Project Report:
Deep Learning for Text Classification in Digital Journalism

Jan Paul Assendorp

Hamburg University of Applied Sciences, Dept. Computer Science,
Berliner Tor 7
20099 Hamburg, Germany
janpaul.assendorp@haw-hamburg.de

I. Introduction

The emerging field of digital journalism benefits vastly from recent developments in *deep learning* for *natural language processing* (NLP). Current deep learning models can be effectively used for NLP tasks such as speech recognition, machine translation, sentiment analysis or simply classification of text and often outperform traditional text mining methods. Those tasks are broadly associated to the latest challenges in journalism in the face of digital media. Not only do modern journalists have a need for systems which can ease access and modification to existing corpora. but also are journalists nowadays faced with a never ending stream of new topics and hence new stories, which have to be identified and reduced into a presentable format for the interested audience.

The scope of this project is to identify and evaluate an appropriate tool chain for text mining in preparation for the subsequent master thesis. This paper will focus on the challenge of document-level classification by exploiting the latest developments in the research fields of NLP and machine learning. The task of incorporating this projects results into a recommender system for dossiers is out of scope for this paper and will be discussed in future work. The following section II will briefly summarize the concept of dossiers and the benefits of enhancing the process of dossier creation with automated machine learning systems. Section III explores different approaches of recent studies which aim to achieve similar tasks to document classification. Therefore the differences in traditional methods compared to its counterparts from the field of deep learning will be examined. Section IV describes *convolutional neural networks* as promising deep learning model for NLP. Subsequently section V will describe experiments of this project and briefly discuss the results.

II. Recommender system for dossier creation

The main focus of this project is the task of text classification due to its similarities to the concept of dossiers in journalism. The goal is to train a predictive model to categorize documents from a selected corpus to a given set of labels.

In the context of digital journalism, dossiers can be seen as a set of multimedia documents focusing on a topic of interest. Hence dossiers form an important tool for representing a selection of significant sources of information for a given subject to an interested audience. In this project the focus will be solely on textual data. In the past the content of dossiers has been handpicked by journalists in a time consuming process which required a deep knowledge of the topic. Today this process can be massively accelerated by text mining applications. The concept of dossiers this project is based on can be further obtained from [1].

A news agencies archive of textual documents is likely already labeled to some extend. Even though dossiers focus on a given historical, political or cultural subject of interest, the task of categorizing news wires can be seen as similar. The core of text categorization is to find an appropriate representation of textual data where metric types of measurements can be used in order to compare different textual documents in terms of similarity. In comparison, dossiers contain texts which also need to have a certain amount of semantical similarity. Based on this assumption, text classification will be seen as a baseline task in the conception of a recommender system for dossiers.

A system for dossier recommendations could be used to reduce the huge amounts of documents in an archive to a manageable set of texts focusing on the subject of interest. The journalists could subsequently evaluate the recommendations and choose a subset of texts to be incorporated into the dossier. For this process it is crucial that the system identifies all relevant items from the corpus. In contrast, a small number of non relevant texts can be easily disregarded by the journalist. This observation is significant for the evaluation of a predictive model, as described in the following sections.

Another important observation is the disparity in performance of machine learning models on different datasets. Therefore it is essential to use a set of documents for the evaluation of the predictive model which is mostly similar to the target textual data.

III. Document classification

The task of document-level classification has been broadly explored in recent literature. Nevertheless, the accuracy of classification models has been steadily improved by applying state-of-the-art research and novel techniques. This section will describe traditional as well as recent research on methods to accomplish tasks similar to the categorization of text on document-level. Section III-A will discuss language models for representing textual data as a crucial task in text processing.

Text classification is one of the standard applications in text mining. The objective in classification is to find appropriate labels for previously unlabeled data from a predictive model which has been trained on a given set of examples. In general the traditional process of text categorization can be described

by a series of necessary subtasks performed to identify and extract relevant features from a given text, which can be further applied to train a predictive model. Those subtasks can be reduced to the different steps shown in figure 1.

Textual data can be seen as a sequence of words where each term is mapped in a global dictionary. In data mining, text is generally seen as unstructured data [2] even though it implies structural information by the given rules of grammar. Hence the difficulty in text mining lies in capturing these structural or contextual constraints from a set of words. Based on the extracted features from documents, text can be represented in the algebraic *vector space model* [3] wherein each document is described by a vector which holds information about its containing words. With these vectors, different documents can be compared and thus clustered or classified utilizing standard vector algebra. Therefore a common method is to construct a document-term-matrix to compute similarities. Language models for representing text in a vector space are represented in the following section III-A.

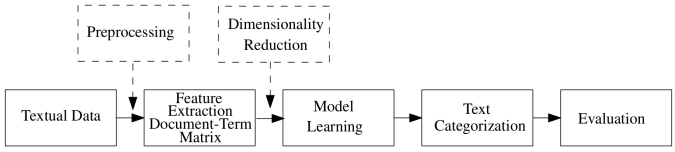


Figure 1. Pipeline of text categorization [4]

In order to improve the information gained from single tokens, *preprocessing* can be applied to clean text prior to feature extraction. Preprocessing includes *tokenization* and *normalization* of text. In addition *stopwords* can be filtered for removal of noise introduced by words without any semantical significance. Another often used method in preprocessing is to reduce words to a common stem in order to decrease the dimensionality of word vectors and thus improve the comparability of documents.

Unique tokens often have different value for understanding the subject of a document. A simple method of weighting features is by counting the corresponding words frequency. However, this does not regard the significance of terms which rarely appear in general but have a relatively high frequency in specific topics. To consider relative importance of terms, the *term-frequency-inverse-document-frequency* (TF-IDF) [5] weighting scheme can be applied. This measure has been proven to increase the quality of feature vectors in classification of text [6].

Especially in classification of large documents the dimensionality of feature vectors is increasingly high due to the large number of terms in the vocabulary. This dimensionality can be even further increased once sequences of words are considered as features. However, these feature vectors are in most cases sparse as many features solely appear in a small fraction of the documents. Hence it is often required to reduce the feature vectors dimensionality to enable the classification algorithms to produce good results. A commonly used method for dimensionality reduction is the *principle component analysis* (PCA) which can reduce a matrix into a lower dimensional representation.

Traditionally, relevant features from text are handcrafted in a time-consuming manner and require an extensive amount of preprocessing of raw text. In addition, those features are often over-specified and incomplete [7]. The eminent difference between the traditional approach and recent deep learning methods is the automated learning of representations from text. Features can hence be learned from raw text and represented in some kind of language model.

The following section describes different language models which can be applied to capture relevant information from a given text. These model can be later used to compare and classify documents.

A. Language models

The naive approach to capture the meaning of text is to count the occurrences of distinct terms in the documents. The result can be captured in a *Term-Document matrix* where documents are represented as a vector of dimension n, where n is equal to the amount of distinct terms in the set of all words in the given dictionary. Consequently those vector representations can be utilized to calculate arithmetic distances between single documents.

1) Bag-of-words: Text can be interpreted as a bag of words ignoring the sequential information of co-occurring terms. Hence the semantics of each combination of words or general grammatical structure will be lost completely. Nevertheless, the *bag-of-words* model and thus the assumption of term independence has been proven to be quite efficient in many applications of text mining due to its simplicity [8].

2) Bag-of-n-grams: To initially capture the semantics of word sequences, co-occurring terms within a fixed window around a given center word can be considered as features. For instance the mining of 3-gram features from text has lead to good results in experiments by *Rousseau et al.* in [9]. However, n-grams arouse the disadvantage of a highly increased dimensionality of feature vectors as well as sparsity due to decreased likelihood of n-grams co-occurring in different documents.

3) Graph-of-words: In order to extend the concept of n-grams *Rousseau et al.* introduced the *graph-of-words* model in [9], wherein word co-occurrences in a document were translated into an undirected graph. In this graph model, vertices represent distinct terms whereas edges are given by co-occurrence between two terms.

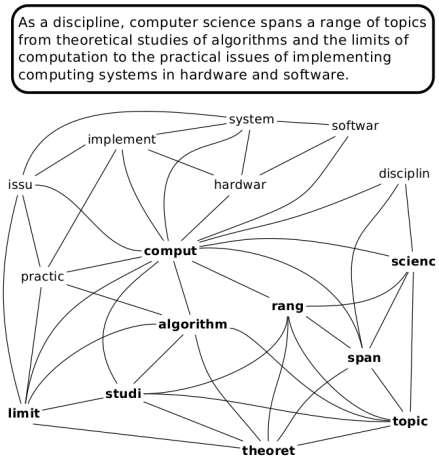


Figure 2. Graph-of-words representation of text *Graph-of-Words* [9]

The resulting graph-of-words can be interpreted as a set of long range n-grams to enhance the extraction of meaningful features from text. Subgraphs can be directly mined as features and mapped to a vector representation of the document. Another interesting structural characteristic of the graph representation is the main core defined as vertices with the highest number of outgoing edges.

Figure 2 shows the main core of a graph-of-words as bold labeled vertices. The core vertices can be seen as the most important terms of the related document and can even be used to reduce the feature space of n-grams for better classification accuracy [9].

However, directly mining subgraph features leads to high dimensional feature vectors which are very sparse. In addition, the feature vectors hold solely binary values as each subgraph can by definition only appear once in a graph representation of a document holding distinct term vertices. Even the reduction of the graph-of-words to the main core results in a high dimensional vector space.

Apart from the main core, the graph-of-words features profit from disregarding word-order compared to n-grams of similar window size. This can lead to an increasing classification accuracy, as shown in [9].

4) Vector representations of words: One of the biggest drawbacks from the previously mentioned discrete language models are the sparse feature vectors which might break the classification algorithms performance or at least require additional work to efficiently handle large but sparse matrices. In comparison, image processing applications in machine learning benefit from dense matrices given by pixel data of images. To approach this problem, *Bengio et al.* introduced in [10] word embeddings as dense lower dimensional vector representations of text. In this model, every word in the dictionary is encoded in a vector representing the possibility of word co-occurrences to other terms in the dictionary. The underlying assumption of this continuous approach is that similar words are likely to have similar vectors [11]. Given a sufficiently large dataset, this pattern captures a great deal of the semantic relationships of words. Semantically close words are therefore encoded in the vector space as close vectors in terms of euclidean or cosine distance [12]. This allows for vector equations such as $king - queen = man - woman$ [13].

The approach of dense vector representations can either be achieved with global matrix factorization methods like *Latent Semantic Analysis* (LSA) [14] or by local context window methods e.g. the *skip-gram* [11] or the *continuous bag-of-words* (CBoW) [15] model introduced by *Mikolov et al.*. The objective of the continuous *skip-gram* model is to predict the context of a center word [16] wherein CBoW predicts target words from their surrounding context which can be useful for a small dataset.

Sophisticated implementations for computing word embeddings are available through *word2vec* [16] or *GloVe* [13]. Both methods ultimately rely on unsupervised learning of co-occurrence statistics of words in the corpus.

Instead of learning word vectors from the limited context of the training set, word embeddings can be trained on the largest available collection of text and subsequently applied for training a classifier based on the original data. Using static or non-static pretrained *word2vec* or *GloVe* vectors has been proven to increase the classification accuracy [17]. However, it has also been shown that non-static word embeddings outperform their static variants as the vectors are adjusted to the dataset during training [17].

B. Traditional text classification

A common approach to text classification is to compute a bag of n-grams from preprocessed textual documents and represent them in a term-document-matrix. Subsequently a *TF-IDF* weighting scheme can be applied to emphasize often occurring words in a single document which are rather rare within the corpus. To further reduce the size of vector dimensionality, algebraic methods like *PCA* can be applied to the term-document-matrix. It is also common to use a linear kernel *SVM* [18] classifier to categorize the transformed documents due to its simplicity as well as good results on high dimensional feature vectors [19] [9]. Even in this projects experiments, this method has proven to result in a good classification accuracy as seen in section V-C.

C. Deep learning for text classification

In traditional machine learning, the selection of relevant features from data is a crucial task for a successful application. In contrast, deep learning introduces end-to-end learning and hence the automated learning of discriminating features from data. This simplifies the otherwise complex process of hand-picking features from text. Compared to traditional methods, CNNs have been proven to deliver highly competitive results [20]. The following section IV briefly introduces *convolutional neural networks* as a promising model for language processing.

IV. Convolutional neural networks for text classification

Convolutional neural networks (CNNs) have recently produced remarkable results in machine learning for image processing [21]. While similar to the concept of *recurrent neural networks* (RNNs) regarding the ability to embrace sequential input data, CNNs can also be used for NLP tasks such as text classification. It can be shown that even a simple convolutional model results in good classification accuracy [12] [17]. This section will briefly introduce the concept of CNNs and how it can be applied for text classification tasks.

A. Concept of CNNs

The underlying concept of CNNs relies on layers of convolving filters applied on shifting windows over an input matrix and thus extracting local features [22]. Another architectural component is a pooling layer, which results in a certain degree distortion of feature locality. All these layers have weights, which are subject to be learned by the model.

An example architecture of a single layer CNN for sentence-level classification by *Kim et al.* in [12] is shown in figure 3. The input layer consists of a dense embedding matrix. Given a sentence represented as a word vector s, where each word is encoded in an embedding vector with dimension d, the input layer is a $s \times d$ matrix. Essentially, the length of each sentence has to be padded to a fixed maximum n, due to the CNNs fixed input size [12].

In contrast to image processing, convolution filters are applied to the entire width of the input layer but have variable height, defined as *region size* [17]. Therefore, a convolution

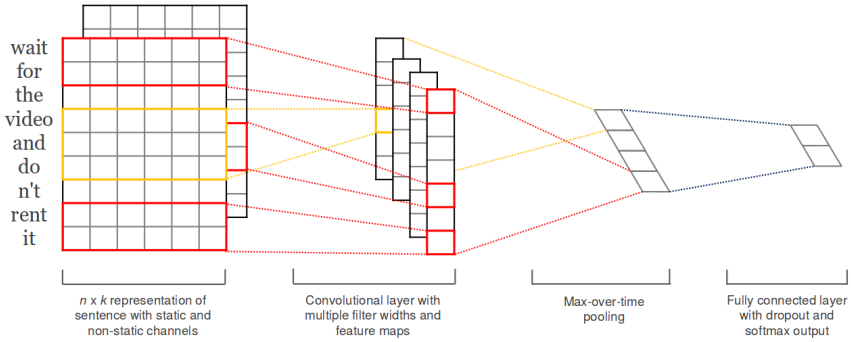


Figure 3. CNN architecture for text classification [12]

filter parameterized by a weight matrix $w \in \mathbb{R}^{hd}$ uses a window of h words for the extraction of a new feature c_i. Given $x_{i:i+h-1}$ refers to a concatenation of words $x_i, x_{i+1}, ...x_{i+h-1}$ of a window, the new feature is defined as shown in equation 1 [12].

$$c_i = f(w * x_{i:i+h-1} + b) \tag{1}$$

The activation function f is essentially non-linear e.g. a *rectified linear unit* (ReLU) [23] and $b \in \mathbb{R}$ is a bias term. The resulting feature map $c \in \mathbb{R}^{n-h+1}$, where

$$c = [c_1, c_1, ...c_{n-h+1}] \tag{2}$$

can then be obtained by applying the filter to each possible window over the input sentence [12].

Following the convolutional layer, a *max-over-time pooling* operation [24] can be applied on the feature maps in order to extract the maximum value $\hat{c} = max\{c\}$ and hence capture the most important feature to the corresponding filter [12]. Often *1-max pooling* [25] is applied to produce the feature maps [17].

Subsequently the extracted features can be classified in a fully connected softmax layer resulting in an output representing the probability distribution over the class labels [12].

Training the model includes the filter weights w, the bias term b as well as the softmax weights. Minimizing the categorical cross-entropy loss is a reasonable training objective [17]. *Stochastic gradient descent* (SGD) can be used in back-propagation to optimize the model parameters.

To prevent the model from *overfitting* the training examples, *regularization* can be added to the softmax layer. *Hinten et al.* proposed *dropout* in [26] as a method for regularization by randomly omitting parts of the model. In this case, dropout can be applied by setting the effected weights to zero during back-propagation [12]. Another common method for regularization is to apply *l2 norm* constraints [27]. However, it does not improve the performance in NLP tasks [17].

The model shown in figure 3 also includes multiple input channels. This allows e.g. the simultaneous use of static as well as non-static word embeddings. Another interesting approach is to use static pretrained word embeddings in one channel along with embeddings learned from the training set in a separate channel [12].

V. Experiments

In order to evaluate an appropriate pipeline for classification of document corpora in journalism, several methods have been experimented with and are presented in this chapter. Firstly, the traditional method of classifying preprocessed texts by extracting weighted n-grams and feeding them to a SVM has been implemented. This method has been further evaluated by extracting subgraph features from the texts and representing them in a graph-of-words model. Finally a CNN-approach and therefore a deep learning approach incorporating pretrained word embeddings led to the best results on the used corpus.

A. Experimental setup

Data transformation and preprocessing has been done in *R* using the text mining package *tm*. Training and validation of the weighted n-gram features has been carried out using the linear SVM implementation from the *e1071*[1] *R* package for statistical computations. *R* has also been used for plotting the results of different classification models. Mining the frequent subgraph features has been carried out using *gSpan*[2] due to its fast *C++* implementation. The code for training the deep CNNs has been written in *python* using Googles recently published *tensorflow*[3] framework for deep learning models.

All experiments have been carried out on a machine running a common 4-core CPU (Intel Core i7 4770k) clocking at 3.5GHz with 32GB of RAM. Training of the CNNs was done on a GPU with 1536 CUDA cores and a total memory of 4GB.

B. Dataset

The classification experiments have been carried out on the *Reuters R8*[4] corpus. The dataset contains the eight most frequent classes from the *Reuters-21578* collection of news stories. The eight classes consist of a minimum of 51 up to 3923 documents and sum up to a total of 7674 texts. The dataset is interesting due to the variety in text length as well as its skewed multi-class scenario. However, the collection has been selected in order to produce comparable results to the experiments *Rousseau et al.* in [9] and *Malliaros et al.* in [4].

[1] https://cran.r-project.org/web/packages/e1071
[2] https://www.cs.ucsb.edu/~xyan/software/gSpan.htm
[3] https://www.tensorflow.org/
[4] http://www.cs.umb.edu/~smimarog/textmining/datasets

C. Results

1) N-grams and linear SVM: To be able to compare the CNN-approach to traditional models, a SVM with a linear kernel has been used to classify documents previously preprocessed and represented as n-gram features. The preprocessing includes the removal of stopwords and stemming of words to reduce textual noise in the documents. Subsequently 3-grams are extracted from the texts and represented in a shared vector space. The features are weighted following the *TF-IDF* scoring function. The resulting term-document-matrix is very sparse due to the large number of n-gram features. Hence the sparsity of the matrix is reduced to allow higher performance of the SVM. The resulting *accuracy* and *F1-score* of the SVM classifier is presented in table I. Measuring the harmonic mean of *precision* and *recall*, the *F1-score* can be effectively used in a skewed class distribution. To generalize the results regarding the used train and test split, 10-fold cross-validation has been applied.

2) Graph-of-words and linear SVM: To evaluate the performance of the *graph-of-words* model for extracting subgraph features, an approach similar to *Rousseau et al.* in [9] has been implemented. The set of documents has been cleaned, stemmed and ultimately transformed into a graph representation of the containing words. The main core was subsequently extracted utilizing the *igraph*[5] package in *R*. Mining of the most frequent subgraphs has been done with *gSpan* on each class separately due to the skewed class distribution of the dataset. Following the example in [9], a relative *support* value of 7% was applied to extract a proportional number of frequent subgraphs regarding the class size. However, the good results from the original paper could not be reproduced in this experiments. A possible reason is the different method for retrieving the main core used here. *Rousseau et al.* achieved an accuracy which clearly outperforms the n-grams combined with a linear SVM as described in the previous section V-C1. Table I contains the results taken from [9] for comparison.

3) CNN and GloVe embeddings: The selected network architecture is similar to the baseline single-layer CNN used by *Kim et al.* in [12] due to its simplicity and yet good results. However, the input has been limited to a single channel to focus on the comparison of different word embeddings. Nonetheless, expanding the model to dual channel input with static pretrained as well as non-static learned representations is subject of future experiments as described in section VI. In addition, the classifiers in this project were applied on document-level instead of sentence-level described in [12].

The embedding layer of the CNN has been altered to fit the dimension of the embeddings as well as the maximum length of the concatenated sentences of single documents. A total number of 128 convolutional filters per filter size were used with *rectified linear units* (ReLU). The different filter sizes were set to $3, 4$ and 5. Dropout was applied with a *dropout rate* of 0.5. These hyperparameters were chosen similar to the network used by *Kim et al.* in [12].

For training the network, *Adam* optimizer [28] has been used for optimization of the models parameters. The data set has been shuffled and randomly split into about 85% training data and 15% evaluation data. In contrast to the experiments on the SVM classifier, cross-validation has been disregarded to speed up the experiments. Training was done with a batch size of 64 and the model was trained in 200 epochs. For visualization, the evaluation of the model was run every 100 training steps. The evaluation consisted of calculating classification *accuracy* as well as *cross-entropy loss*.

Several models have been trained on different pretrained GloVe word embeddings[6]. GloVe has been selected over word2vec due to its better performance in terms of word analogy and word similarity [13]. The GloVe word vectors used in these experiments have been trained on a set of 6 billion tokens from *Wikipedia* and *Gigaword 5* containing $400k$ words and were subsequently reduced to a dimension of $50, 100$ and 300. A major difficulty has been handling unknown terms or padding tokens, which are not trained in the GloVe embeddings. This has been resolved by using a semantically insignificant default lookup term for unknown words. However, it is expected, that the models accuracy can be increased by pretraining a term without any importance to the classification task.

Initially, the first model was trained using an non-static embedding layer of dimension 128 which has been initialized randomly and learned during the training. Figure 4 shows the classification accuracy during training as well as evaluation. Most interestingly, learning the embeddings from the training data seems to work well on the dataset and provides a good accuracy. However, it can also be noted, that during training the models seems to *overfit* the training data which results in a decreasing trend of difference in training accuracy compared to evaluation accuracy.

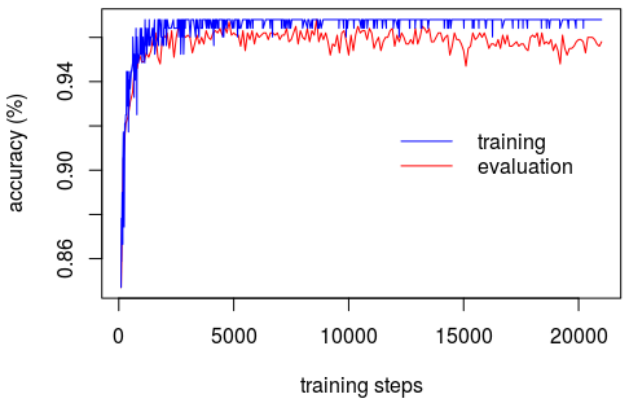
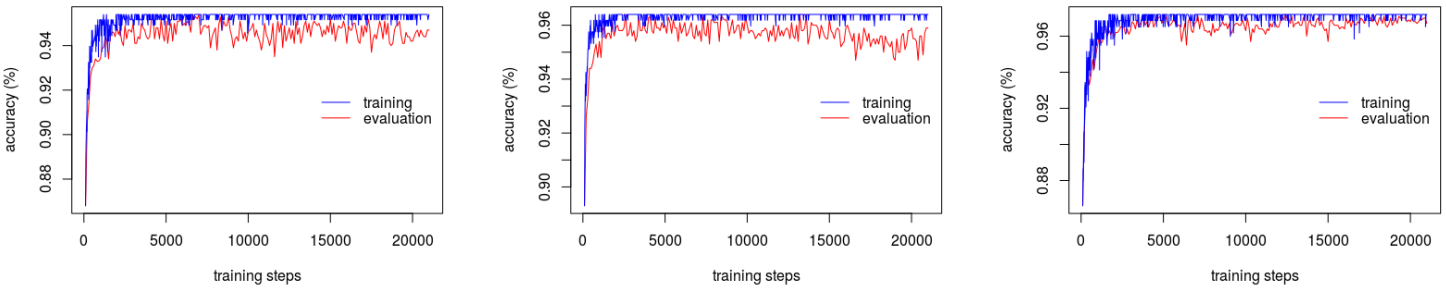


Figure 4. CNN training and evaluation accuracy on randomly initialized word embeddings learned in training

An explanation for the good results of word embeddings learned from the training data is the length of the documents. Compared to short-text classification, the dataset used here incorporates sufficient contextual information for distinct terms of the vocabulary. Due to limited memory on the GPU used for training, the experiments with non-static embeddings have been limited to preprocessed input data only. Hence an addition

[5]http://igraph.org/r/

[6]Embeddings available here: http://nlp.stanford.edu/projects/glove/



(a) 50-dimension embeddings on preprocessed text (b) 100-dimension embeddings on preprocessed text (c) 50-dimension embeddings on raw text

Figure 5. CNN training and evaluation on static GloVe word embeddings

in future works can be rerunning this model on raw input text and comparing it to the previous results.

The experiments shown in figure 5 have been applied on static pretrained GloVe embeddings. Due to memory limitations the experiments have been carried out starting with low-dimensional GloVe embeddings. Furthermore, in some cases the input data has been preprocessed and thus stemmed to reduce the number of distinct terms in the training set. Figure 5a shows the training and evaluation results on 50-dimensional GloVe embeddings pretrained on a vocabulary of $400k$ terms with preprocessed input data.

In contrast to word embeddings learned from the training set, figure 5a shows a significant decrease in classification accuracy during evaluation. This example shows, that the model does not benefits from the pretrained embeddings once their dimensionality is significantly lower.

Based on this assumption a model has been trained on 100-dimensional GloVe embeddings, which are similar to the 128-dimensional non-static embeddings learned during training. Figure 5b proves that pretrained GloVe embeddings outperform the learned word vectors with similar dimensionality on preprocessed input text previously shown in figure 4.

Using preprocessed text on top of pretrained embeddings does not fully exploit the underlaying semantic structure captured in the word vectors. Hence it is more natural to apply the raw text on top of the embeddings. Due to hardware limitations in these experiments, applying raw text has been solely used on top of 50-dimensional GloVe embeddings. Figure 5c shows the training and evaluation accuracy of the model trained with raw text inputs. The evaluation accuracy increases significantly compared to the other models. This can be explained by the better use of semantic information in the embeddings as described before.

However, figure 6 shows a significantly increasing trend in cross-entropy loss over the training period. This is a sign for a certain extend of overfitting of the training model which can be improved by adjusting the *dropout rate*. The cross-entropy loss might also be improved by changes in the CNN architecture. Ultimately it is also influenced by the chosen training and evaluation split of the shuffled documents. Nevertheless, the results from the different CNN models demonstrate that deep models can easily outperform traditional classifiers.

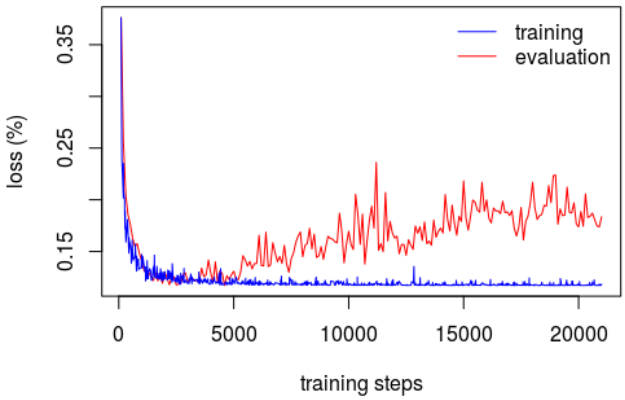


Figure 6. CNN training and evaluation cross-entropy loss on 50-dimensional GloVe embeddings using raw documents

D. Summary of results

During the different experiments the model *accuracy* has been recorded to allow for a practical comparison of the different approaches. The following table I lists the results of classification on the *Reuters R8* collection using a traditional method and different CNN models. In addition to the results gained through experiments in this project, the results of the *graph-of-words* approach coupled with a linear kernel SVM has been added from [9] for comparison.

TABLE I. Classification results on *Reuters R8* dataset

Method	Accuracy
N-Grams and linear SVM	0.937
GoW and linear SVM [9]	0.955
CNN + variable Embeddings + preprocessed text	0.958
CNN + GloVe (50d) + preprocessed text	0.947
CNN + GloVe (100d) + preprocessed text	0.959
CNN + GloVe (50d) + raw text	**0.967**

The *accuracy* of the n-gram model has been recorded with 10-fold cross-validation which has been omitted for the CNN

models due to the computational complexity. In addition, the weighted *F1-score* has been calculated for the n-gram approach which resulted in a value of 0.848. In comparison, the GoW approach in [9] achieved a *F1-score* of 0.864.

VI. Conclusion and Future work

In this project different classification models have been evaluated for categorizing the *Reuters R8* news collection. The chosen dataset is seen as a good baseline for simulating a task on a journalistic archive. This projects experiments focused on deep learning methods for text classification due to promising results achieved in recent academic papers.

Even though the CNN used in the experiments is a basic model and has not been further optimized to the task at hand, the results show a significant higher performance in direct comparison to traditional approaches on the considered dataset. This leads to the assumption that a CNN similar to the one evaluated in the experiments is the preferable model for the task of classification in a context of journalism.

However, the evaluated model can be further optimized for the given dataset. The first step would therefore be the implementation of cross-validation as well as a *F1-score* measurement to improve the comparability of different models on a dataset with skewed class distribution. Furthermore, the CNN can be tuned by including *l2 norm* regularization or adapting the *dropout rate* to prevent overfitting. Nevertheless, as stated in IV-A the effect of regularization might be only insignificantly improving the models accuracy.

A significant improvement can be expected from switching to higher dimensional pretrained word embeddings. Commonly 300-dimensional word embeddings are used in recent publications [17] [12]. In addition, applying raw text data which ultimately holds the most contextual information has been shown to improve the classification accuracy. These changes however require training on a superior GPU compared to the one used in the experiments. The pretrained word embeddings can also be used in a non-static way and hence be optimized during training. It is also possible to experiment with dual input channels holding both static as well as non-static embeddings as described by *Kim et al.* in [12].

A trend in latest research on text classification is to use input on character-level [20]. Character-level classification utilizing CNN models has been shown to improve the results on large datasets e.g. on millions of examples whereas it does not achieve better results on smaller datasets.

References

[1] N. Hälker, "Teilautomatisierte Erstellung von Dossiers auf Basis von Textmining-Verfahren," Masterthesis, HAW Hamburg, 2015. [Online]. Available: http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/haelker.pdf

[2] R. Feldman and J. Sanger, The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data. Cambridge University Press, 2007. [Online]. Available: https://books.google.de/books?id=U3EA_zX3ZwEC

[3] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," Commun. ACM, vol. 18, no. 11, Nov. 1975, pp. 613–620. [Online]. Available: http://doi.acm.org/10.1145/361219.361220

[4] F. D. Malliaros and K. Skianis, "Graph-based term weighting for text categorization," in Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015. ACM, 2015, pp. 1473–1479.

[5] A. Singhal, J. Choi, D. Hindle, D. D. Lewis, and F. Pereira, "At&t at trec-7," NIST SPECIAL PUBLICATION SP, 1999, pp. 239–252.

[6] X. Fu, E. Ch'ng, and U. Aickelin, "An improved system for sentence-level novelty detection in textual streams," 2015.

[7] R. Socher, "Recursive deep learning for natural language processing and computer vision," Ph.D. dissertation, Citeseer, 2014.

[8] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," arXiv preprint arXiv:1405.4053, 2014.

[9] F. Rousseau, E. Kiagias, and M. Vazirgiannis, "Text categorization as a graph classification problem," in Proceedings of the 52th Annual Meeting of the Association for Computational Linguistics and the 6th International Joint Conference on Natural Language Processing (ACL-IJCNLP '15), 2015.

[10] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," journal of machine learning research, vol. 3, no. Feb, 2003, pp. 1137–1155.

[11] T. Mikolov, W.-t. Yih, and G. Zweig, "Linguistic regularities in continuous space word representations." in HLT-NAACL, vol. 13, 2013, pp. 746–751.

[12] Y. Kim, "Convolutional neural networks for sentence classification," arXiv preprint arXiv:1408.5882, 2014.

[13] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[14] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," Journal of the American society for information science, vol. 41, no. 6, 1990, p. 391.

[15] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," arXiv preprint arXiv:1301.3781, 2013.

[16] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in Advances in neural information processing systems, 2013, pp. 3111–3119.

[17] Y. Zhang and B. Wallace, "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," arXiv preprint arXiv:1510.03820, 2015.

[18] T. Joachims, Text categorization with support vector machines: Learning with many relevant features. Springer, 1998.

[19] O. Walter, R. Haeb-Umbach, B. Mokbel, B. Paassen, and B. Hammer, "Autonomous learning of representations," KI-Künstliche Intelligenz, vol. 29, no. 4, 2015, pp. 339–351.

[20] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in Advances in Neural Information Processing Systems, 2015, pp. 649–657.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in neural information processing systems, 2012, pp. 1097–1105.

[22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, 1998, pp. 2278–2324.

[23] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in Proceedings of the 27th International Conference on Machine Learning (ICML-10), 2010, pp. 807–814.

[24] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," Journal of Machine Learning Research, vol. 12, no. Aug, 2011, pp. 2493–2537.

[25] Y.-L. Boureau, J. Ponce, and Y. LeCun, "A theoretical analysis of feature pooling in visual recognition," in Proceedings of the 27th international conference on machine learning (ICML-10), 2010, pp. 111–118.

[26] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," arXiv preprint arXiv:1207.0580, 2012.

[27] A. Y. Ng, "Feature selection, l 1 vs. l 2 regularization, and rotational invariance," in Proceedings of the twenty-first international conference on Machine learning. ACM, 2004, p. 78.

[28] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.