CS-433 Machine Learning - Sentiment Analysis

Mortiniera Thevie, Petrescu Diana, Wagner Patrik

***Abstract*—Social medias becoming one of the main platforms to exchange or express beliefs and feelings, this led to an increase in the demand of automated tools in this area for business purpose. In this paper, the task we want to solve is a problem of discrete binomial supervised classification. More precisely, given a data set of tweets, we want to classify them based on the positive or negative sentiment they express. We apply several machine learning models in order to train the model. We then test it on a new data set (provided in the corresponding CrowdAI competition) and achieved an accuracy of 85.4% with Logistic Regression. However, some additional techniques like CNNs could outperform this result and should be further explored.**

I. Introduction

Opinion Mining[1], also called Sentiment Analysis, is a type of Natural Language Processing (NLP) for tracking the mood of the public about a particular product. Being able to identify this kind of information in a systematic way allow, for example, to identify which demographics like or dislike particular product features. With the emergence of Deep Learning and NLP, Sentiment Analysis has become an important topic of interest in Machine Learning. Social medias becoming one of the main platforms to exchange or express beliefs and feelings, this led to an increase in the demand of automated tools in this area for business purpose. In this paper, the task we want to solve is a problem of discrete binomial supervised classification. More precisely, given a data set of tweets, we want to classify them based on the positive or negative sentiment they express. We propose a twitter-based solution implemented in Python with the use of numerous of its libraries. Among the principal ones, we use TensorFlow, Keras, Word2Vec, Gensim and Scikit-learn. Moreover, regardless of the method choice, there are three main important stages in the process. The first one is preprocessing of the data set and is crucial for the next steps to get a good accuracy. After having a cleaned data set, there is the Embedding tweets part that turn tweets into digital vectors either word by word (like Word2Vec) or tweet by tweet (like text2Vec). The last part is the actual learning using various ML approaches.

II. Preprocessing

A. Data Description

The full training data set is composed of 2.5 millions labelled tweets , half of them used to contain a positive :) smiley while the other half contains a negative one :(. Additionally, 10'000 unlabelled tweets are used for prediction. Some parts of the data was already "cleaned" for example by replacing users and urls by '¡user¿' and '¡url¿'. Exploratory analysis allows to have more insight on the properties of the data set. For example, on the data set provided, we can notice that most of the tweets consists of very informal words and phrases, usually more common in speech than writing. This can make pattern recognition more difficult. Furthermore, the presence of some characters or repetition of punctuation usually not used in formal writing may denote particular or meaningful emotion. Therefore, we need to pay a particular attention to those cases during the preprocessing task.

Furthermore, some tweets have duplicates in the data set (due to retweets or spams for example) and we need to get rid of them in order to remove redundancy.

Moreover, in order to speed up choices of implementations, we first test our approaches on a subset (containing from the positive and negative data set of tweets) of the full data set consisting of 200k tweets extracted from the whole training set.

B. Data Preprocessing

Preprocessing is a crucial step for the accuracy of the implementations. Preprocessing steps for NLP is widely discuss and there is no clear guideline to preprocess a data set for sentiment analysis. Several startegies can be tested and compared but in the end the choice is usually the ones that provide the best accuracy with the chosen methods.

Several well-known techniques are lemmatization, handling negation, remove duplicate chars, etc. [2]. We did some testing and finally in this experiment we process the data as follow:

- We first handle **contraction** specific to English language, by expanding them in two. As an example, "can't" become "can not".
- We notice that some end characters in words and punctuation are **repeated** like "yesssssss" or "!!!" and that it can emphasize a particular emotion. Therefore, for more homogeneity ant to empathize its relevance in sentiment analysis, we replaced the repeated character with a tag.
- We decide to get rid of **numbers** in the data set as we think that they do not carry any particular meaning in sentiment analysis. Hence, we choose to replace each number by particular tag indicating the presence of a number.

- Although the target **emojis** (namely the positive smiling ":)" emoji and the negative ":(" emoji) are not part of the tweet, other ones are still present. Considering their particular relevance for our task, we decide to replace them by a word describing the most their meaning. We manually create a list of emojis with their corresponding word from a list taken in Wikipedia[3].
- While lot of words may appear to be neutral, others have a significant emotional weight. Hence, we decide to handle them in a similar way as with emojis by looking up into a list of **meaningful positive and negative words**[4].
- Since different forms of a word often communicate essentially the same meaning, we use **lemmatization and stemming**[5] techniques to reduce inflectional forms of different lemmas which can contribute to noise and in doing so diminish our performance.
- Finally, as mentioned, more observations are made but not used. We have tried to implement stopwords removal since they do not carry emotional weight. However, as surprising and counter-intuitive as it may seem, it lowered our performance. Moreover, hashtags, particularly important in the context of tweets, can, unlike stopwords, carry meaningful emotion. However, as they were already to lower case, we considered it an ambiguous task to decide where to split them. Furthermore, replace elongation [6] haven't significantly changed the accuracy of our models. Finally, the presence of the tags "url", "rt" and "user" do not appear to affect the performance of our classifiers.

III. Embedding Tweets

We tried different techniques to find a good feature representation of the tweets as numerical data :

- As a baseline for the first approach, we use the **Bag-of-Words**[7] (BoW) representation. First of all we vectorized tweets using CountVectorizer from scikit-learn library which just converts a collection of text documents to a matrix of token counts. However, since order does not matter to this vectorizer, it misses semantic representation of text. In order to tackle this, we used, as an alternative, the **n-gram model** to preserve spatial information. Finally, an other issue in the BoW model is that only providing raw count information tend to score very frequent words (e.g stopwords) better, which indicates that high raw count does not necessarily mean that the corresponding word is more important. To address this issue, one of the most popular ways to normalize the term frequencies is to weight a term by the inverse of document frequency, or **TF-IDF**. One of the main characteristics of IDF is that it weights down the term frequencies while scaling up the rare ones. Hence, it allows us not to target widely spread words independently from having significant sentimental meaning, but to extract meaningful ones, relevant on a particular class. In our settings, we use **tri-gram** without removing stopwords.
- Convolutional Neural Networg (CNN) with Word2Vec [8]: In short, in this method, after the preprocessing, the embedding is realized using Word2Vec library and the learning process is based on a neural network thank (a Sequential model with Embedding layers from keras library).
  We can mention that the supervised (labeled) data were randomly and appropriately shared in training data, validation data and test data.
  For the embedding, we associated each word to a vector but also each tweet to a sequence.
  Since tweets can contain any number of words and the neural network needs data of same size, padding was used by choosing a single maximum length that is large enough (based on training data tweets).
- We also tried a third approach with the new Universal Sentence Encoder which encodes text into high dimensional vectors that can be used for text classification, semantic similarity, clustering and other natural language tasks. The model is trained and optimized for greater-than-word length text, such as sentences, phrases or short paragraphs. The input is variable length English text and the output is a 512 dimensional vector [9]. We use the "universal-sentence-encoder/2" for tweet analysis.

IV. Classification Processes

After transformation of textual data to numerical ones during the feature engineering, we need to apply classification. We experiment several machines learning techniques such as Logistic Regression Random Forest Classifier, Neural Networks and Support Vector Machines (SVM). The number of features being small in comparison to the number of training samples we choose to use a linear kernel for SVM.

A preliminary experiment allowed to us to reduce our area of investigation by filtering out some models.

We decide first to filter out Random Forest which, for lower results as others on the first tests, was penalized due to its unstability in performance and very long computing time issues.

In the litterature, it is said that Logit Model and SVM with linear kernel perform comparably in practice. Logit Model optimizes the log likelihood function with probabilities modeled by. the sigmoid function while SVM tries to find the widest possible separating margin between two classes. The former is more sensitive than the latter to outliers because its cost function (logistic loss) diverges faster than those of SVMs (Hinge Loss). However, Logit model produces probabilistic values while SVM produces discrete outcomes (0 or 1). Hence, it does not make absolute

prediction assuming that data is enough to give a final decision. In order to compare which model is more suitable for the presented data set, we decide to keep them both.

Hyperparameters were tuned running a randomized grid search along with 5-fold cross validation to narrow down the area of the regularizers used to avoid over-fitting. Then we run a grid search in the neighborhood of the best seen parameter(s) to find the best one(s).

Namely for Logit Model, the inverse of regularization strength C is choosen among $[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]$ and we try between $L1$ (Lasso regularization) and $L2$ (ridge regression) penalties. In our settings, $L2$ penalty and $C = 6$ is chosen.

As for Linear SVM, we use $C = 0.1$ along with $L2$ penalty.

Moreover, for the CNN with Word2Vec approach we can say the following things. The Word2Vec library offers two basic models, namely the Continous Bag Of Words (CBOW) model and the Skip-Gram (SG) model. In order to take advantage of these two approaches, for each word two vectors with the same dimensions (for example 100 or 200 features) were calculated using each of the two mentioned models. Then, by concatenation, we obtained a single vector per word with a double dimension (for example 200 or 400 features).

Furthermore, only 100000 most frequent words were considered in the training set.

Once the initial embedding realized, it was necessary to decide to transmit it to our neural network by means of an Embedding layer of Keras. We took into account three strategies:

the embedding obtained by Word2Vec is included in the neural network which does not modify it anymore (static word vectors provided by Word2Vec);

the Embedding layer learns itself and calculates the embedding from scratch;

the Embedding layer receives the initial embedding obtained by Word2Vec and improves it during the learning process (initial words vectors are updated through training).

Concerning the neural network (NN), a 1D Convolutional Neural Network (CNN) was used with filters (matrices) with the column width having the same value as the data column with.

As for the other dimension of the filter matrix (the kernelsize), it specifies what kind of n-grams is taken into account. In a first model, we consider bigrams and in the most complete model a combination of 2-, 3- and 4-grams.

We have used 100 filters which have been completed, in an improved variant, by a Global Max Poling layer which extracts the maximum value from each filter and which is connected in output to a fully connected layer.

With the training data, for the model with bigrams filters, the scenario 3) proved to be the best because the model 2) had the tendency of overfit to the trainig data especially when one advances in the epochs. The comparison criterion was the accuracy.

A last improvement was the one already mentioned by a combination of 2-, 3- and 4grams and we used the latter model to train and then use it to classify (label) the new tweets.

Possible improvements:

define a deeper structure with more hidden layers

try different pool size to see how the performance differs.

Finally, we also try a training with a combination of Convolution Neural Network and Bidirectional Long Short Term Memory Network is used (CNN-LSTM) with Keras for the Universal Sentence Encoder embedding tweets [10]. However, as it takes much longer to run than the approach mentioned above with the CNN and as the increase in accuracy was really slow, we decided not to continue investigating this approach.

V. Results

The results obtained both in the local test and in the CrowdAI competition are reported in the tables below. When looking at the results, in the smaller local validation set, Linear SVM with $C = 1$ obtained comparable results than Logit Model with $penalty = L2$ and $C = 10$, which was expected as stated during the model comparisons.

However, on more training examples with the full training set, Logit Model outperformed Linear SVM in the test from CrowdAI competition.

Model	Logit Model	Linear SVM
Accuracy	83.04%	83.08%

Results obtained on a local validation set

Model	Logit Model	Linear SVM
Accuracy	85.4%	84.6%

Results obtained on CrowdAI competition

Moreover, unfortunately, with the second approach described above (Word2Vec + CNN), we haven't had the time and the power to run it till completion to test the results in the CrowdAI competition. However, for example, with the raw (not preprocessed) smaller data set, we obtain a validation accuracy (accuracy completed on the validation set and not seen by the model) of 84.35% for the best epoch. For the full preprocessed data set, we have only run a single complete epoch and had obtain a good validate accuracy of 84.4% and an accuracy of 83.33%. Usually, the accuracy increase in the further epoch and the second or the third is usually the best one (with the highest validation accuracy) because after there is over-fitting and the accuracy increases but the validation accuracy decreases.The accuracy at the second epoch at one third of the epoch time reached 85.40% but we haven't time to let it run anymore. However, it seems a quite interesting approach that could give us even better results than the one obtained.

VI. Conclusion

The investigation revealed how important data preprocessing and feature engineering are, discriminating bad models from a good ones. This is verified during this study, when running Logit Model with hyperparameters tuning on almost raw data achieved a score of 0.812% on CrowdAI whereas after passing through multiples transformations we achieved 85.4%.

An other important fact to notice is the running time. While our best model, namely Logit Model, run in less than one hour on the complete data set (with CPU only), more complex and advanced models such as Convolutionnal Neural Networks which tends to give better results (achieved 85% on the local data set already) take multiples hours for hyper parameters tuning on the smaller set only, and so even more hours on the full data set. This is a very know compromise in machine learning where computing time and performance often collapse. Hence, one might chose a faster model but with less performance.

As a conclusion, while we achieved great results, the model still need to be update with new data and some improvement need to be done. Today computer still have issues to deal with irony which can contain both positive and negative sentiment, which is manageable for a human by analyzing sentences one at a time, not yet for a computer. However, the more informal the medium like social medias (twitter or blog posts for example), the more likely people are to combine different opinions in the same sentence.

In a future work, provided with more time at our disposal, we would like to investigate more with CNN and transfer learning [11] which looked really promising.

VII. Libraries used in the analysis

Here are some main libraries used for our analysis: TensorFlow, Keras (open source neural network library capable of running on top of TensorFlow), Word2Vec, Gensim, Scikit-learn, Pandas, Numpy, Matplotlib, Random, Sqdm, Nltk, Spacy.

References

[1] M. R. et Ian Barber. opinion mining (sentiment mining). [Online]. Available: https://searchbusinessanalytics.techtarget.com/definition/opinion-mining-sentiment-mining

[2] D. E. et al. A comparison of pre-processing techniques for twitter sentiment analysis. [Online]. Available: https://www.researchgate.net/publication/319178537_A_Comparison_of_Pre-processing_Techniques_for_Twitter_Sentiment_Analysis

[3] Wikipedia. List of emoticons. [Online]. Available: https://en.wikipedia.org/wiki/List_of_emoticons

[4] M. Hu and B. Liu, "Mining and summarizing customer reviews," *Department of Computer Science University of Illinois at Chicago*, 2004. [Online]. Available: https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html

[5] H. Risueo. What is the difference between stemming and lemmatization? [Online]. Available: https://blog.bitext.com/what-is-the-difference-between-stemming-and-lemmatization/

[6] zed. replace-elongated-words.py. [Online]. Available: https://gist.github.com/zed/9616954

[7] Wikipedia. Bag-of-words models. [Online]. Available: https://en.wikipedia.org/wiki/Bag-of-words_model

[8] T. R. Ricky. Another twitter sentiment analysis with pythonŁŁpart 11 (cnn + word2vec). [Online]. Available: https://towardsdatascience.com/another-twitter-sentiment-analysis-with-python-part-11-cnn-word2vec-41f5

[9] universal-sentence-encoder. [Online]. Available: https://tfhub.dev/google/universal-sentence-encoder/1

[10] jatinmandav. Sentiment analysis on twitter data using universal sentence encoder (tensorflow) in keras. [Online]. Available: https://github.com/jatinmandav/Neural-Networks/blob/master/Sentiment-Analysis/universal-sentence-encoder/universal_sentence_encoder_sentiment-analysis.ipynb

[11] A. Deshpande. A beginner's guide to understanding convolutional neural networks part 2. [Online]. Available: https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/