University of Amsterdam

Natural Language Models and Interfaces

Convolutional Neural Network versus Naive Bayes classifier on Internet Movie Database dataset

March 30, 2020

Author: Thomas van Orden - 12192880 - thomasvanorden@live.nl
Teaching Assistant: Puck de Haan - UvA

Universiteit van Amsterdam

Contents

1 Introduction

1.1 Problem

Nowadays, we see reviews everywhere around us. Concerning the film industry, reviews can 'make or break' a movie. The score of a movie might be the reason to go. If this score is mainly calculated based on the amount of positive and negative reviews, how does one decide if a review is positive or negative? This question yields a binary sentiment classification, since a review can only be positive or negative. Doing this for all reviews by hand will be very time exhaustive. Binary classification machine learning techniques might help to automate this task and reduce time significantly. However, there is a wide range of techniques to choose from. So, choosing the best technique is a very extensive, yet important, task.

In this report we discuss and compare two different techniques used to establish this task, concerning the well known Internet Movie Database (IMDb) dataset (Maas et al. 2011). The first technique is a classic technique, namely a Naive Bayes (NB) classifier. The second technique is a more modern technique, namely a Convolutional Neural Network (CNN). For more details on the configuration of both models, see section 2.3.2 and 2.3.1.

1.2 Research questions

The main research question of this report is: **To what extend can a CNN improve the F1-score on the IMDb dataset in comparison to a NB classifier?**

A CNN is mostly used for image processing, where the convolutional hidden layers in the network can recognize patterns in images by acting as a series of filters on the image. We expect that texts can contain some patterns too. For example, some words are mainly used in a positive manner and other words in a negative manner. If these patterns can be discovered in the dataset, the F1-score of the CNN is expected to improve. In addition, previous research showed that high accuracy can be obtained by applying a CNN for binary classification on short texts (Dos Santos and Gatti 2014). Movie reviews are short texts too, according to the dataset they used.

NB classifiers are based on Bayes' theorem with a strong (naive) independence assumption between the features. Since single words are the features for the NB classifier (see section 2.3.2 for more detail), all words are viewed without context. However, we assume that context does highly contribute to the tone (positive or negative) of a review. Based on the previous research (Dos Santos and Gatti 2014) and above assumption, we expect the CNN to outperform the NB classifier in terms of F1-score.

To extend this main question, below are two sub-questions listed.

- **How is the performance of the CNN effected by a skip-gram embedding versus a continuous bag of words (cbow) embedding?**

  Both cbow and skip-gram are model architectures for a Word2Vec model (*Google Code Word2Vec* 2013). A Word2Vec model converts a word to a vector. The cbow model uses a window of words surrounding a given target word to predict the target word. The bag-of-words assumption is used in the cbow model, hence the order of the words in the context (window) does not matter. On the other hand, skip-gram does the opposite. It predicts a window of surrounding words (context) given a target word. Words nearby the target word weigh heavier then words further away. For more details on the implementation of both embedding models, see section 2.3.1. According to the author's recommendation on the Word2Vec configuration, skip-gram should perform better for infrequent words than cbow (*Google Code Word2Vec* 2013). We assume that the most important words, for example *"good"* and *"bad"*, occur less often than general English words such as *"the"*. So, the important words are infrequent in our corpus and therefore we expect skip-gram embedding to perform better.

- **How effects lemmatization and stemming the input the performance of both models?**
  We expect that lemmatization yields the best improvement in terms of F1-score, since the input words are converted to their lemma. This lemma represents a higher morphological meaning of the word. Therefore, these lemmas will probably better show the tone of the input; negative or positive. Stemming on the other hand shortens words to their stem by removing pre- and suffixes. In this way, multiple forms of one word map to the same stem. However, this stem does still has the same meaning as the original word and is purely obtained by applying English language rules. So, we think that stemming will not make much difference in the performance of both models.

1.3 Analysis of the dataset

The dataset is split into two equally sized subsets, namely 'train' and 'test'. These sets are then further split into, again, equally sized subsets, namely positive and negative samples. Therefore, the dataset is completely balanced. The dataset can be found here. We assume that the train and test set both show the same language characteristics, such as sentence length. This assumption is based on the information in the *'README'* file, which is included in the dataset.

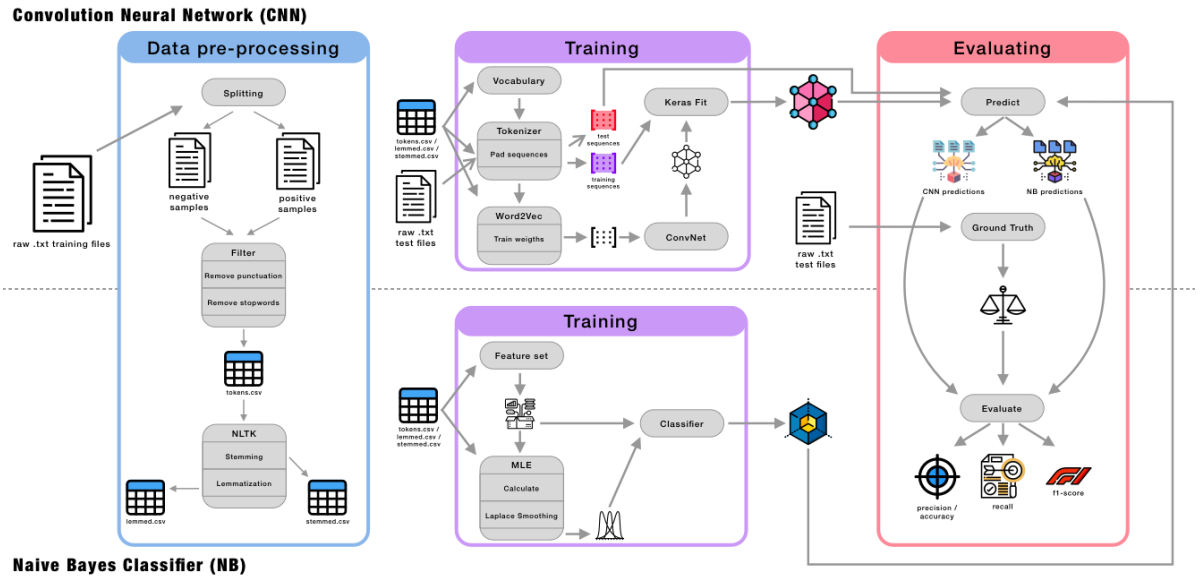

Figure 1: A schematic overview of this research' method.

Name	Version
gensim (Řehůřek and Sojka 2010)	3.8.1
keras (Chollet et al. 2015)	2.3.1
scikit-learn (Pedregosa et al. 2011)	0.22.2.post1
nltk (Bird and Klein 2009)	3.4.4
numpy (Oliphant 2006)	1.18.1
pandas (McKinney et al. 2010)	1.0.3
python (Van Rossum and Drake Jr 1995)	3.7.6

Table 1: Overview of the most important packages in environment.

2 Method

The method is displayed as a schematic overview in figure 1. All steps of the schema are discussed in further detail in the following sections. In addition, any requirements to run the experiments are listed in section 2.1. The source code can be found on the GitHub repository (Orden 2020).

2.1 Environment setup

Anaconda Distribution 2016 is used to create a custom and closed environment. Table 1 shows the most important packages and their versions that we used in this environment. Some functions in the training methods might use a substantial amount of memory or demand high CPU power. These functions are highlighted in section 2.3.1 and 2.3.2. The exported environment file ('env.yaml'), and information on how to install it, can be found on the Github repository (Orden 2020).

2.2 Data pre-processing

The dataset consists of a folder named 'train'. This folder contains two folders; 'positive' and 'negative'. Both contain reviews in .txt format. First of all, all .txt-files are converted to a dataframe. Each entry in the dataframe corresponds to one review, where the column 'label' can be 1 (positive) or 0 (negative).

As shown in the blue section of figure 1, the second step is applying multiple filters on the dataframe. At first, all reviews are tokenized by the *word_tokenize()* function from nltk (Bird and Klein 2009). Second, all tokens are converted to lowercase, since our models should not distinguish between tokens such as [*"The"*] and [*"the"*]. Third, all punctuation is removed from the sentences by using a regular expression. At last, stop words are removed from the sentences by using the *stopwords.words("English")* set of words from nltk (Bird and Klein 2009). All of the above filters are applied to obtain a more clean dataset, which better represents the meaning of a sentence by containing less meaningless characters or words. This dataframe will be further referenced as *tokens.csv*.

To further evaluate the effect of filtering the input, some more filters are applied. All entries in *tokens.csv* are lemmatized by using the *WordNetLemmatizer().lemmatize()* from nltk (Bird and Klein 2009). This lemmatized dataframe will be further referenced as *lemmed.csv*. Also, all entries in *tokens.csv* are stemmed by using *PorterStemmer().stem()* from nltk (Bird and Klein 2009). This stemmed dataframe will be further referenced as *stemmed.csv*.

2.3 Training

As shown in figure 1, the training process for both models differs and will thus be discussed in two different sections below. Since we created three different input dataframes (*tokens.csv, lemmed.csv* and *stemmed.csv*), the training process has to be done for all three files. However, this section will only discuss the general training process and therefore we will use *tokens.csv* here.

2.3.1 CNN

The training method for the CNN is adapted from the GitHub repository from saadarshad102 2019. All hyper-parameters are listed in table 2.

The CNN works with a input vector containing only digits, so the first step is to convert the input to a vector. The input may be one of three possible pre-processed files. The hyper-parameter INPUT takes care of this. First of all, we need to create a vocabulary of the words that are used in the dataframe. This vocabulary is then used to create a tokenizer object from keras (Chollet et al. 2015). The tokenizer is fit on the dataframe, by using the object-function *fit_on_texts()*. This functions creates a dictionary with words as key and the corresponding value is a unique integer. The lower the integer, the more frequent the word occurs in the dataframe. With this dictionary, we can convert a sequence of tokens to a sequence of integers. This is done with the object-function *text_to_sequence()*. In order to create a matrix of these sequences, all sequences have to be the same length. The keras *pad_sequences()* function

Name	Possible values	Used value
INPUT	{"tokens", "stemmed", "lemmed"}	All three
EMBED	{"cbow", "skipgram"}	Both
MIN_COUNT	$\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} \mid x \geq 0\}$	2
WINDOW_SIZE	$\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} \mid x \geq 0\}$	10
ITER	$\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} \mid x \geq 0\}$	10
EMED_TRAINABLE	{True, False}	True
MAX_SEQUENCE_LENGTH	$\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} \mid x \geq 0\}$	500
NUM_EPOCHS	$\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} \mid x \geq 0\}$	8
BATCH_SIZE	$\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} \mid x \geq 0\}$	64

Table 2: The hyper-parameters of the CNN.

is used to pad all sequences with 0.0 to a length of MAX_SEQUENCE_LENGTH. This matrix will be further referenced as *training_sequences*. The *text_to_sequence* and *pad_sequences* is applied on the test data too, the resulting matrix will be further referenced as *test_sequences*.

The third step is to create a cbow or skip-gram Word2Vec model on the input data (*Google Code Word2Vec* 2013). The hyper-parameter EMBED takes care of this. This creates trained word vectors, based on a context of length WINDOW_SIZE, for every word in the input data. The training process takes ITER number of iterations and words that occur less than MIN_COUNT are ignored. These word vectors are then transformed to a matrix of embedding weights such that all words in the tokenizer object correspond to a word vector. Note that this step might require a lot of memory. To reduce the memory usage, it is possible to not train the embedding weights further by setting the hyper-parameter EMBED_TRAINABLE to False. However, this might reduce the accuracy of the model.

The fourth step is creating the CNN itself. We used the same configuration of the CNN, excluding the values of the parameters, as the code from saadarshad102 2019. A schematic overview of this model is shown in figure 2. The trained embedding weights are used to create the *embedding_1* layer.

The last step is to fit the CNN on our input data. This is done by using keras' built-in function *fit()*. Note that this demands high CPU power and requires a lot of memory. On multi-core machines, multi-threading can be turned on by setting the argument *use_multiprocessing=True*. In addition to reduce time, the number of epochs can be set manually.

2.3.2 NB

Since this code for the NB classifier was already given and therefore only hyper-parameters are tuned, we will discuss the code only briefly. All hyper-parameters are listed in table 3. The source code can be found on the GitHub repository Orden 2020.

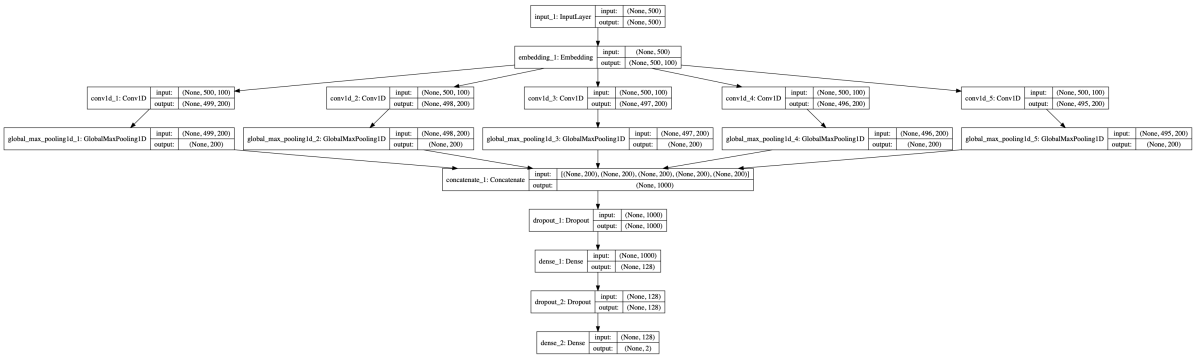

Figure 2: Schematic overview of CNN model.

Name	Possible values	Used value
INPUT	{"tokens", "stemmed", "lemmed"}	All three
BINARY	{True, False}	True
MARK_NEG	{True, False}	False
ALPHA	$\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} \mid x \geq 0\}$	1
MIN_FREQ	$\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} \mid x \geq 0\}$	1

Table 3: The hyper-parameters of the NB classifier.

The first step in training a NB classifier is to create and define the feature set. The hyper-parameter MIN_FREQ determines the minimum frequency that a word has to occur in the corpus to include it in the feature set. Also, the hyper-parameter MARK_NEG determines if nltk's *mark_negation()* is applied before creating the feature set (Bird and Klein 2009). As shown in figure 1, we can use three different pre-processed inputs. The hyper-parameter INPUT corresponds to one of these. We use a unigram feature set with MIN_FREQ = 1, so every word in the training set is a feature. The second step is calculating the maximum likelihood estimators (MLE) and applying Laplace smoothing. If the hyper-parameter BINARY is True, all features weigh the same. If BINARY is False, the weigh of the feature depends on its occurrences in the corpus. The hyper-parameter ALPHA is the smoothing value for the Laplace smoothing. These smoothed estimators make up the classifier.

2.4 Evaluating

We compare the models on the same test set and the ground truth data is obtained from the raw test files from the 'test' folder in the dataset. The *test_sequences* are fed to the fitted CNN model and by using the keras function *predict()*, predictions are generated. The last step is to evaluate both models. The CNN predictions are evaluated by using the *classification_report()* function from scikit-learn (Pedregosa et al. 2011). This calculates the precision, accuracy, recall and F1-score of the model. The NB predictions are created and evaluated by *evaluate_model()* which calculates the same metrics (Orden 2020, see *nb_classifier.ipynb*).

Input	Embedding	Accuracy	Precision	Recall	F1-score
tokens	cbow	0.900	0.900	0.900	0.900
tokens	skipgram	0.890	0.885	0.885	0.890
stemmed	cbow	0.890	0.895	0.895	0.895
stemmed	skipgram	0.870	0.875	0.870	0.875
lemmed	cbow	0.890	0.895	0.885	0.885
lemmed	skipgram	0.870	0.870	0.875	0.865

Table 4: Results of the CNN on the test set.

Input	Accuracy	Precision	Recall	F1-score
tokens	0.8315	0.8737	0.7750	0.8214
stemmed	0.8274	0.8673	0.7730	0.8174
lemmed	0.8271	0.8701	0.7690	0.8164

Table 5: Results of the NB classifier on the test set.

3 Results and discussion

Table 4 and 5 show the results for the CNN and NB classifier, respectively. It is clear that the CNN outperforms the NB classifier in all cases. We use the F1-score to compare the models, because this is based on both precision and recall. The combination of *stemmed* as input and *cbow* as embedding, yields the best F1-score for the CNN. The NB classifier yields the best F1-score with *tokens* as input. In conclusion, the best CNN outperforms the best NB classifier in terms of F1-score by 9.569% (see equation 1). To obtain even higher F1-scores, further research can be done with parameter fine-tuning or using another type of neural network. One good alternative network might be a recurrent neural network (RNN) according to the research of AlSurayyi, Alghamdi, and Abraham n.d.

$$\begin{aligned} &\frac{F1_{CNN} - F1_{NB}}{F1_{NB}} * 100\% = \\ &\frac{0.900 - 0.8214}{0.8214} * 100\% \approx 9.569\% \end{aligned} \tag{1}$$

As shown in table 4 the embedding of the CNN does have a significant effect on the performance. With all inputs, the cbow embedding outperforms the skipgram embedding in terms of F1-score. Further improvements might be achieved by using other types of embedding models, such as Global Vector (GloVe) embedding.

In almost all cases, and for both models, the stemmed and lemmed input scored lower F1-scores in comparison to the tokens input. Therefore, it seems that applying more pre-processing filters to the tokens input, effects the performance of both models in a negative way. Further research could investigate the reason of this negative effect or look into more pre-processing filters such as removing numbers to increase the performance of both models.

References

[13] *Google Code Word2Vec*. `https://code.google.com/archive/p/word2vec/`. 2013.

[AAA] Wejdan Ibrahim AlSurayyi, Norah Saleh Alghamdi, and Ajith Abraham. "Deep Learning with Word Embedding Modeling for a Sentiment Analysis of Online Reviews". In: ().

[BK09] Edward Loper Bird Steven and Ewan Klein. *Natural Language Processing with Python*. O'Reilly Media Inc, 2009.

[Cho+15] François Chollet et al. *Keras*. `https://keras.io`. 2015.

[DG14] Cicero Dos Santos and Maira Gatti. "Deep convolutional neural networks for sentiment analysis of short texts". In: *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*. 2014, pp. 69–78.

[Dis16] Anaconda Software Distribution. *Anaconda Computer software*. `https://anaconda.com`. Version 2-2.4.0. 2016.

[Maa+11] Andrew L. Maas et al. "Learning Word Vectors for Sentiment Analysis". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. URL: `http://www.aclweb.org/anthology/P11-1015`.

[McK+10] Wes McKinney et al. "Data structures for statistical computing in python". In: *Proceedings of the 9th Python in Science Conference*. Vol. 445. Austin, TX. 2010, pp. 51–56.

[Oli06] Travis E Oliphant. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA, 2006.

[Ord20] Thomas van Orden. *NTMI repository*. `https://github.com/thomasvanorden/NTMI`. 2020.

[Ped+11] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[ŘS10] Radim Řehůřek and Petr Sojka. "Software Framework for Topic Modelling with Large Corpora". English. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. `http://is.muni.cz/publication/884893/en`. Valletta, Malta: ELRA, May 2010, pp. 45–50.

[saa19] saadarshad102. *CNN repository*. `https://github.com/saadarshad102/Sentiment-Analysis-CNN/blob/master/Notebook.ipynb`. 2019.

[VD95] Guido Van Rossum and Fred L Drake Jr. *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.