Machine Learning Course: Project 2

Peter Krcmar, Anton Ragot and Robin Zbinden
Machine Learning (CS-433), School of Computer Science and Communications Sciences, EPFL, Switzerland

***Abstract*—Twitter is a social network where users can post short messages limited to 280 characters called "tweets". This restriction forces the users to be succinct and to compress its ideas to keep only the essential. Therefore, a single tweet can still capture a lot of information. Machine learning is very useful to get that information and analyze the sentiment of a given tweet.**

Through this work, we aim to train a classifier to predict if a tweet message used to contain a positive :) or negative :(smiley, by considering only the remaining text. We propose several approaches using different machine learning algorithms. We compare these distinct approaches to find out which one produces the most accurate classifier and the best predictions. We conclude that a Convolutional Neural Network obtains the best predictions on AIcrowd, with an accuracy of 0.869.

I. Introduction

Sentiment analysis is defined as being the extraction and the study of the subjective information contained in a text trough data mining and natural language processing. It has many applications such as helping recommender systems to recognize which product will please to a given customer, or in social sciences to interpret the behavior of a population.

One of the common task of sentiment analysis is to classify the polarity of a given text, i.e., determine if the opinion expressed is more positive or negative. However, for short texts as tweets, it is not always trivial even for humans as we show in subsection VI-B. Some subtleties like irony are sometimes difficult to understand.

In this work, the task is slightly different. We classify the polarity of a given tweet by looking if it contains a positive :) or a negative :(smiley. We are aware that a positive smiley does not always means positivity of a tweet [**?**], but this is an effortless manner to label huge amount of tweets. Thus, the dataset consists of 2,500,000 tweets, with the same number of positive and negative tweets [1].

II. Data pre-processing

Tweets, while having some common patterns, are often very messy. We observe a huge amount of words and many of them appear scarcely. This can be seen in figure 1. In this section, we explain the steps we take to normalize the dataset as much as possible without discarding some important information.

1) **Splitting the Hashtags**
   "#" symbols are widely used on Twitter and tell a lot about a tweet. To process them, we first add a token *<hashtag>* and then split the words contained in the hashtag using word frequency to know where to split. E.g., "#iloveyou" becomes "<hashtag> i love you".
2) **Replacing important symbols**
   We notice that three special symbols (*!,?,<3*) appear more often in positive tweets. We replace them by tokens. E.g., "do you <3 me ? !" becomes "do you <heart> me <question> <exclamation>".
3) **Removing punctuation**
   Some symbols (*#"$%&'()*{}/*+-,.:;=@_^|\~*) are not important for sentiment analysis, we simply remove them from the tweets.
4) **Replacing numbers**
   From the dataset, we observe that numbers appear more often in negative tweets. We replace them by a token *<number>*. E.g., "3538 3773" becomes "<number> <number>".
5) **Replacing repetitions of words**
   To emphasize on a idea, users often repeat a single word in a row. In this case, we keep only one occurrence of the word and add the token *<rep>*. E.g., "i love you very very very much" becomes "i love you very <rep> much".
6) **Replacing repetitions of letters**
   Some characters are sometimes repeated to accentuate a word. If the character is repeated three or more times, we keep only one occurrence of the character and add the token *<elong>*. E.g., "i looove youuuu" becomes "i love <elong> you <elong>". Note that we don't handle cases where the letter should be repeated twice (e.g. kazoooo should be changed to kazoo, not kazo). This could be addressed by looking up both versions in a dictionary, however special cases arise when both version are valid words (e.g. beeets can be beets or bets, which are both valid).
7) **Lemmatizing**
   We lemmatize the words of the tweets to convert them to their base form. E.g., "he likes the teaching assistants" becomes "he like the teach assistant".

III. Text representation

Most machine learning techniques cannot take as input raw text, but require numerical-value inputs of fixed size. This means converting the tweets into a valid format, which can be achieved by multiple methods. We explore more classical representations such as bag-of-words as well as supervised and unsupervised embedding techniques.

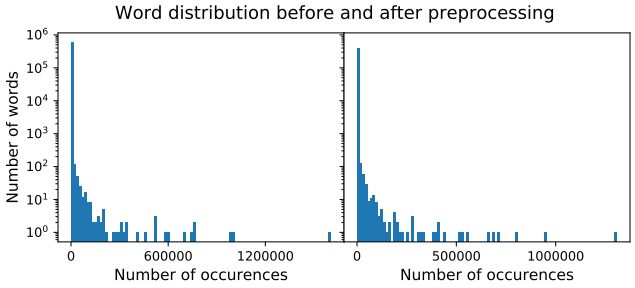

Figure 1. The distributions look very similar. Most words are not frequent. However, our processing techniques reduce the vocabulary from 592k unique words to 390k.

A. Bag-of-words (BoW)

This method represents a text as a set of k words, which forms its vocabulary. Each word w_i has a unique index and each document (i.e. tweet) is represented by a vector of length k in which the i-th entry contains the number of occurrences of the word w_i in the document. Unfortunately, important information such as word order is lost. For instance, the BoW representation of *"Alice is not happy she is miserable"* doesn't capture which of the two adjectives is negated by the *"not"*. This can be addressed by extending the vocabulary with groups of n adjacent words, called ***n*-grams**. In the previous example, adding 2-grams would add *"not happy"* to the vocabulary, showing the dependency between the two words. To further improve the BoW representation, it is possible to re-weight the vocabulary. The idea is to give less importance to words (or n-grams) that appear very frequently in all tweets and more importance to those that are unique. This approach is called *term frequency–inverse document frequency* (**TF-IDF**) weighting.

B. Word embedding

Another way of representing text is learning a representation for each word. It takes the form of a d-dimensional vector of floating point numbers which is referred to as the word's *embedding*. A tweet can then be represented as the aggregation (e.g., the mean) of the word embeddings that compose it. The explored techniques of obtaining such word representations are listed below.

1) **Global Vectors (GloVe)** is an unsupervised technique developed at Stanford where training is performed on aggregated global word-word co-occurrence statistics from a corpus. [2]
2) **Continuous bag of words (CBOW)** is part of the *word2vec* project by Google. It's an unsupervised technique which tries to predict a target word from its context, i.e., the surrounding words.
3) **Skip-Gram** is also part of the *word2vec* project. As opposed to CBOW, it tries to predict surrounding words given a word. [3]

Various implementations exist for each technique, in different programming languages. The **word2vec** library as well as Facebook's **fasttext** offer cbow and skip-gram implementations. Fasttext also offers a supervised version for learning sentence embeddings. **Sent2vec** is an unsupervised extension of fasttext, and also offers a cbow-c+w-gram mode, which also takes into account lexical semantics such as sub-words (e.g. suffixes) during training.

We compare the different techniques by generating embeddings using similar parameters (e.g., 200 dimensions) and feeding them into a model trained with logistic regression. We then check the model's accuracy on a local test set. The results can be found in I.

Table I
ACCURACY OF DIFFERENT EMBEDDING TECHNIQUES USING LOGISTIC REGRESSION

Embedding	Logistic regression accuracy
GloVe	0.7425
word2vec (cbow)	0.6957
word2vec (skipgram)	0.6753
fasttext (cbow)	0.7351
fasttext (skipgram)	0.7287
fasttext (supervised)	0.8080
sent2vec	0.7679
sent2vec (cbow-c+w-ngrams)	0.7622

We realize that the supervised technique yields better results, which is predictable. Indeed, it has more knowledge about the data than unsupervised techniques. However, it may overfit more easily.

IV. Models and Methods

A. Traditional machine learning

Now that we have a valid input, we can find out which machine learning algorithms produces the best accuracy to predict the polarities of the tweets. In this subsection, we will first consider the non deep learning algorithms:

1) **Naive Bayes**
2) **Logistic Regression**
3) **Linear Support Vector Classifier (SVC)**
4) **Decision Trees**

We train a model for each of these algorithms and compute the predictions each time using the TF-IDF text representation. The accuracies obtained are summarized in table II.

Table II
ACCURACY OF DIFFERENT CLASSIC MACHINE LEARNING ALGORITHMS

Algorithm	Accuracy
Naive Bayes	0.8890
Logistic Regression	0.8821
Linear SVC	0.9307
Decision Trees (depth=8)	0.6927

B. Deep learning

Deep learning techniques can have better performance than traditional machine learning techniques [4] when having large amount of data.

In this subsection, we will compare different layers that can be used to create a model:

1) **Long Short Term Memory (LSTM)** is a recurrent neural network architecture capable of learning long-term dependencies. It only preserves information from the *past*.
2) **Bidirectional LSTM** is one unidirectional LSTM as in 1) combined with another LSTM that runs backward. It enables this architecture to preserve information from the *past* and the *future*.
3) **Gated Recurring Unit (GRU)** is related to LSTM as they are both used to preserve information, but GRU are much simpler (hence less accurate on longer sequences) and more computationally efficient.
4) **Convolution Neural Network (CNN)** is an architecture which extracts features using convolutions. The features are then passed on to one or more fully connected layers.
5) **Convolution Neural Network (CNN) + GRU** is an architecture [5] which combines the feature extraction of the CNN layer with the learning ability of the GRU.

To be able to use these layers, we first need to embed the tweets. We can either use the Keras library to train an embedding layer [6] while training the complete model, or use the embedding we created in III-B.

The **keras embbeding** is easier to use as an API call will create a fully functional word embedding where the input is just the pre-processed tweets. One major drawback is that it adds a huge amount of trainable parameters to the model, which slows down the training. On the other hand, learning and fine tuning the embedding during training can lead to a much better embedding, if one is careful not to overfit the data.

To create the **custom embedding layer**, we use the best model from table I: a *supervised fasttext model*. From this model we define an **embedding matrix**. The embedding contains a word's vector representation in its row, for every word in the model's dictionary. To make the tweets usable by the Keras model, the following is applied to each tweet: each word is replaced by its row number in the embedding matrix then padded to match the length of the longest tweet.

Hence, for a small example, the complete pipeline for creating the input would be (length of the longest tweet would be 5 here):

$$\text{"I love pizza"} \Rightarrow \begin{bmatrix} 123 \\ 2 \\ 346 \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 123^{rd} \text{ row of emb. matrix} \\ 2^{nd} \text{ row of emb. matrix} \\ 346^{th} \text{ row of emb. matrix} \\ 0^{th} \text{ row of emb. matrix} \\ 0^{th} \text{ row of emb. matrix} \end{bmatrix}$$

For each of the layers cited above, we try both embedding method to see which one is the most efficient. We obtain the following results:

Table III
ACCURACY OF DIFFERENT DEEP LEARNING TECHNIQUES

Algorithm	Accuracy
LSTM	0.8476
Bi-LSTM	0.8486
GRU	0.8511
CNN	0.869
CNN+GRU	0.865

Due to the LSTM's inherent property of capturing patterns in sentences using "memory", we are surprised that they give a lower accuracy than CNN networks.

C. Most accurate classifier

Using traditional machine learning, we obtain good accuracies, especially with the linear support vector classifier. However, this classifier seems to have difficulty to adapt to new data, i.e. the test set from Aicrowd. Therefore, it is the Convolutional Neural Network using the embedding from Keras which gives the best accuracy on AIcrowd. The associated architecture is depicted in figure 2.

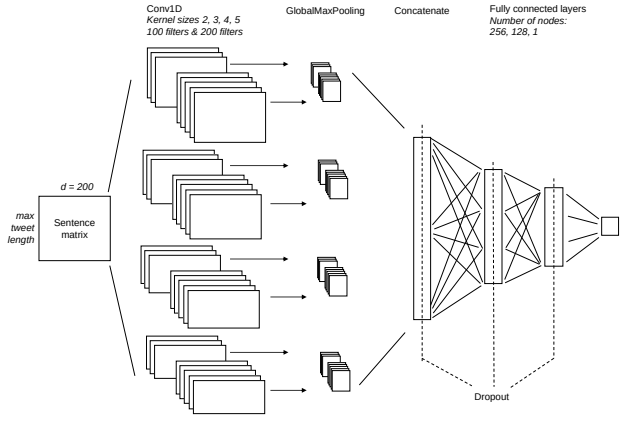

Figure 2. Our final architecture. The embedding layer is omitted for clarity.

Following the embedding layer, we have multiple one-dimensional independent "branches" of convolution layers with kernel sizes of 2, 3, 4 and 5. The branches represent 2-grams, 3-grams, 4-grams and 5-grams. Each branch is composed of 2 sub-branches, generating 100 and 200 filters.

The most meaningful ones are then selected using global maximum pooling and concatenated into a single large vector. This vector is fed into a 3-layer fully connected network.
In order to avoid overfitting, we add l2 regularisation to the embedding layer. Furthermore, we add dropout layers before each fully connected layer. Finally, we monitor the loss on a validation split and adapt our learning rate accordingly, i.e. reduce it by half when reaching a plateau, in order to fine tune our model. The effect of the adaptative learning rate can be seen in figure 3.

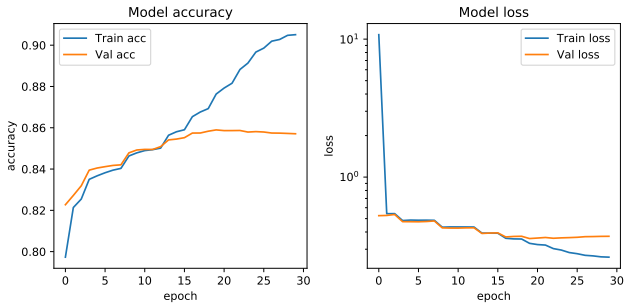

Figure 3. Accuracies and loss during our training. Note that the staircase shape is due to our adaptative learning rate.

V. Results

Finally, after applying all the stated steps, we have a fully trained model. When testing our model on the AIcrowd platform, we obtain an accuracy of 0.869.

VI. Discussion

We are satisfied by this accuracy of prediction, but we recognize that it is possible to do slightly better.

A. Resources

To achieve a better accuracy, we would need more of two particular resources: time and computational resources. The first one because there is a tremendous amount of different techniques, each with multiple parameters, to accomplish this task. Test them all takes a lot of time (for us and the machines) and require often cross validation to find the best combination of hyperparameters. Therefore, It needs a lot of computational resources. State-of-the-art machine learning algorithms require highly effective GPUs and even with the help of *Google colab* [7], we cannot run them with the parameters which we would like.

B. Human classifier

However, it stays interesting to compare the accuracy of our classifier to what a human being can achieve doing the same task of prediction. Thus, we sample 300 random tweets of the validation set and divide them equally between each writer of this report. Then, we try to determine if the tweets are positive or negative. We obtain an averaged accuracy of 0.7 which is considerably below the accuracy of our classifier. It could mean that the machine does better in classifying polarity of tweets than human beings.

It could also show that there is some inherent upper (lower) bound on what accuracy (error) we can achieve. Indeed, some tweets are really hard to classify either by a human or by a machine, e.g., the tweets ”gonna be a fun night writing essays about northern irish political parties” and ”satnite with <user> by phone .. pdhal mggu kmrn satnite brg” contain both negative smileys. We could detect some irony in the first one, but classify the second one is close to pure guess.

VII. Conclusion

The largest difficulty we encountered was selecting the architecture. At each step of the project, there is an overwhelming amount of techniques, models and parameters available. We had a hard time deciding which ones to focus on, and how to combine them (e.g. which embedding to use with which model). Since no model seem to stand out particularly, we worked on multiple models in parallel. It would have been interesting to fully commit to a single architecture and try to get the most out of it. Furthermore, the black box nature of neural networks rendered our work challenging and unintuitive. Indeed, we were surprised that a CNN was giving the best predictions. We are convinced that this work is not perfect but we are satisfied with the result and we definitely learned a lot!

Acknowledgements

The authors would like to thank all the staff from the Machine Learning Course to have made this project possible.

References

[1] “Text classification datasets.” [Online]. Available: https://github.com/epfml/ML_course/tree/master/projects/project2/project_text_classification/Datasets

[2] 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

[3] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” 2013. [Online]. Available: https://arxiv.org/pdf/1310.4546v1.pdf

[4] S. Mahapatra, “Why deep learning over traditional machine learning?” 2018. [Online]. Available: https://tinyurl.com/r7nr54p

[5] D. Nguyen, K. Vo, D. Pham, M. Nguyen, and T. Quan, “A deep architecture for sentiment analysis of news articles,” 06 2017, pp. 129–140.

[6] “Embedding.” [Online]. Available: https://keras.io/layers/embeddings/

[7] “Google colab.” [Online]. Available: https://colab.research.google.com//