Tweet Sentiment Classification

Sarah Antille, Lilia Ellouz, Zeineb Sahnoun
CS-433 Machine Learning, EPFL, Switzerland

***Abstract*—This paper reports our approach to address the problem of text classification on a data set consisting of tweets as part of the Machine Learning course taught at EPFL: based on the already labeled dataset, we use natural language processing techniques and try out different machine learning models to make the most accurate prediction of whether the tweets express a happy feeling or a sad feeling of the users.**

I. Introduction

The provided data set consists of 2.5 million tweets separated into two files: tweets that express a happy feeling (originally contained happy emoticon) and tweets that express a sad feeling (originally contained sad emoticon). The task at hand is a supervised learning classification where we want to predict if a tweet originally contained a positive smiley `:)` or a negative smiley `:(`.

To this end, we use different natural language processing techniques to preprocess the tweets' text and create word embeddings that we will feed to our classification models. We then apply different machine learning algorithms, tune their respective hyper parameters and compare their accuracy on the validation set. We mainly use basic classification models before moving on to different architectures of neural networks. Finally we use an ensemble method that chooses the final prediction based on a majority vote of our best performing models.

The following sections explain our methodology in more details.

II. Exploratory Data analysis and Data Preprocessing

A. Data exploration and cleaning

We start by checking if the dataset provided is balanced: it is indeed the case, as we have 1.25 million tweets in each file. We then use boxplots (figure 1) to visualize the lengths of the tweets in our dataset. We see that tweets expressing sadness tend to be longer than the ones expressing happiness. From the same plot, we also notice the presence of outliers: after further exploration of those tweets it looked like some html formatting was present. We can hence move on to cleaning the tweets' text. Here are the main steps we went through:

- The original dataset contains some duplicates, probably because of retweets or spams. We drop those, so that they don't affect the weights attributed to words by our vectorizer later on.
- A large number of tweets contained emoticons, we identify their patterns using Regular Expression library and translate them to words that are meaningful to our analysis.

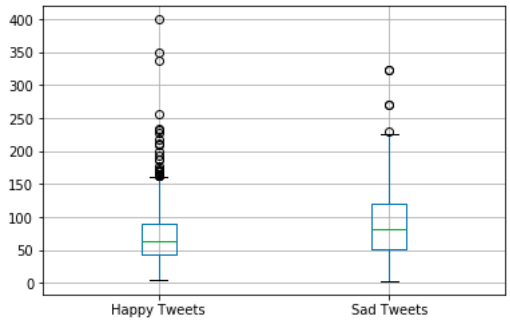

Fig. 1. Tweets length (in number of characters)

- We replace exclamation and interrogation marks as well as ellipsis by explicit words.
- We split negations of the form verb-n't into two separate words: verb not.
- We normalize some contractions, some widely used slang words and abbreviations by their corresponding words.
- We keep <user> and <url> tags, as well as the hashtags and transform the symbol '#' into the word 'hashtag'.
- We remove what is left of the punctuation and numbers and lemmatize the tweets for further normalization of the text.

B. Tweets Vectorization

Machine learning models require text data to be converted into vectors of real numbers. We try different word embedding methods to vectorize the tweets that we explain below:

1) Count Vectorizer counts occurrences of tokens and builds a sparse matrix of tokens by tweets. We start our baseline model using Count Vectorizer but we later on switch to TF-IDF as it yields better results.
2) TF-IDF assigns a weight to each token as a statistical measure of how important the word is to the document taken in the context of the entire corpus.
   Parameters such as n-grams and stopwords can be selected for the vectorizers: we evaluate their effect on the accuracy on our validation set.
   - Stop Words
     As a first approach we try using generic English stopwords as defined by NTLK [1], removing those actually reduced our accuracy. Looking more closely, words like 'but', 'not'... actually make sense to our analysis. Hence we decided to define our own set of stopwords extracted from the most common words in our corpus. This results in a better accuracy than the one we obtain from removing English stopwords. However, the best accuracy was the

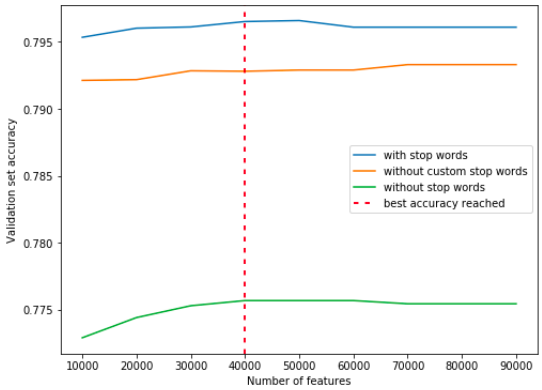

Fig. 2. Accuracy on validation set using different stopword methods

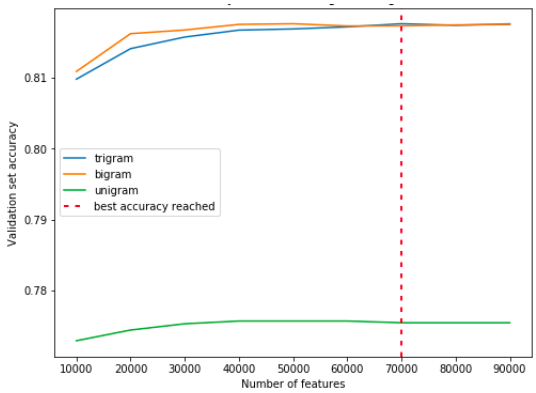

Fig. 3. Accuracy on validation set using different n-grams

one resulting from not removing stopwords at all. Figure 2 summarizes the accuracy on the validation set of a logistic model using the three different approaches: removing English stopwords, removing custom stopwords, and keeping all stopwords.

- N-Grams
  We use a similar approach as for stopwords where we tried different n-gram ranges and measured accuracy on the training set. We see that using bi-grams and tri-grams improves accuracy considerably which makes sense, because textual data is always contextual and has intrinsic dependencies, whereas models like logistic regression and Naive Bayes assume independence of features. The results of our validation set accuracy are displayed on 3 where we see that bi-grams and tri-grams give us almost similar results and uni-grams performed a lot worse. Notice that we do not go beyond tri-grams as our models would become too computationally costly.

3) Word2Vec
We tried using a Glove pre-trained dictionary from Glove [2] but the obtained results were not satisfying. Thus, we decided to create our own Word2Vec vectors from the pre-processed dataset to then construct the embedding matrix needed to train our neural networks. We choose 200-dimension vectors to avoid over-simplifying

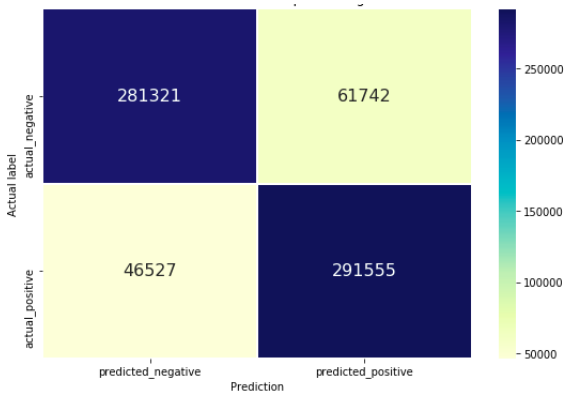

Fig. 4. Confusion matrix of logistic regression on validation set

the models. Only words that appeared 5 times throughout the dataset are used to create the vectors.

III. Basic Machine Learning Models

For our model selection, we try out five different models consisting of Naive Bayes, SGDClassifier, SVM, and Logistic Regression. We train them on the cleaned dataset for which we apply the tf-idf vectorization with the selected parameters as discussed above. To test our models, we do a random $70-30$ train-validation split.

Table I summarizes the accuracies on the validation set.

Models	Accuracy	AUC
Naive Bayes + Count-Vectorize	0.782	0.863
Naive Bayes + TF-IDF	0.800	0.887
SGDClassifier + TF-IDF	0.824	0.904
SVC + TF-IDF	0.840	0.919
Logistic Regression + TF-IDF	**0.841**	**0.919**

TABLE I
Methods used and Accuracy on validation set

We see from our summarized results in table I that logistic regression outperforms the other models where we get an accuracy equal to **0.849** on the test set after submission on AIcrowd.

Figure 4 illustrates the confusion matrix of logistic regression when testing on the validation set for a threshold equal to 0.5. This is actually the furthest we could push our "basic" machine learning models. For the next section, our main focus will be neural networks.

IV. Neural Networks

After applying the different machine learning methods explained above, we decide to implement neural networks using Keras with TensorFlow backend. We quickly realized that we needed a lot of computing power to train each of our neural nets thus we ran most of our models on Colab [3] to speed up the training. It was still moderately computationally costly, and as a result, we were not able to tweak as many parameters as we would have wanted. To measure the accuracy of each model, we consider 5% of the data as a validation set and use the remaining 95% for training.

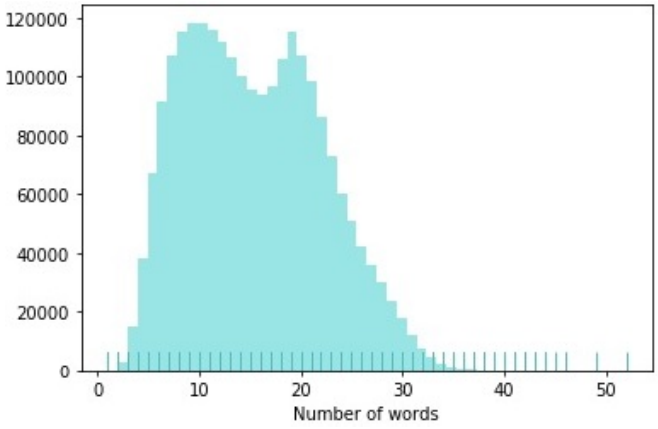

Fig. 5. Distribution of tweet length by word

A. Tweets embeddings and implementation choices

All of the neural networks models use an embedding layer based on our own pre-trained word2vec embedding. The word2vec embedding ignores all words that occur less than 5 times: the value 5 seems to yield the best validation accuracy.

Each tweet fed in the neural net is tokenized, and we use a vocabulary consisting of the top $120'000$ words that we obtain from the tokenization procedure. We then transform word tokens to sequences of indexes that correspond to their ranks, with 1 being the index of the most common word. Each sequence is padded, in order to ensure all inputs to the training model have the same length (which is the max_len parameter corresponding to the input of the Embedding layer). The optimal length was determined to be 140 for some models, and 40 for others. The padding only keeps the first max_len words or adds 0s to the sequences shorter that 140.

Since the Adam optimizer [4] usually shows considerable performance gains in terms of computational speed, all our neural network models are implemented using it. Unlike Stochastic Gradient Descent [5] which maintains a unique learning rate, Adam finds distinct learning rates for each parameter and updates them through the training process.

Considering the binary nature of this classification problem, we choose to combine binary cross-entropy loss to a final Sigmoid [6] (defined as $f(x) = \frac{1}{1+e^{-x}}$) activation function. Thus, the final computed values are mapped to a probability. The activation function used *inside* the neural networks (between the hidden layers) was the ReLU (Rectified Linear Unit) function [7] which is defined as $f(x) = max(0, x)$.

As the final activation function in our models is a Sigmoid function, it computes the probability that a tweet is positive. Hence the output of our neural network is a number that is fitted in the range $[0, 1]$. To choose the best threshold, we try values ranging from 0.3 to 0.7 and plot the corresponding accuracy on the training set as can be seen in figure 6. We hence choose for each of our neural networks the threshold that maximizes that accuracy (the default value being 0.5 which does not necessarily yield the highest accuracy). In case of RNN GRU the threshold giving the highest accuracy is 0.44 as can be seen on the plot.

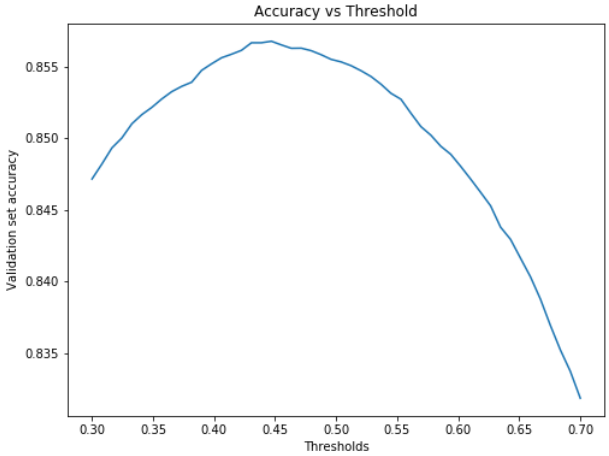

Fig. 6. Accuracy of RNN GRU by selected threshold

Table II summarizes the results of our different neural network models on our validation set. For each of them, 6 epochs are completed with a batch size of 1024.

Methods	Accuracy
Simple NN	0.765
RNN LSTM	0.851
RNN GRU	0.859
RNN Bi-LSTM	0.853
CNN with max_len 140	0.864
CNN with max_len 40	0.864
Ensemble	**0.880**

TABLE II
METHODS USED AND ACCURACY ON VALIDATION SETS

In the following subsections we discuss more in details the architecture of each one of our neural network models.

B. Simple Neural Network

This first simple model gives us insight about how a simple neural network would perform on this problem and has the following architecture:

Input layer $\rightarrow$ Embedding layer $\rightarrow$ Flatten layer $\rightarrow$ Output layer

C. Recurrent Neural Network with Gated Recurrent Unit

Since we are dealing with text, the context of appearance of a word is relevant, which means that the words before the current word are important and should not be left out. A recurrent neural network serves that purpose: it has loops that retain previous words to inform later ones. The main advantage of RNN with gated recurrent unit is that it is faster to train than RNN with long-short term memory (described in the next subsection), since it has a smaller number of parameters.

The architecture of our model is the following :

Input layer $\rightarrow$ Embedding layer $\rightarrow$ GRU layer $\rightarrow$ Dropout layer $\rightarrow$ Flatten layer $\rightarrow$ Dense layer $\rightarrow$ Dropout layer $\rightarrow$ Dense layer $\rightarrow$ Output layer

The two dropout probabilities we found to be the best are 0.3 for the first dropout layer and 0.5 for the second one.

D. Long Short-Term Memory

- Recurrent Neural Network with Long Short-Term Memory [8]
  This is a recurrent neural network as the one described above with an additional LSTM layer. The architecture of our model is the following :
  Input layer $\rightarrow$ Embedding layer $\rightarrow$ Masking layer $\rightarrow$ LSTM layer $\rightarrow$ Dense layer $\rightarrow$ Dropout layer $\rightarrow$ Output layer
  This model's accuracy was around 50% at first. Adding a Masking layer solves this issue since it implies disregarding the padding part of the data [9]. This was a critical step and helped us reach the result displayed in table II.
- Bidirectional Long Short Term Memory
  The architecture of this model is the same as the one described above; in addition to that, we wrap the LSTM layer into a Bidirectional layer [10], which is based on the idea that additional context improves the training.
  This doubles the training and uses two LSTMs on the input: the second uses a reversed version of the input text.

E. Convolutional Neural Network

Since we are dealing with a natural language processing task, we also implement a Convolutional Neural Network. We attempt a different number of convolutional layers, and find that 4 convolution layers yield the best validation accuracy. The architecture of our model is as follow :

Input layer $\rightarrow$ Embedding layer $\rightarrow$ Dropout layer $\rightarrow$ Conv1D layer $\rightarrow$ Conv1D layer$\rightarrow$ Conv1D layer $\rightarrow$ Conv1D layer $\rightarrow$ Flatten layer $\rightarrow$ Dense layer $\rightarrow$ Dropout layer $\rightarrow$ Activation layer $\rightarrow$ Dense layer $\rightarrow$ Activation layer $\rightarrow$ Output layer.

Among the values $\{3, 5\}$ for the kernel size, the value 3 yields the best result. The first convolution layer has 600 filters and the number of filters decreases for each of the following convolution layers $(300, 150, 75)$. The first dropout layer drops a node with probability 0.25 while the second one with probability 0.4.

F. Ensemble methods

In order to improve the accuracy of our predictions, we use Ensemble learning which consists of combining several independent classifiers by taking the majority vote of their predictions. Through trial and error, the best accuracy on the validation set is achieved by combining the following 3 models:

1) Bidirectional Long Short Term Memory with max_len = 140 and min_count= 5
2) Convolutional Neural Network where the first Dropout Layer has a value of 0.25 with max_len = 140 and min_count= 5
3) Convolutional Neural Network where the first Dropout Layer has a value of 0.4 with max_len = 40 and min_count= 5

where max_len refers to the maximum length of a padded vector resulting from the tweets embedding discussed earlier in our report and min_count is the minimal number of times a word has to appear to be taken into account in our embedding.

V. Conclusion

Our approach to the problem of classifying tweets by the sentiment of their user made us realize that this type of classification problem is hard to make 100% accurate: the tweets dataset contain not only valid words, but also "texting" language, numbers, URLs, and other symbols. It requires a lot of effort to normalize the text perfectly because of the misspelling mistakes, use of slang and regional variations of some words... Furthermore tweets can be classified as expressing a happy feeling or a sad feeling using the presence of some smiley, but the smileys are not always consistent with the text, because of the use of sarcasm for example. Hence in order to improve the accuracy of our models and as an area of improvement, we could try to detect and take into account sarcasm [11].

On the other hand, the fact that our predictions reach a level of 88% accuracy, means that machine learning can be quiet efficient at the task of classifying text. The best performing models we found are neural methods. Although Convolutional Neural Networks are known for their use in Computer Vision, they are actually able to give very interesting results in Natural Language Processing[12]. Long Short Term Memory based models, which are an important Recurrent Neural Networks variation, are also part of the best Ensemble: by securing long range dependencies, they make classifying tweets easier, since the context of a word matters.

We also realized that there is also no secret recipe to find the neural network which works the best for the task at hand, it is mostly through a trial and error approach that the accuracy can be improved.

References

[1] [Online]. Available: https://www.nltk.org/
[2] [Online]. Available: https://nlp.stanford.edu/projects/glove/
[3] [Online]. Available: https://colab.research.google.com/
[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[5] J. Kiefer, J. Wolfowitz *et al.*, "Stochastic estimation of the maximum of a regression function," *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, 1952.
[6] [Online]. Available: https://en.wikipedia.org/wiki/Sigmoid_function
[7] 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.
[8] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," *arXiv preprint arXiv:1503.00075*, 2015.
[9] "Masking and padding with keras." [Online]. Available: https://www.tensorflow.org/guide/keras/masking_and_padding
[10] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural networks*, vol. 18, no. 5-6, pp. 602–610, 2005.
[11] R. González-Ibánez, S. Muresan, and N. Wacholder, "Identifying sarcasm in twitter: a closer look," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers-Volume 2*. Association for Computational Linguistics, 2011, pp. 581–586.

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