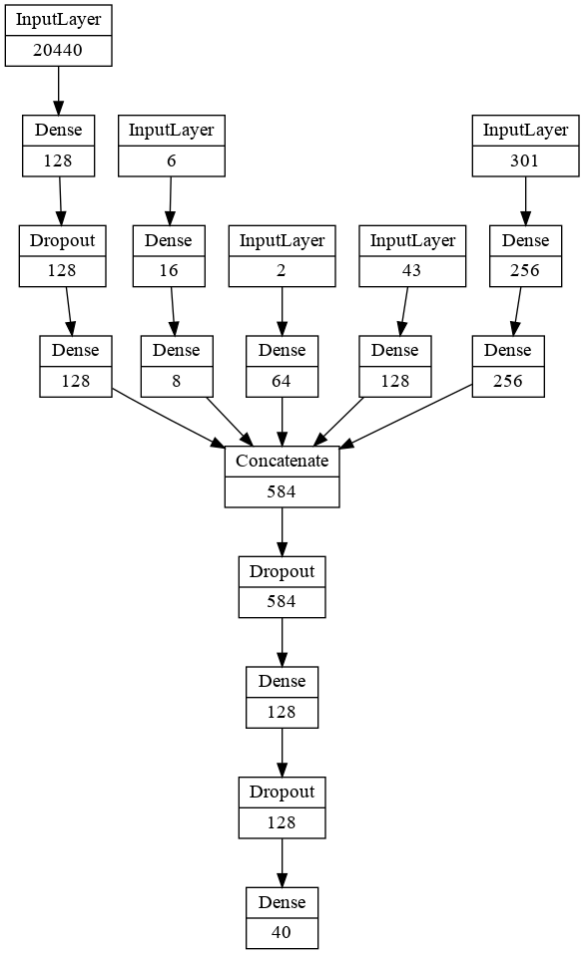


Figure 2: Model of the MLP+ model: Layers with corresponding number of neurons. The input layers correspond to the following feature groups (f.l.t.r.): word count vectors, stylometric features, rhyme feature, POS count vectors, Word2Vec embeddings.

5.2 Evaluation measures

Given a (test) set X, each sample $x \in X$ has a class y_x and a prediction $\hat{y}_x$. Based on the predictions, we can calculate class-wise precision (P), recall (R) and F-score as follows:

$$P(y) = \frac{|\{x \in X \mid y_x = y \wedge y_x = \hat{y}_x\}|}{|\{x \in X \mid \hat{y}_x = y\}|} \quad (2)$$

[7] See `https://deeplearning4j.org/docs/latest/deeplearning4j-nlp-doc2vec` for a quick example.

[8] Since deeplearning4j does not document the possibility to get intermediate evaluation results during training, 10 models are trained separately for 10, 20, 30 etc. epochs to obtain data points for the learning progress analysis.

$$R(y) = \frac{|\{x \in X \mid y_x = y \wedge y_x = \hat{y}_x\}|}{|\{x \in X \mid y_x = y\}|} \quad (3)$$

$$F(y) = \frac{2 \cdot P(y) \cdot R(y)}{P(y) + R(y)} \quad (4)$$

The macro-averaged F-score of all classes is the average of the class-wise F-scores:

$$F_{macro} = \frac{1}{|Y|} \cdot \sum_{y \in Y} F(y) \quad (5)$$

For the overall precision and recall, the nominators and denominators are summed up for all $y \in Y$, resulting in:

$$P = R = \frac{|\{x \in X \mid y_x = \hat{y}_x\}|}{|\{x \in X\}|} \quad (6)$$

And the formula for the micro-averaged F-score:

$$F_{micro} = \frac{2 \cdot P \cdot R}{P + R} = P = R \quad (7)$$

The identity of overall P and R causes their identity with F_{micro}. This measure is identical with the overall accuracy (correct predictions divided by all predictions). Since F_{macro} gives every class the same weight, but we deal with an unbalanced dataset, we choose F_{micro} as evaluation measure and only show F_{macro} in some graphs for comparison.

5.3 Results

Table 3 shows the micro-averaged F-score for all models. MLP+ performs best, followed by Perceptron and Doc2Vec. The use of additional features significantly decreases the performance of the perceptron (Perceptron+), but increases it for the multi-layer network (MLP+). This observation is discussed in section 5.4.

Figures 3–5 show the performance of Perceptron, Doc2Vec and MLP+ in dependence of the number of training epochs. The Perceptron (Figure 4) shows a generally increasing learning curve, i.e. more epochs lead to better results. Peaks like the one after the 51st epoch are ignored since the model uses a fixed number of 100 training epochs. Doc2Vec (Figure 5) reaches its best performance with 20 epochs and does not show any learning progress after that, even if trained for 100 epochs. The MLP+ model (Figure 3) exhibits increasing performance until around 100 training epochs and then reaches a plateau. Since the model uses that number of epochs which works best on the validation set (i.e. 105), it misses the best performance at the 109th epoch but gets a final score close to it.

Model	MST	final		best	
		F	ep.	F	ep.
Perceptron	0	.066	100	.069	94
Perceptron+	0	.003	100	.006	23
Doc2Vec	0	.032	100	.033	70
MLP	0	.023	97	.025	114
MLP+	0	**.079**	97	.089	80
Perceptron	140	.146	100	.160	51
Perceptron+	140	.021	100	.055	54
Doc2Vec	140	.101	100	.120	20
MLP	140	.050	201	.088	46
MLP+	140	**.182**	105	.193	109

Table 3: Micro-averaged F-score for each model on the test set after training (final) and during training (best), together with the corresponding training epochs. Lower part: only artists with more songs than (MST) 140 are kept in the training and the test set.

5.4 Error Analysis

In this section, we shall focus on the confusion matrices produced by our three main models which are depicted in Figure 6 and to be read as follows: The x-axis and the y-axis represent artists in the same order (labels are omitted due to legibility). Each cell indicates how often the artist on the x-axis was classified as the artist on the y-axis (darker colours are higher numbers). The cells on the main diagonal correspond to the cases where a class was classified correctly hence a visible diagonal correlates with good results. Darker cells outside of the diagonal are significant misclassifications[9] and might be interesting to look into.

As is clearly visible in Figure 6 (a), something is wrong with the predictions of the Doc2Vec model. There are hints of a diagonal showing at the top-left—however, there are entire columns of dark colour. This means that there are classes that are almost always predicted, no matter which class a sample actually belongs to. Interestingly, we ob-

[9] Such mis-classifications will be denoted as outliers in the following, since we are talking about the correlation of the axes here.

Figure 3: Micro-averaged (above) and macro-averaged (below) F-score of the MLP+ model on the test set (MST=140) after each training epoch.

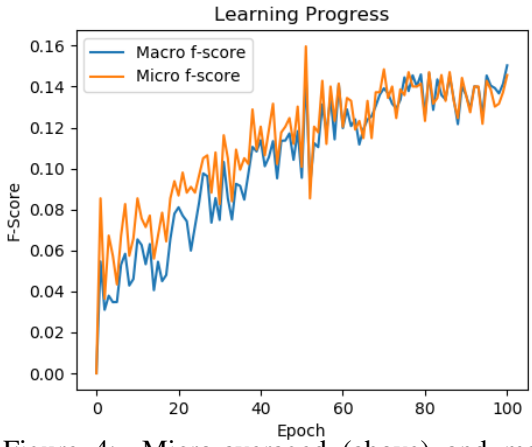


Figure 4: Micro-averaged (above) and macro-averaged (below) F-score of the Perceptron model on the test set (MST=140) after each training epoch.

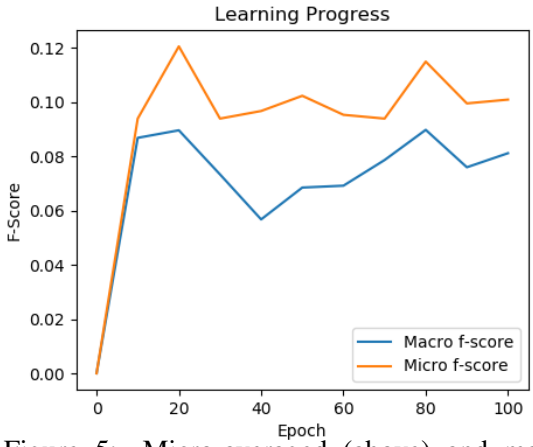
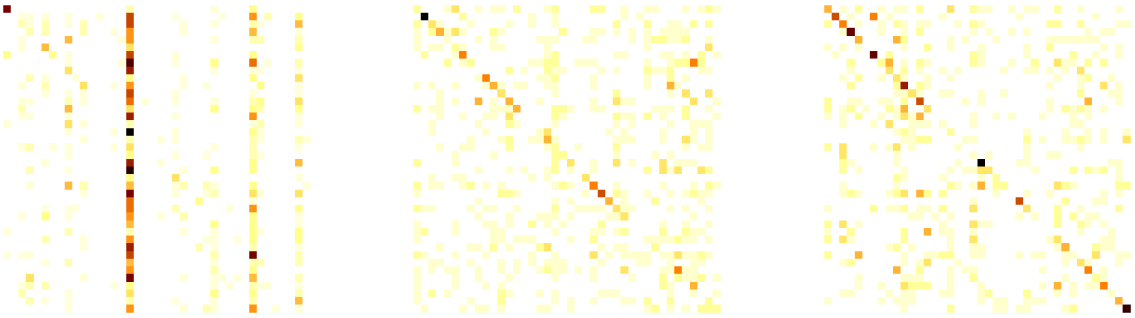


Figure 5: Micro-averaged (above) and macro-averaged (below) F-score of the Doc2Vec model on the test set (MST=140) for different numbers of training epochs.

served the same behaviour with our perceptron implementation in preliminary tests which changed when we started using batch learning. However, our Doc2Vec implementation already uses batch learning and mini-batch learning did not improve the performance in postliminary tests.

Figure 6 (b) shows that the perceptron performs well on most classes. However, there are very few classes where it actually recognises (almost) all of the samples. There are three major outliers and many outliers overall. This explains the rather low scores the model achieves, even though the diagonal is clearly visible. Looking at the three big outliers and engineering new features specifically for those could improve the performance. However, the informativeness of features can behave differently from what one might expect. Additional stylometric features that were specifically designed for the task lead to significantly worse results than the simple noun count vectors (Perceptron+ vs. Perceptron). However, this does not tell anything about the perceptron's performance when using other feature combinations.

The error distribution of the multi-layer perceptron is displayed in Figure 6 (c). Compared to the perceptron, most of the classes are predicted much better and there are less outliers overall. However, there are more major outliers which explains the still rather low performance. In further contrast to the perceptron, the use of all features leads to a performance increase (MLP+ vs. MLP). This is probably the case because the multi-layer percep-



(a) Doc2Vec (b) Perceptron (c) MLP+

Figure 6: Confusion matrices of different models on the test set (MST=140).

tron can learn individual weights for the different feature groups through multiple branches and layers much better than the single-layer perceptron.

6 Summary & Conclusion

Songtext–artist classification is an example of multiclass text classification on unbalanced, sparse and noisy data. Three neural network models have been investigated on this specific task. 1) A single-layer perceptron which can be used for all kinds of classification on vectorised data. 2) Doc2Vec which is a contemporary tool for text classification. And 3) an extended multi-layer perceptron which we designed specifically for this task. While the third and most complex model achieves the best results, it becomes also visible that the choice of features has a significant effect on the classification performance. Here, too, the multi-layer network with its advanced combination of different, stylometric as well as count/embedding-based, feature groups outperforms the other models.

We come to the conclusion that a vast number of and unbalanced classes as well as sparse and not directly correlated data do not allow for a perfect performance. Thus, given a text classification task where the data is as difficult, it makes sense to reduce the data to something that is manageable and meaningful. Sparse classes in a noisy sample space are little more than guesswork which might confuse the classifier and decrease the performance on more important classes. While it is somewhat obvious that removing difficult cases from the data improves the overall results, it is not something that one would usually do in a real-world application. We argue, that it can be a practical step to approach such a classification task, for elaborating the complexity of the classifier and engineering good features.

7 Future Work

A general clustering-based topic model encoded in new features could potentially improve the performance of songtext classifiers. Looking at our multi-layer perceptron, new features seem to be a good way to handle such difficult data. Other network architectures such as CNNs and RNNs can be considered worth a look as well since they improved (noisy) text classification in previous studies (e.g. Lai et al., 2015; Apostolova and Kreek, 2018).

Another way of dealing with imbalanced data is to apply oversampling to raise the number of samples for sparse classes, or undersampling to reduce the number of samples for frequent classes.

8 Acknowledgements

We thank the three anonymous reviewers for their valuable comments. This work emerged from a course at the Institute for Natural Language Processing of the University of Stuttgart; we thank our supervisors Evgeny Kim and Roman Klinger. We further thank Sebastian Padó for the organisational support.

References

Emilia Apostolova and R. Andrew Kreek. 2018. Training and prediction data discrepancies: Challenges of text classification with noisy, historical data. *arXiv preprint arXiv:1809.04019*.

Russell Beale and Tom Jackson. 1990. *Neural Computing: An Introduction*, pages 67–73. CRC Press.

François Chollet et al. 2015. Keras: Deep learning library for theano and tensorflow. *URL: https://keras.io/k*, 7(8):T1.

Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*.

Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196.

Ji Young Lee and Franck Dernoncourt. 2016. Sequential short-text classification with recurrent and convolutional neural networks. *arXiv preprint arXiv:1603.03827*.

Rudolf Mayer, Robert Neumayer, and Andreas Rauber. 2008. Rhyme and style features for musical genre classification by song lyrics. In *Ismir*, pages 337–342.

Ryan McDonald, Keith Hall, and Gideon Mann. 2010. Distributed training strategies for the structured perceptron. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 456–464. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Frank Rosenblatt. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1985. Learning internal representations by error propagation. Technical report, Institute for Cognitive Science, University of California.

Stacy L. Smith, Marc Choueiti, Katherine Pieper, Hannah Clark, Ariana Case, and Sylvia Villanueva. 2019. Inclusion in the recording studio? Gender and race/ethnicity of artists, songwriters & producers across 700 popular songs from 2012-2018. USC Annenberg Inclusion Initiative.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Alper Kursat Uysal and Serkan Gunal. 2014. The impact of preprocessing on text classification. *Information Processing & Management*, 50(1):104–112.