Machine Learning - Project 2
Text Classification

François Michel, Adrien Laydu, Manon Michel

***Abstract*—The aim of this text classification project is to perform sentiment analysis on tweets. Specifically, we would like to perform binary classification in order to predict whether a tweet originally contained a positive :) or negative :(smiley by analysing the remaining tweet contents. To achieve this we implement various data pre-processing and implement several text classification models.**

I. Introduction

Sentiment analysis is a specific field of Natural Language Processing which aims to depict the general emotions of a given text by analysing the latter. Sentiment analysis is often performed on textual data to detect sentiment in emails, survey responses, social media data, and beyond.

In this project the goal was to perform sentiment analysis on a specific type of textual data: tweets posted on Twitter. Such textual data is specific in that there is a character limit, there are hashtags and the text can contain slang and misspelled words. These peculiarities add an extra level of difficulty for the sentiment analysis.

The tweets from our datasets used to contain either a `:)` or `:(` smiley, which was removed. Our goal is to implement the best possible model to predict whether a tweet originally contained a positive or negative smiley. Note that this is vastly different to simply predicting the positive or negative undertone of a tweet. For example, positive smiley faces can sometimes be used for ironic purposes in an otherwise negative tweet.

In order to achieve our goal we preprocess our data, tokenize it and find the most accurate model. The classification accuracy is compared against our peers on the AICrowd platform.

II. The Data

The training data consists of 2.5 million tweets, half of them used to contain the `:)` smiley, we will call these positive tweets, and the other half previously contained the `:(` smiley, we will call these negative tweets. The test data is similar and consists of 10K numbered tweets. All of this data is somewhat preprocessed in that the text is lowercase and mentions of other users, such as @user, have been replaced by a `<user>` tag. Similarly, all urls have been replaced by a `<url>` tag.

The training data is labeled with a `1` if it originally contained a `:)` smiley and otherwise labeled with a `-1` if it originally contained a `:(`. Our training data is thus of the following form:

$$(\boldsymbol{x}_n, y_n)_{n=1}^{N}, \boldsymbol{x}_n \in \mathcal{A}_n^D, y_n \in \{-1, 1\}$$

Where $\mathcal{A}$ is the set of UTF-8 characters and D_n is the length of the n^{th} tweet and is at most 140, which is the character limit for tweets.

III. Data Tokenization

A. GloVe

GloVe [1] (for Global Vectors for word representation) is an unsupervised learning algorithm used to represent words as vectors. It uses words co-occurrences in a corpus (in our case, a corpus of tweets in which only words that appear at least 5 times have been kept) to construct a vector space in which distance relations can have a semantic meaning. The vector representation of a tweet is computed by taking the average of the vector representation of the words that compose the tweet.

B. BERT

The BERT tokenization is done in several steps as detailed in Figure 1. First, we split the sentence into words and apply some merging and splitting rules that were learnt during the pre-training phase of the model with the WordPiece algorithm [2]. For example, the word `eating` could be split into two: `eat` and `##ing`. Then, we add the string `[CLS]` at the begging and the string `[SEP]` at the end. We also add `[PAD]` multiple times in the end to reach the maximum size which we define in our model. In case the token is too long we simply truncate it. We transform the tokens into long indices (each long correspond to one word or subword). The position of the word/subword will be taken in account to compute the long index. And finally we compute an attention mask with the value `0` if the index is null (which corresponds to the padding `[PAD]`) and `1` otherwise. We end up with two arrays of long: the indices of the token and the attention mask.

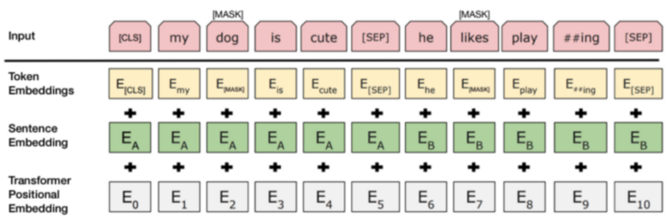

Figure 1. BERT Tokenization

IV. Pre-processing

In order to increase the potential accuracy of our model we implement various pre-processing methods. The particular format of our textual data being tweets, this gives us a wide scope of possible pre-processing methods such as dealing with other emojis, hashtags, tags and slang.

A. *Tag and RT removal*

Since our data consists of tweets, the latter contain tags such as `<user>` and `<url>`. On top of this, many tweets contain the acronym `rt` which is short for 'retweet'. These elements are specific to the twitter context and do not add anything to the sentiment of the tweet, notably to the possible presence of a :) or :(smiley. We therefore removed these three elements from the tweets.

B. *Contractions*

Tweets are often informal and therefore contain many contractions. We therefore transform all contractions into their full form, i.e. `don't` becomes `do not`. For this purpose we created an extensive dictionary of contractions. This dictionary also contains certain slang terms such as `cos` which means because or `u` which means you.

C. *Hashtags*

Hashtags are very present in social media, notably in tweets. They can contain valuable information for sentiment analysis. However, hashtags are often hard to read for models as they can consist of several words stuck together and to the # symbol. For this reason we transform hashtags into a more readable form, for example we aim to transform hashtags such as `#ilovetokyo` into `<hashtag> i love tokyo`. For this purpose we use a hashtag splitting code in addition to a library of most used words in order to separate words effectively.

D. *Emojis*

Another common textual feature used in social media is the use of emojis. These elements are of particular interest to us when performing sentiment analysis. We therefore create a list of common `happy`, `love` and `sad` emojis. We then replace every occurrence of such emojis with the corresponding emotion (happy, love or sad). For example `this is so cool :p` will become `this is so cool happy`.

E. *Sentiment Emphasis*

The goal of sentiment emphasis is to locate positive and negative words in the dataset and emphasise the latter. For example `i am happy` would become `i am happy positive` or `i am sad` would become `i am sad negative`. This would therefore give more weight to the sentiment-heavy words of the data and ideally help classify our tweets. We used a dictionary of positive and negative words which include slang and common spelling mistakes [3] for this.

With this technique we have to pay attention to negations, notably we chose to not apply this technique to tweets that contain the negation `not` as this technique then becomes counterproductive. On top of this, certain 'negative' words, such as swear words, can be used in a positive context to emphasise certain emotions, likewise 'positive' words can also be used in ambiguous contexts, especially on social media. We therefore created a frequency dictionary of 'negative' words appearing in our positive training set and inversely and decided to remove words which appear above a certain frequency (notably 0.0008% and 0.0004%). We found that this improved the accuracy gained from this pre-processing.

F. *Stopwords*

Stopwords are common, short function words, such as *the*, *is*, *at*, *which*, and *on*. Such words are considered as less important in the sense that they do not add information to the sentence and hence can usually be ignored when analysing textual data. Therefore, we use the nltk library [4], which provides stopwords for the English language, in order to reduce the noise through removing all of the stopwords in our tweets. For example `this is a sentence` will become `this sentence`. We also added a variation of this that removes all stopwords other than `not`, as the latter adds value to our sentiment analysis. As expected, this improved the accuracy gained from this pre-processing.

G. *Reduce punctuation*

The tweets contain a lot of repeated punctuation such as `!!!`. This repeated punctuation creates noise in the text that is unnecessary. We therefore reduce any repetition of `?`, `!` and `.` to a single occurrence of the latter. For example `this is great!!!` will become `this is great!`.

Preprocessing	Accuracy, %		
	N=8000, B=30	N=12000, B=30	N=20000, B=50
Nothing	85.09	86.92	86.55
Contractions	85.09	86.79	87.05
Tag removal	85.16	86.71	86.25
Hashtags	86.10	86.88	87.20
Stopwords	82.77	83.67	83.42
Sentiment	85.79	87.13	87.20
Emojis	85.28	86.88	86.47
RT	85.09	86.88	87.37
Punctuation	85.16	86.38	87.55

Table I
Influence of pre-processing on accuracy with BERT. N is the number of tweets and B is the batch size.

V. Models

Three simple models were trained : logistic regression, a 3-layer MLP, as well as a SVM. All these models were trained using various methods of scikit-learn [5] on a subset containing 60'000 randomly chosen tweets (with 30'000 positive tweets and 30'000 negative tweets), using GloVe tokenization and 4-fold cross-validation. Model parameters were fine-tuned to try and obtain the best accuracy. Then, a much more complex model was used, implementing BERT [6], and using the BERT tokenization.

A. *Logistic Regression*

Logistic regression is a classification model that uses the negative-log likelihood criterion to estimate the probability that a sample x belongs to a certain category. A grid search was performed in order to tune the regularization term. We use the elastic-net penalty function, which mixes $l1-$ and $l2-$losses with a certain ratio, which is also found by grid-search.

B. *SVM*

SVMs are a family of algorithms that try to maximise the margin between the data and the decision boundary, using a subset of the data (the support vectors). Although SVMs were designed to classify linearly, the kernel trick can be applied to classify non-linearly. In our case we used a linear kernel (already implemented in the LinearSVC method of scikit-learn), while performing a grid-search to determine the regularization term C, as well as the best penalty and loss functions to use.

C. *Simple MLP*

Multi-Layer-Perceptrons [7] are a class of feed-forward neural networks that generalize the linear perceptron [8]. They use back-propagation as a way to learn the weights matrices that fit best the data. A MLP is characterized by hyper-parameters including the number of hidden layers, the size of the hidden layers as well as of the input and output layers, the activation function used on each neuron, the regularization term α. In our case we fixed the number of hidden layers to 3. The size of the hidden layers as well as α were fine-tuned using a grid search. We tried ReLU, tanh and logistic activation functions - the ReLU function performing better in all cases.

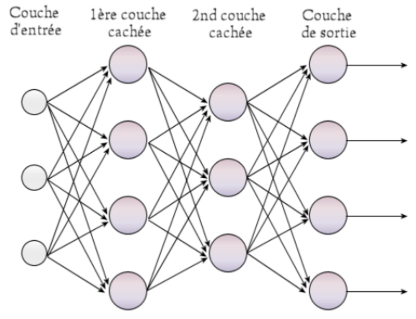

Figure 2. A schematic representation of a MLP

D. *BERT model*

Bidirectional Encoder Representations from Transformers (BERT), which is based on Transformers [9], was designed and published in 2018 by researchers from Google [6]. It is a deep learning model that has given state-of-the-art results on a lot of different tasks in natural language processing tasks like in text classification. The model has all its input elements connected to every output element and the weights between them are computed based on their connections. To create state-of-the-art models we have to use a pre-trained BERT model and finetune it with only one output layer. That's what we did for our project using the Transformers library by Huggingface [10].

BERT is pre-trained on a large set of English texts in a self-supervised way and the library has different models that are pre-trained on different data. We tried many of them and the best result was given by 'bert-uncased'.

First, we tokenized our tweets using the BERT tokenization. The tokenizer uses the pre-trained model and the WordPiece algorithm. This algorithm initializes the vocabulary to include every character present in the training data and then learn some merging and splitting rules. The maximum length of those tokens is 512 but we decided to use a smaller value to speed up the computation. Then, we fine-tune 'bert-uncased' model with the training data. This model is composed of 12 layers, 12 attention heads, and 110 million parameters. We finally used Adam [11] which is an adaptive learning rate optimization algorithm and linear scheduler with no warm-up steps to optimize our model as much as possible. We decided to use only one EPOCH because the computation time was already very long on our computers.

VI. Results

A. *Training accuracy*

The 3 simple models (Logistic Regression, SVM and Simple MLP) were trained on the GloVe tokenizations. Of all the pre-processing techniques discussed, the best combination uses only a subset : RT removal, hashtag transformation, emoji parsing, sentiment emphasis and punctuation reduction. Although these models perform better than chance (i.e their accuracy is greater than the expected 50% of a random classifier), it is clear that they are not precise enough for the aim of this project. The simple models were also trained on BERT tokenizations for completion purposes, but the results were systematically worse than when using GloVe.

Pre-processing was found to have a substantial effect on improving accuracy when training on GloVe embeddings, increasing the absolute training accuracy by up to 2.47% in the case of the MLP (see Table III.) . On the other hand, it appears that it had little to no effect when training on BERT embeddings, hence we chose not to display the corresponding accuracies in Table II. Our best guess as to why this is the case is that the BERT embedding system does

	LogReg	SVM	MLP	BERT-model
GloVe	63.53	63.60	66.22	N/A
GloVe + pre-processing	65.11	65.33	68.69	N/A
BERT-embeddings	61.78	64.00	64.31	87.52
BERT + pre-processing				87.77
Cross-validation	Yes	Yes	Yes	No

Table II
TRAINING ACCURACY (IN PERCENTAGE) OF THE CONSIDERED MODELS ON DIFFERENT TOKENIZATIONS

	LogReg	SVM	MLP
Absolute accuracy gain	1.58	1.73	2.47
Relative accuracy gain	2.49	2.72	3.73

Table III
ACCURACY DIFFERENCES (IN PERCENTAGE) ON THE SIMPLE MODELS WHEN USING PRE-PROCESSING

not have the same semantic distancing as GloVe, meaning converting the data into BERT tokens does not keep semantic relations between words. Since pre-processing is done at a semantic level (for example, translating emojis to emotions), it has no influence on the result of simple algorithms on BERT tokens. To prove this would require further work.

The main outcome of Table II is the massive difference in accuracy between the simple models (that are relatively similar in performance) and BERT (a difference of the order of 20% !). For this reason, we focused on BERT when considering different implementation options to choose our final model.

Note that the pre-processing used with BERT in Table II includes only the reduced punctuation and sentiment emphasis with frequency 0.0008% as these performed best. Notably, stopword removal works very poorly with BERT as this model looks very closely at prepositional words and where they appear in a search query. As shown in Table I, the influence of pre-processing varies depending on the number of tweets we train on.

B. Training and testing accuracy on BERT

Once we established BERT was the best option based on our testing on a small subset of the data, we chose to train it on a much larger subset of the dataset, composed of $N = 400'000$ tweets. We then generated the predictions corresponding to the given dataset to let the AICrowd verifier check the model's test accuracy. The resulting training accuracy is 88.9%, and the corresponding test accuracy is 88.2%. From those figures we can say that our model generalises well.

VII. Summary and discussion

To conclude, BERT is a very powerful model for natural language processing. Unfortunately, this meant that the usual pre-processing for sentiment analysis on tweets were not as effective as on classic models. Therefore in order to improve the model we could have focused our pre-processing on the compatibility with BERT. In addition to this, we could have obtained a better accuracy by running the model with a smaller batch size and for multiple epochs. For this we would also need to optimise the model as it takes four hours to run on 400,000 tweets with batch size 20.

Most of all, to improve our results we could try to implement XLNet [12], a generalized autoregressive pretraining for language understanding. Under comparable experiment settings, XLNet outperforms BERT on 20 tasks, often by a large margin, including question answering, natural language inference, sentiment analysis, and document ranking.

References

[1] 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

[2] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, and M. Norouzi, "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016. [Online]. Available: https://paperswithcode.com/paper/googles-neural-machine-translation-system

[3] M. Hu and B. Liu, "Mining and summarizing customer reviews," in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 168–177. [Online]. Available: https://doi.org/10.1145/1014052.1014073

[4] E. Loper and S. Bird, "Nltk: The natural language toolkit," in *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*, 2002.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018. [Online]. Available: https://arxiv.org/abs/1810.04805v2

[7] Wikipédia, "Perceptron multicouche — wikipédia, l'encyclopédie libre," 2020, [En ligne; Page disponible le 23-juin-2020]. [Online]. Available: http://fr.wikipedia.org/w/index.php?title=Perceptron_multicouche&oldid=172277711

[8] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958. [Online]. Available: http://dx.doi.org/10.1037/h0042519

[9] I. Polosukhin, L. Kaiser, A. N. Gomez, L. Jones, J. Uszkoreit, N. Parmar, N. Shazeer, and A. Vaswani, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[10] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6

[11] J. B. Diederik P. Kingma, "Adam: A method for stochastic optimization," 2014. [Online]. Available: https://arxiv.org/abs/1412.6980

[12] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," 2020.