Text Sentiment Classification

Machine Learning (CS-433), Department of Computer and Communication Sciences,
École Polytechnique Fédérale de Lausanne

Amir Ghali
amir.ghali@epfl.ch

Mahmoud Sellami
mahmoud.sellami@epfl.ch

Victoria Adcock
victoria.adcock@epfl.ch

***Abstract*—This report details our approach to classifying the sentiment of a given piece of text. By targeting our focus on the social networking giant, Twitter, our model aims to predict whether a tweet had previously contained a positive or negative smiley. Numerous text preprocessing techniques are considered and we discuss our decision to represent the data as a vector. By combining machine learning methods, particularly supervised classification algorithms, we are able to generate our predictions of a text's sentiment to an accuracy of 85.8%.**

I. Introduction

Sentiment Analysis is a field within Natural Language Processing (NLP), which aims to extract a sentiment from a subjective source, such as a piece of text [1]. The recognised sentiments are subsequently classified as being either positive or negative. In exceptional cases, a more fine-tuned sentiment classification scheme can be implemented, where a wider range of sentiments are outputted.

The age of the internet has revolutionised the way in which people express their opinions. With the introduction of blog posts, online discussion forums, product review sites etc, it has become easier and faster for people to express their content or disdain about a given topic. The applications of sentiment analysis are extensive and powerful. The ability to extract insights from online data is a practice that is being universally adopted by organisations across the globe. Sentiment analysis systems allow companies to gain key insights on the users' thoughts, thus, improving their products and services. Equally, shifts in sentiment on social media have been shown to correlate with shifts in the stock market [2].

Social networks are amongst the most frequented sites on the internet; with more than 330 million monthly active users, Twitter is in the top 40 most visited websites worldwide [3][4]. In our project, we aim to analyse tweets sourced from Twitter, in order to determine whether they had previously contained a positive or negative smiley. A variety of data preprocessing techniques are used, and a study into the different text representations is considered before implementing our model. Finally, we analyse various classification algorithms, in order to garner the best results.

II. Data

A. Data Preprocessing

In order to train our model, 2.5 million pieces of labelled data (which used to contain either a positive or negative smiley) were sourced from Twitter. Our model is then tested on a data set containing 10,000 tweets. Upon inspecting the given data, it became evident that data preprocessing would be a crucial first step, in order to transform the text into a form that is predictable and analysable. Since the data originates from an informal environment, the majority of the text samples contained hashtags, special characters, URLs and informal language, such as the use of colloquialisms. Several data preprocessing techniques were considered when creating our model;

Hashtag removal: The use of hashtags # in messages posted online was popularised following the launch of Twitter. Consequently, the hashtag is a prominent feature present in the data set, and is usually followed by a word or combination of words. Therefore, we remove the hashtags and leave the remaining word or concatenation of words, which are split into separate words using the library, wordninja.

Removal of URLs, usernames and mentions: Since URLs, usernames and mentions (represented in the data as <url>, <user> and '@' followed by a user respectively) are irrelevant when analysing a text's sentiment, it was deemed useful to remove them.

Removal of small words and stop words: We classify a word as having a length greater than one. Hence, we remove any single letters, as they cannot demonstrate a sentiment alone. Stop words (e.g. "the", "at", "which" etc) are equally removed for the same reason.

Correcting spelling mistakes: It is important to recognise that spelling mistakes occur often, so common spelling mistakes should be corrected, in order to allow for a more efficient text sentiment classification model.

Treating colloquialisms: When writing informal text, it is common to use colloquialisms, such as slang words and abbreviations. Typical examples include "lol" to mean "laugh out loud" and "l8r" for "later". We decided to use a dictionary to transform these words to their original meaning.

Punctuation deletion: Although considered to be a controversial decision, we deemed it appropriate to delete punctuation marks in our data set. Occasionally, punctuation can be used to convey a certain sentiment, however, we decided that it was not particularly beneficial to include it when analysing the given data.

Apostrophe treatment: We return word contractions (e.g. "it's" instead of "it is", "i've" instead of "I have" etc) to their formal English terms.

Translating emojis: Emojis are infiltrating online texts and when processed correctly, can help with a better sentiment analysis. By evaluating their intended purpose, we can garner greater insight into the sentiment shared by the user. We categorise emojis into three distinct groups: smiley faces, sad faces, and neutral faces. When an emoji is detected, it is translated into a word according to its group.
Emphasising certain words' sentiment: Certain words convey a distinct sentiment, so by using a set of positive and negative words, a "positive" label is added whenever a word in the text data features in the positive set of words, and a "negative" label if it appears in the negative set. Also, if there is a presence of the word 'haha', it was automatically classified as positive.
Deleting numbers: We do not consider numbers to act as strong sentiment polarisers as they do not convey any particular emotion. Therefore, they were deleted from the data set.
Lemmatisation and Stemming: Since conjugating a verb does not alter its sentiment, words were reduced to their stem, as it does not negatively affect our text sentiment analysis. Therefore, words will be given the same weight regardless of the different tenses.

B. Vector Representation of Tweets

A critical component of many NLP systems is word representation. It is common practice to represent words as indices in a vocabulary list, however, this is unsuccessful in capturing the rich relational structure of the lexicon [5]. Vector-based models perform better in this regard.

Several techniques were applied to transform the tweets into numerous numeric-vector representations which depict significant characteristics of the text.
N-gram: The N-gram model is a concept frequently used in NLP systems. It is the simplest model that assigns probabilities to sentences and sequences of words [6]. It is essentially a sequence of N words. For instance, an example of a 2-gram is the sentence "turn right". This model accounts for sequences of words in contrast to just using singular words. For high N values, we require more resources and greater computational power to train our model. For the implementation of N-gram, we used the convolutional neural network. It helps to better detect negative tweets when using a filter of length 2.
Term Frequency - Inverse Document Frequency (TF-IDF) Representation: The TF-IDF was our baseline technique since it captures a word's importance by computing its weight according to the frequency at which it appears in a single tweet, and then every tweet.
Word embedding: We will primarily focus on pre-trained Global Vector (GloVe) embedding. GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Similar words are mapped to high dimensional vectors that are close to each other in space. For our project, it was possible to use pre-trained word vectors, which were trained using 2 billion tweets and a vocabulary consisting of 1.2 million words [7]. We use the provided GloVe_solution file to create our own word vector embedding from co-occurrence matrices.
Word embedding to tweet embedding: In order to achieve this, we implemented 3 different methods. The first one method being 'TF-IDF', which uses the $TfidfVectorizer$ to compute the embedding of the tweets. The second one is 'average', which takes the mean word embedding of the words present in the tweet. The third method is 'tfidf-average' which takes the mean of the words embedding of the words present in a tweet and multiplies them with their respective TF-IDF values.
.

III. Models

A. Linear Models

Combining different linear classifier with our own created word embeddings was ineffective in providing good results. In fact, using the logistic regression and the method 'TF-IDF' method resulted in an accuracy of 0.54 on AiCrowd. Hence, this method was considered our base-line model. Using the 'average' method enabled us to garner the best results for this combination (linear classifier + own embeddings), reaching a value of 0.727. We used a pipeline from sklearn for a different combination of linear classifiers (logisticRegression, LinearSVC, SVM) but the results did not improve.

B. Bidirectional Encoder Representations from Transformers

Bidirectional Encoder Representations from Transformers (BERT) is a technique for NLP pre-training developed by Google. Its primary function is to apply the bidirectional training of Transformer to language modelling. This contradicts previous methods which focus on looking at a text sequence either from left to right or combined left-to-right and right-to-left training. The transformer is an attention mechanism that learns contextual relations between words in a text. It includes an encoder that reads the text input and a decoder that produces a prediction for the task [8].

C. Pre-trained GloVe Representation

Using the pre-trained GloVe embeddings, we achieved an average increase in accuracy of 15%.

Neural Networks (NN)
A neural network models itself after the human brain, which by means of an algorithm, allows the computer to recognise patterns and learn by incorporating new data. It is a collection of connected nodes (artificial neurons). Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. These neurons are aggregated into layers, where the different layers execute different transformations on their inputs. With the first layer being the input layer and the last being the output layer, a signal may also traverse multiple hidden layers in between. For our model, these layers include:

1) *The Embedding Layer:* We initialise this layer with a minimum of 3 parameters; the maximum size of the vocabulary (vocab_size) is (input dim = 1 + nb_word), the size of the vector space in which words will be

embedded (output dim), and the length of the input sequence (input length). By feeding this layer with these input parameters, it will learn and devise an embedding for each word in the training data set. Alternatively, we can provide our own embedding_matrix which is computed from the pre-trained word embeddings. In doing so, the results improve most of the time.

2) *The Dense Layer:* Each neuron receives an input from every neuron in the previous layer, thus, is densely connected. This layer has an associated weight matrix, stating the importance of the input signal, a bias vector, and the activation functions(non-linear) of the previous layer. The activation function of an artificial neuron defines the output of that neuron, given its input. It maps the resulting values onto the desired range (between -1 and 1, in the context of our project). But since relu actiation function achieved better results, we fed our neural network with y values ranging from 0 for positive tweets and 1 for negative tweets.
3) *The Dropout Layer:* In this layer, individual nodes are 'dropped out' of the net at random, with probability $1-p$, so that a reduced network remains. This is necessary, as it prevents the problem of over-fitting. Neurons start to become codependent on each other during training, since a fully connected layer occupies the majority of the parameters. As a result, this limits the individual power of each neuron, leading to over-fitting of the training data.

Convolutional Neural Networks (CNN)

A CNN is a class of deep neural networks, typically applied to analysing visual imagery. They are simply neural networks that use the linear mathematical operation, convolution, in place of ordinary matrix multiplication in at least one of their layers. They are a good extension in addition to the previously described layers. Although not essential, since standard algorithms will still perform the desired task, they are more efficient because they reduce the number of parameters. They use filters of size (filter_size * MAX_LENGTH), which is equivalent to using N-grams, as stated previously. Therefore, we deemed it appropriate to trial a CNN on the pre-processed data set.

We have a matrix representation with dimension size '*nb_word* $\times$ D' for each tweet, where D = 200 is the dimension of the GloVe word embeddings, according to the GloVe representation. We provide a summary of the additional layers for the CNN models:

1) *The Convolutional Layer:* This is the core of CNN systems; it is used for feature extraction. It interprets the number of filters, the size of the filter and the activation function as parameters. Several convolutions are executed on the input, where each convolution uses a unique filter. Consequently, this ensures different feature maps, which are then aggregated to output the final result. We used the Convolution1D layer in our model.
2) *The Pooling Layer:* Pooling layers reduced the dimensions of the data, by merging the outputs of neuron clusters at one layer into a single neuron in the following layer. This is to reduce the number of parameters and computations in the network, and thus, to overcome the problem of over-fitting. It also reduces the run time significantly. Our model uses the MaxPooling1D layer.
3) *The Flatten Layer:* Flattening transforms a multidimensional feature matrix into a feature vector (after the signal has traversed the convolutional layer) that can be passed through to a fully connected neural network classifier for the final classification.

Recurrent Neural Networks (RNN)

RNN systems are an extension of a conventional feedforward neural network, discussed previously. The connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behaviour. Contrary to feedforward neural networks, RNNs are able to use their internal state to process a variable-length sequence input. They handle the variable-length sequence by possessing a recurrent hidden state, whose activation at each stage is reliant on the previous stage. By using their hidden layers, they can retain information for a long period.

Long Short-Term Networks (LSTM) have feedback connections, unlike standard feedforward neural networks. They can equally process entire sequences of data, and not just single data points. A typical LSTM unit is comprised of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. They also aid in preserving the error which can be backpropagated through time and space. Our model uses LSTM systems' varying hidden layers, memory cells of size (30, 100) and dropout rates of (0.2, 0.3)

Gated Recurrent Units (GRU) are similar to LSTM systems, as they possess a 'forget gate', but they use fewer parameters, since they lack an output gate. Likewise, GRUs have been shown to exhibit even better performance on certain smaller data sets [9]. For our project, we tested a GRU system and were found to solve the problem of the vanishing gradient problem of an ordinary RNN, by implementing the update and reset gate techniques.

Bidirectional Networks

Implementing the Bidirectional method on LSTM or GRU, this modification on the normal RNN network made it possible to analyse the words in a sentence, not only the trivial (from left to right) direction, but also from right to left, as we sometimes need to see what we have before and after a word to better understand the sentiment of the whole sentence. E.g: "Teddy bears are on sale" and "Teddy Roosevelt is the president".

IV. Results

A. *Pre-processing*

Using our implemented preprocessing methods, we attempted a several combinations in order to achieve the best one. Our best preprocessing model was to process: hashtags, punctuation, number removal, letter repetition removal, small words removal, slang word translation, correcting spelling mistakes, apostrophe processing, punctuation removal and sentiment emphasis. The order in which to perform these processing methods is important, as the slang word dictionary and spell check dictionary enables emphasize sentiment to detect more words. Initially, we also included the processing of emojis and stop words. However, when we removed these preprocesses, we managed to increase our accuracy by approximately 3.5%.

B. Hyper-parameter choice

Each of the neural networks we used had several hyper-parameters. By cross-validating, we managed to find the best values that suit our task. See table I.

Hyper-Parameter	**Value range**	**Best value**
Word embedding dimension	25, 50, 100, 200	200
nb_word per tweet	10, 30, 50, 70	30
CNN number of filters	32, 64, 128	32
CNN Filter length	2, 3, 4	3
pool$_length$	2, 3, 4	4
Dropout rates	0.2, 0.3, 0.5	2
Batch size	32, 64, 128, 256	128
Activation function	relu, sigmoid, tanh	sigmoid, relu
Optimizer	adam	adam

TABLE I
HYPER-PARAMETER VALUES

Our model results using best preprocessing approach are represented by table II.

Model	**Training accuracy**	**Validation accuracy**
GRU + CNN + pretrained glove	84.2%	83.7%
NN without glove	77.02%	76.46%
GRU + +dropout + pretrained glove	85.17%	84.7%
LSTM + pretrained	84.18%	83.7%
GRU + pretrained glove	84.23%	83.7%
CNN + bidir GRU(30) + pretrained glove	86.72%	85.31%
CNN + bidir GRU(100) + pretrained glove	86.81%	85.37%
CNN + bidir LSTM(100) + pretrained glove	86.35%	85.46%
bidir LSTM(30) + pretrained glove	86.44%	85.72%
bidir LSTM(100) + pretrained glove	87.11%	86.17%
Bert	85.17%%	85.8%

TABLE II
MODEL RESULTS

Our best models were BERT and (CNN + bidir GRU(100) + pretrained glove) where both of them gave an equal accuracy of 85.8%

V. CONCLUSION

In this report, we explained different preprocessing techniques, word vectorization, text representation and different models options to fulfill the task of classifying tweet sentiments. However, sentiment analysis is not 100% effective, and is facing challenges that machine learning algorithms still cannot overcome, which may not necessarily be a negative result. In fact, natural language when spoken by humans can be misunderstood by others, even though they are human and speak natural language. Another example is the use of sarcasm and irony in texts that sometimes even human may not be able to register. Equally, having a positive emoji in a tweet does not necessarily imply a positive message, especially if the tone is sarcastic. So one can suppose that some of the training tweets were already misclassified. To conclude, sentiment analysis is a difficult task and requires a good understanding of the context of the message to correctly classify it. However, when the proper tools are used, one can hope for a satisfying result above 85% accuracy.

REFERENCES

[1] Techopedia.com. (n.d.). *What is Sentiment Analysis? - Definition from Techopedia.* [online] Available at: https://www.techopedia.com/definition/29695/sentiment-analysis

[2] Pagolu, V., Challa, K., Panda, G. and Majhi, B. (2016). *Sentiment Analysis of Twitter Data for Predicting Stock Market Movements.*

[3] Statista. (2019). *Number of monthly active Twitter users worldwide from 1st quarter 2010 to 1st quarter 2019 (in millions).* [online] Available at: https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/ [Accessed 10 Dec. 2019].

[4] Alexa Internet, Inc. (2019). *Alexa - Top sites.* [online] Available at: https://www.alexa.com/topsites [Accessed 10 Dec. 2019].

[5] Gautam, G. and Yadav, D. (n.d.). *Sentiment Analysis of Twitter Data Using Machine Learning Approaches and Semantic Analysis.*

[6] Kumar, P. (2017). *An Introduction to N-grams: What Are They and Why Do We Need Them?.* [online] blog.xrds.acm.org. Available at: https://blog.xrds.acm.org/2017/10/introduction-n-grams-need/

[7] Pennington, J., Socher, R. and Manning, C. (2014). *GloVe: Global Vectors for Word Representation.* [online] nlp.stanford.edu. Available at:https://nlp.stanford.edu/projects/glove/

[8] Horev, R. (2018). *BERT Explained: State of the art language model for NLP.* [online] blog.xrds.acm.org. Available at: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

[9] Chung, Junyoung; Gulcehre, Caglar; Cho, KyungHyun; Bengio, Yoshua (2014). *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.*