Machine Learning Project 2 - Sentiment Analysis

Filippa Bång, Liamarcia Bifano, Roman Bachmann
{filippa.bang, liamarcia.bifano, roman.bachmann}@epfl.ch
Machine Learning CS-433, EPFL, Switzerland

***Abstract*—The EPFL ML Text Classification is a classification competition in predicting if a tweet originally contained a ☺ or a ☹ . The given dataset has 2.5MM tweets written in English about diverse subjects and it is balanced by category. In order to create a machine learning methodology to classify the tweets, exploratory data analysis, text preprocessing, model selection and tuning of hyperparameters were performed. A solution, where five Convolutional Neural Network models are ensembled by a Random Forest classifier is presented in this report. The words are embedded using either *word2vec* or a combination of *word2vec* and *tf-idf*.**

I. Introduction

Text classification is the subject of classifying text into predefined classes. Machine learning methods trained to automatically classify text can be applied to a variety of classification tasks in a variety of fields. The problem encountered in this project is to classify unseen tweets as positive or negative, by training machine learning models on preclassified data. This is a binary text classification problem of identifying the sentiment in a text. An ensembling of five Convolution Neural Network models with dropout and batch-normalization was used to solve this problem. The word embedding method, the kernel size and the number of channels are differentiating the five models. The ensembling is made using a Random Forest.

II. Models and Methods

Machine learning methods such as *Simple Neural Networks*, *Convolutional Neural Networks* and different versions of *Convolutional Neural Networks with Dropouts and Batch Normalization* were run on this dataset. All the models were optimized based on *Binary Cross Entropy* given by the formula:

$$-\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{i,j}\log \hat{p}_{i,j} \tag{1}$$

where m is the number of categories (in our case $m = 2$), y_{ij} is the i-th observation of the category j and $\hat{p}_{i,j}$ is the estimated probability given by the model. The metric to choose the final model was the prediction *accuracy* and it was evaluated in a randomly selected test dataset containing 1% of the entire dataset, which is 25.000 tweets.

A. Data Analysis

Before continuing to the machine learning models it is beneficial to look into the data in order to extract insights that could be useful for creating the models.

Word cloud is a visual representation of text data where each word's font size is shown in proportion to the number of times that this word appears. In order to build this chart it is relevant to remove all stop words, as they otherwise will be the most frequent words appearing in the chart. The figure 1 contains the word cloud for the *positive* and *negative* tweets. As we can see in figure 1, words like *haha, lol, please, follow* differentiate *positive* tweets from *negative* tweets.

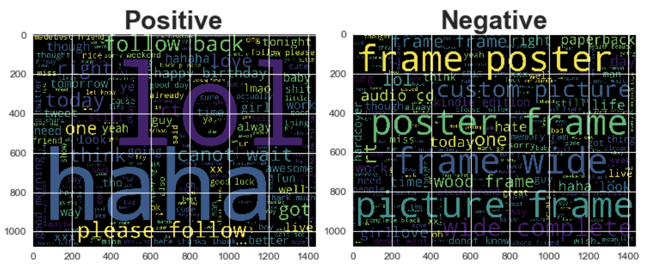

Figure 1: Word Cloud

An additional sentimental analysis was performed using *VADER* (Valence Aware Dictionary and Sentiment Reasoner). This is a tool for sentimental analysis, specially tuned in order to perform well on data from social media. For each tweet there are 4 scores *negative, positive, neutral and compound* where compound is the weight average of *negative, positive and neutral*. The figure 2 has the distribution of each score split by *positive, negative* tweets. According to the charts, *negative* tweets tend to have a slightly less positive score and a more neutral one, but it's not something that seems relevant. More information about the performance and how the scores are calculated can be found in "VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text" [3].

Another hypothesis that was considered is if the writing style might change according to the type of the tweet, for instance, negative tweets might have more adjectives than

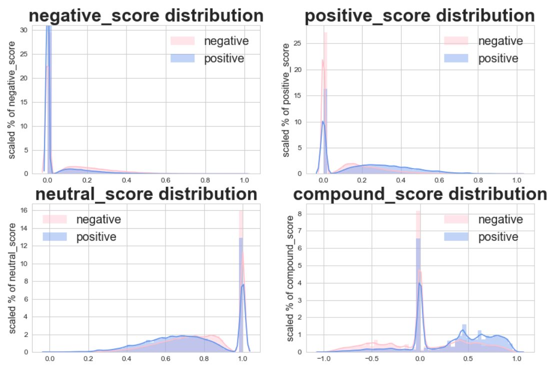

Figure 2: VADER Scores

positive ones. Figure 3 shows the part of speech distribution of the tweets.

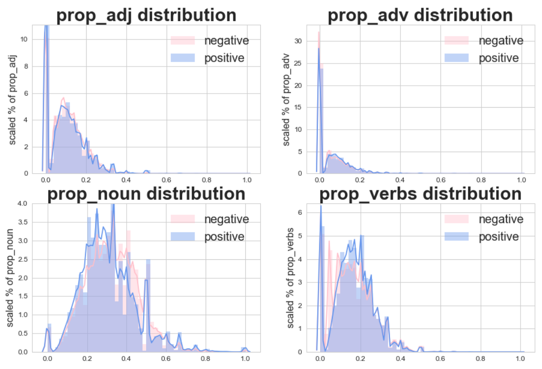

Figure 3: Word's Tags

In order to investigate that, a Random Forest model from the *scikit-learn* library, using features about styling and *VARDER* score, was run in order to predict the tweet type. The model had a performance of 73.2% which is a reasonable performance considering the fact that no encoding or tuning was done.

B. Embedding

Embedding is a method used to transform a vector of words into a matrix representing the meaning of each word in the text. To represent the twitter data as word embeddings, *word2vec* alone and a combination of *word2vec* with *tf-idf* were used. *Word2vec* considers the relative frequency co-occurrence of words and *tf-idf* considers the number of times a words appears in a document relative to all documents, which gives a motion of how important a word is in order to describe the context. Those two methodologies were combined by multiplying *word2vec* and *tf-idf*. The *word2vec* model is trained on all the tweets in the dataset in order to make sure that all words appearing in the training exists in the *word2vec*-vocabulary.

Initially, a simple convolutional network model (Figure 4), inspired by [4], was run with just *word2vec* and afterwards including *tf-idf*.

Another alternative to encode text is the *bag-of-words*. *Bag-of-words* count the number of times a word appears in the tweet and does therefore not consider the context of the word.

C. Preprocessing

Before encoding the text in *word2vec*, many possibilities for preprocessing the text are available in the literature, for example replacement of special characters, removing punctuation, normalizing cases, removing stop words and word stemming.

In the given dataset, all words and punctuations are separated by a whitespace, in each tweet. This is a preprocessing step, that we otherwise would have chosen to perform.

A very simple treatment was done to fix usual contractions in the English language, for instance, *haven't* was split into the two words *have* and *n't*.

In order to observe the result of the preprocessing, a convolutional neural network with one fully connected layer was run after every trial. The network's schema is visualized in the image 4 and the results are shown in Table I.

Method	**Test Accuracy**
Conv 1 layer without stop + punct	77.89%
Conv 1 layer without stop	81.56%
Conv 1 layer without punct	80.85%
Conv 1 layer with stop + punct	84.43%

Table I: Convolutional Neural Network performance for each preprocessing

In contrast to our intuition, removing punctuation and stopwords yielded worse results. One possible explanation is that it removes relevant information expressing the sentiment in the text, for instance, tweets without stopwords and punctuation have around 15% fewer words.

Remove morphological affixes from words by using the method *Snowball* in *gensim* package was as well tested. By doing a heuristic check with a couple of words, *computer, politics, sports*, and its top closest words in *word2vec*, we saw non-intuitive results. The model's performance was as well worse which led to the decision to not use this method in the final model.

Before continuing to hyperparameter tuning and model selection, it is convenient to check with a simple model if the data preprocessing and encoding is reasonable as it is not feasible to do this for every model, considering the computational expensiveness.

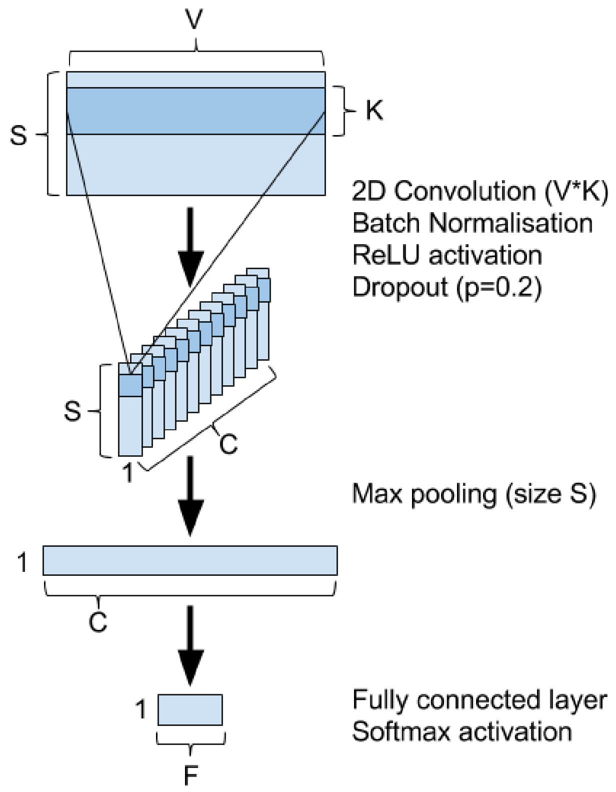

Figure 4: Simple Convolutional Network

D. Modeling Methodology

A Convolutional Neural Network (CNN) is the machine learning model chosen to classify the tweets in this project (see [4], [2], [1]). A CNN was chosen over a regular Deep Neural Network in order to limit the amount of parameters in the network and thereby reducing the computational complexity and avoid overfitting. CNNs also allow to give more importance to words that appear next to each other, which is often highly relevant in natural language processing.

Each tweet that the CNN receives as input is a matrix representation having S rows (number of words in a tweet) and V columns (length of each word2vec vector). Each row in the matrix contains one word represented as a word embedding. The structure of the CNN requires that all the matrices have the same size. As the number of words are not the same for all tweets, the matrices are padded to the length of the longest tweet.

As each of the C convolutional kernels is observing only a subset of the words in each tweet at any given step, where the number of words is decided by the kernel size (K, V), the closeness of the words is taken into account in the training and classification.

Batch-normalization is used in the CNN in order to allow higher learning rates and to receive a regularization effect without loosing as much information as when using a higher dropout percentage. The increase in accuracy when applying batch-norm can be seen in the Table II.

The last output layer has size F; the number of output classes.

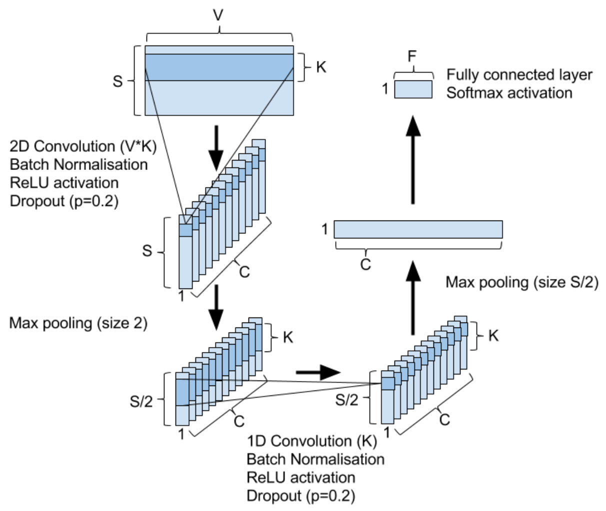

Figure 5: Final Convolutional Network Architecture

E. Model Selection and Tuning Hyperparameters

A neural network has a lot of hyperparameters to be optimized, for example the number of layers, the kernel dimensions for each convolutional layer, activation function for each layer, dropout and regularization term. In Table II, we outline all different architectures that we tried. When adding dropout, the performance was severely reduced, but by applying batch-normalization, it was improving again. By using different kernel sizes, we attempted to capture different kinds of useful information about the tweets. Using kernel size 5 with two convolutional layers, dropout and batch-normalization yielded the best results over long training sessions.

Method	**Test Accuracy 10 Epochs**	**Test Accuracy 20 Epochs**
Sentiment 1: Conv 1 layer, Kernel size 3	85.638%	85.59%
Sentiment 2: Conv 2 layers, Kernel size 3	85.966%	86.03%
Sentiment 3: Conv 2 layers + Dropout, Kernel size 3	79.622%	80.072%
Sentiment 4: Conv 2 layers + Dropout + BatchNorm, Kernel size 3	85.848%	85.546%
Sentiment 5: Conv 2 layers + Dropout + BatchNorm, Kernel size 5	85.51%	86.038%
Sentiment 6: Conv 2 layers + Dropout + BatchNorm, Kernel size 7	85.742%	85.85%

Table II: Hyperparameter Optimization

F. Ensemble Modeling

Ensemble methods are used in machine learning in order to combine different trained models. The main idea behind this technique is that different models can capture different relations and standards in the data, which when combined can yield better results. There are various ways to combine models, for instance taking the median of individual models' output or running a machine learning model with the output from the individual models as features. Usually, the models chosen to ensemble the output are models which do not tend to overfit the data, for instance *Random Forest*, *Ridge* or *XG-Boosting*. For this project, we chose to train a *Random Forest* and its hyperparameters *maximum tree depth* and *number of estimators* were tuned with *RandomizedSearchCV*. As *Random Forest* is an example of a bootstrap aggregating algorithm, it counteracts overfitting by reducing the variance. We also chose to train our best model (Model 2) multiple times (Table III) and stabilize its output by taking the median of all predictions.

III. Results

Five CNNs are trained using a variation of hyperparameters (see Table III). Model 2 was trained ten times, as it often yielded very good results on the test data and on Kaggle and combined to one prediction by taking the median. The predictions of the five models were then ensembled into a Random Forest Model. The schema of the CNNs is described in 5 and all of them were trained using:

$$\text{num\_epochs} = 20$$
$$\text{batch\_size} = 100$$
$$\text{learning\_rate} = 0.001$$
$$\text{dropout probability} = 20\%$$

The CNNs are fed with data with varying preprocessing and different kernel sizes to extract information from the tweets at different scales. The table III below shows how each CNN is structured and displays its performance.

The ensembled model, using the tuned Random Forest with 71 estimators and a maximum tree depth of 61, gives us a performance of 87.38% on Kaggle.

CNN	Preprocessing	Kernel size	#Channels	Test Accuracy
Model 1	word2vec	3	256	86.06%
Model 2.0	word2vec	5	256	85.744%
Model 2.1	word2vec	5	256	85.664%
Model 2.2	word2vec	5	256	86.004%
Model 2.3	word2vec	5	256	85.76%
Model 2.4	word2vec	5	256	85.6%
Model 2.5	word2vec	5	256	85.188%
Model 2.6	word2vec	5	256	85.584%
Model 2.7	word2vec	5	256	86.34%
Model 2.8	word2vec	5	256	85.816%
Model 2.9	word2vec	5	256	85.868%
Model 3	word2vec	7	256	85.22%
Model 4	word2vec + tf-idf	3	256	82.65%
Model 5	word2vec	5	128	85.964%

Table III: CNNs hyperparameters and performance

IV. Discussion

A majority of the preprocessing steps tested on the dataset gave lower prediction accuracy for our models than using unprocessed data. This could depend on the usage of word2vec for embedding the words, and its consideration of the context. The decrease in accuracy after removal of stopwords, could be due to the definition of stopwords in the library used, i.e. NLTK. The words *no* and *not* are for example considered as stopwords and therefore removed, which can impact the training of the model. A possible improvement could therefore be to use a modified version of the stopwords library, where negations aren't considered as stopwords.

The use of a CNN yielded good results and could capture the sentiment of tweets quite well. However, due to the max pooling operation, we loose information about where exactly in the sentence negative or positive parts are located, which might make a difference for the overall sentiment of the tweet. It would be interesting to try solving this issue in a next step.

V. Summary

The model yielding the prediction score in the Text Classification competition on Kaggle uses Random Forest as an ensemble model to combine different runs of the CNN. As showed in table III each CNN has a unique combination of kernel size, number of channels and embedding strategy. Dropout with probability 0.2 and batch-normalization are implemented in all CNN models, counteracting overfitting.

References

[1] Jan Deriu. *Sentiment Analysis using Deep Convolutional Neural Networks with Distant Supervision*. Department of Computer Science, ETH Zürich, 2016.

[2] Jan Deriu et al. "Leveraging Large Amounts of Weakly Supervised Data for Multi-Language Sentiment Classification". In: *CoRR* abs/1703.02504 (2017). arXiv: 1703.02504. URL: http://arxiv.org/abs/1703.02504.

[3] C.J. Hutto and Eric Gilbert. *VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text*. Georgia Institute of Technology, Atlanta, 2014.

[4] Yoon Kim. "Convolutional Neural Networks for Sentence Classification". In: *CoRR* abs/1408.5882 (2014). arXiv: 1408.5882. URL: http://arxiv.org/abs/1408.5882.