ML Project 2: Text Sentiment Classification

Guerraoui Nada, Peeters Thomas, Petersen Molly
School of Computer and Communication Sciences, EPFL, Switzerland

***Abstract*—In this report, different machine learning and deep learning techniques are used to perform sentiment classification on 10k tweets. The goal of the classification task was to correctly determine whether a tweet contained smiling face ':)' or a frowning face ":(". To do this, some basic classifying techniques were employed (regression and a support vector classifier), as well as different artificial neural network architectures, such as LSTM model with GloVe embeddings, as well as different variations of huggingface BERT transformers fine-tuned on twitter data. The analysis of those models shows that neural network architectures out-perform linear classifiers. A publically available roBERTa model first finetuned by Cardiff University on 58 million tweets having the best performance, with 89.8 percent of tweets being classified correctly.**

I. Introduction

The field of natural language processing (NLP) has grown over the past few years, due to the increased availability of text data and improvement of machine learning methods to analyse this data [1]. The internet, and social media in particular, has provided a rich source of text data to make inferences in a wide range of domains, from analysing the spread of fake news, understanding sentiment regarding a product, to the infamous "Google Flu Trends". [1], [2]

Twitter is a social media platform that has been heavily used in a variety of NLP tasks, including sentiment analysis. With over 300 million monthly users, the platform's presence is ubiquitous across the world. Additionally, the character limit of 280 characters (previously 140 characters) requires the user to communicate in a concise manor. The concise and widespread nature of tweets has likely been instrumental to its value for sentiment analysis [3]

Tool for modern supervised sentiment analysis can be split into two main sections: linear and non linear . Linear methods include naive bayes classifiers, support vector machine, perceptrons, and logistic regressions. Non linear methods include kernel methods and neural networks. Neural networks have become a dominant tool in many machine learning tasks over the past decade, and NLP was not excluded from these advancements. There are many neural network architectures that have applications in NLP, such as feed forward neural networks, Long short term memory, convolutional neural networks and transformers [4]

II. Models and Methods

A. Data

The total available training data was made up of 2.5 million tweets, equal parts of class ":)" and ":(".

When looking through the tweets, there appeared to be some non-english tweets, although we are unsure of all the languages that were represented in the dataset, and what proportion was not english. Since the data was large it would have been impossible to inspect all of them. When skimming through the data, most non english words found seemed to be Southeast Asian Pacific languages such as Tagalog or Indonesian. Although, there were mispelled words in the tweets, we chose not to perform any spelling correction given the dataset appeared to be multilingual, and we did not know which languages were included.

B. Simple Classifier

At first, the tweets were preprocessed. This is an important step in machine learning, and this is done to ensure the performance of the model. In our case, for the text sentiment classification, we have different processing to clean the text such as:

- Remove urls, html elements, numbers and punctuations.
- Converting the text to lower case
- Reduce the length of words, for example: caaaar becomes caar
- And finally apply lemmatization [5]

Lemmatization will replace word to their root forms (for example cars will be replace by car). This will allow to reduce the number of unique words. Putting all the text in lower case will also have the same effect ('CAT' and 'cat' will be seen as the same words). Using those two methods will reduce the redundancy in our dataset and can therefore improve the output of the model. Removing some strings that has no impact on the analysis such as some of the punctuations, numbers, stop words ('the', 'a', 'and') will allow to reduce the size of the input and to improve the output of the model.

To create the tensor input, word embedding technique is used. This method allow to convert the words from the tweets to a vector of float numbers of a specific size. To build this embedding matrix, the files from the github of the project are used. First a list of the words appearing 5 times in the training set are made, then the co-occurrence matrix that gives the number of times different words have appeared together in a Context Window is build. Using this matrix and the vocabulary, we will be able to build an embedding vector for each word using matrix factorization. The input tensor will be the average of the embedding vectors corresponding to the words in the tweet.

After getting the input data for the models and the labels, we start building simple and basic models which are all known classification methods from **sklearn library**. The first one is **Logistic Regression**, then we use the **Linear Support Vector Classifier** and finally, we use the **Support Vector Classifier**.

We also used a **Grid Search** from **sklearn** and cross validation to find the best combination of hyperparameters that gives the best performance of these models.

Using this, we deduced that the best hyperparameters for these models were:

- For Logistic Regression: { 'C' : 10, 'penalty': 'l2 }
- For linearSVC: { 'C' : 1, 'loss': 'hinge', 'penalty': 'l2 }
- For SVC : { 'kernel' : 'rbf' }

And for the other hyperparameters, we kept the ones set by default.

We deduced that the performance of these three models was about the same. Mevertheless, since SVC use a gaussian as kernel we will obtainend a non linear model that has slightly better results than the other two (see I). However the computation time for training is much larger than for the two linear ones.

C. LSTM & Bidirectionnel LSTM

Long short term memory (LSTM) models are introduced by Hocheriter Schmindhuber in 1997 [6].They revolutionized many domains such as speech recognition in 2007 and are now widely used for text classification. Those models can grasp long and short term dependencies. The characteristics of those models make it more accurate for text sentiment analysis that the previous one presented in section B.

To train those models, we will first do the same pre-processing as for the first model. Nevertheless, we will build the embedding matrix using pre-train word vectors : 'glove.twitter.27b.200d.txt'. This file contains 1.2 embedding vectors (vectors of size 200) of words coming from 2B tweets. Those embedding vectors were trained using the glove method [7]. We will extract the vectors corresponding to words of the training dataset to build the embedding matrix.

To use this embedding matrix in the models we will use the Tokenizer() function from the **Keras.preprocessing** library in python to create a dictionary of unique words and assign an integer to each word. To build the input sequence we will replace words in each tweet by their unique integers and add a zero padding at the end to make sure that each input has the same size. The output of the models will be the sentiment analysis of the tweet : positive or negative.

The architecture of those models is composed of one embedding layers with weights equal to our embedding matrix, two LSTM layers, two dense layers with the activation function 'relu' and a last dense layer with the sigmoid activation function. We will use tensorflow and keras libraries to create the model.

We build a second model by adding bidirectional layer before the LSTM layer so that we read the tweets in the two ways from left to right and from right to left. This allow us to preserve information from both past and future and therefore improve the performance of this model compared to the forward LSTM.

We used grid search and cross validation to find the number of epoch and the learning rate of the model. We will use 'Adam' optimizer with cross-entropy loss, a learning rate equal to 0.001 and 7 epochs.

The main challenges of implementing the LSTM model were to have good accuracy for the training and testing set and to reduce overfitting. For this task, we will use some regularization techniques such as dropout and batchnormalization. Dropout and batchnormalization layers will be add in between the initial layers of this model. We can observe on the figures below that those two techniques will reduce the gap between the training and testing accuracy and also training and testing loss. Moreover, the best accuracy is observed for the testing data using this model with 7 epochs and with regularization.

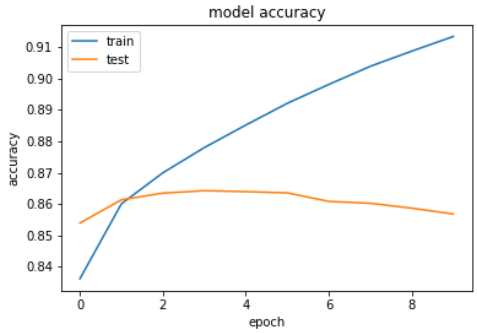

Fig. 1. Accuracy of the Bidirectional LSTM model without regularization

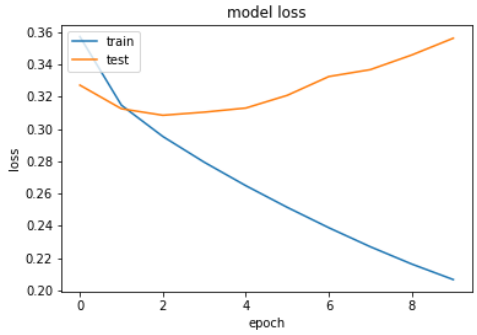

Fig. 2. Loss of the Bidirectional LSTM model without regularization

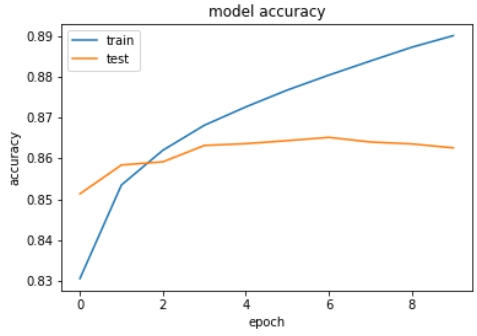

Fig. 3. Accuracy of the Bidirectional LSTM model with regularization

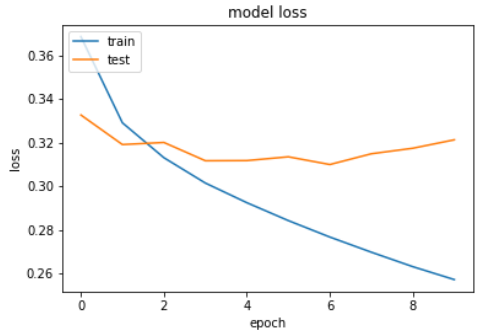

Fig. 4. Loss of the Bidirectional LSTM model with regularization

D. Transformers - BERT models

In addition to the models above, we finetuned several BERT models available through Huggingface (https://huggingface.co/). BERT stands for Bidirectional Encoder Representations from Transformers and was developed by researchers at Google based on the seminal "Attention is all you Need" paper by Vaswani et al which introduced the transformer model [8], [9].

The original transformer is an encoder-decoder neural network architecture, where the encoder involves one or more layers made up of two sublayers: a multi-head self-attention layer, and a feed forward layer. Self attention allows neural network to relate the importance of words that are potentially distant from each other in the input. This was an improvement over the previous state of the art, which made it difficult to relate words as they grew further in distance in an input. Additionally, transfomers involve positional embeddings of words in addition to the word embeddings themselves [9].

BERT is an variation of transformer models, which involves only the encoder. BERT base is made up of 12 layers, each with an attention and feed forward sublayer. [8] We used four different pretrained models available on huggingface. Two are models directly provided by huggungface (bert-base-multilingual-uncased, roberta-base), while two were roBERTa models uploaded by Cardiff University (https://huggingface.co/cardiffnlp) "cardiffnlp/twitter-roberta-base-sentiment", which they trained on 58 million tweets and "cardiffnlp/twitter-xlm-roberta-base-sentiment", which they trained on 198 tweets in 8 languages [10]–[12].Important to note, cardiff models were originally trained to do three class analysis (0-negative, 1-neutral, and 2-positive). For this reason, we labeled all positive data as two, and not one. Despite the models originally having three classes, none of the test data was given as a "neutral" label after fine-tuning.

We fined tuned all four of these models using all 2.5 million tweets. Although we used the tokenizers that belonged to the individual models, we also tested different text preprocessing models, including removing all punctuation except "!", removing numbers, and removing "<user>". Punctuation turned out to be important, so our final models removed numbers and " <user>" tokens. For all fine-tuning, we used a drop out probability of 0.1 for both attention heads and hidden layer dropout. We used an Adam with weight decay optimizer with a learning rate of 1e-3. Batch sizes of 8 were distributed across 8 google TPU's for total batch sizes of 64. All inputs were padded to a maximum character length of 512. Initially we started to do a grid search for hyperparameters, using a small subsample (200K) of the full training set. The goal was to test a variety of different model/hyperparameter combinations and then run a full model on the best one. However, there were concerns that the results from the subset of data were not generalizing to the full dataset. Therefore, it seemed preferable to compare many fully trained models : I.

III. Results

For the simple models, we will measure their accuracy on the training set and the testing set with and without the pre-preprocessing. All test scores were calculated by AI crowd.

The accuracy of all those models can be seen in Table I. If one were to make a naive classifier and assume all tweets had smiling faces ":)", the accuracy is 0.491, which is the baseline to compare all models' performances to. Simple models all showed about a 10 percent improvement over this baseline assumption, correctly classifying around 60 percent of all tweets. Non linear Support vector classifiers using gaussian kernel were better than the two linear (linear regression and SVC), correctly classifying 61.5 percent of the training data.

Neural network architectures saw vast improvements as compared to linear models. The two LSTM models have an accuracy close to 86 percent while BERT models can even reach an higher accuracy. The best performing model was by Cardiff roBERTa base which achieved an accuracy of 89.9 percent, about 40 percentage points over the naive classifier. Important to note is that all those neural network architectures saw data outside our training set. The LSTMs using word vectors pretrained from tweets, and all BERT models from huggingface are pretrained on large corpuses, while the cardiff models also used tweets [10]–[12].

In regards to multilingual BERT models (F and H) vs. english only BERT models (G and I), models trained on English language only outperformed those trained on multiple languages. One reason for this may be many different languages use different alphabet, and there were no tweets that had characters from different alphabets. Multilingual Bert was trained on Wikipedia articles from different languages which would have used the scripts associated with them. However, if tweets came from another language that had a different alphabet- it's possible they would have been transliterated to roman characters and potentially undetected. Again the languages used in the tweets were not possible to quantify.

IV. Discussion

The main goal was to improve the accuracy of the models for the task of text sentiment analysis. This task was done step by step by first implementing classic models such as linear regression and linear SVC. Nevertheless the accuracy was still low and we decided to implement non linear model

Model	Training Score (no P.) (%)	Training Score (with P.) (%)	Test (AICrowd) Score (with P.) (%)
(A) Linear regression	0.58992	0.593265	0.596
(B) Linear SVC	0.594985	0.59796	0.601
(C) SVC	0.619675	0.62467	0.615
(D) LSTM	0.8607	0.8794636	0.861
(E) BLSTM	0.8737728	0.8869512	0.863
(F) BERT base multilingual	-	0.925	0.880
(G) roBERTa base	-	0.918	0.892
(H) Cardiff xlm roBERTa base	-	0.924	0.894
(I) Cardiff roBERTa base	-	0.930	0.898

TABLE I
PERFORMANCE OF LINEAR AND NON-LINEAR METHODS FOR CLASSIFICATION OF TWEETS.

such as SVC with a gaussian kernels and deep learning models such as LSTM models and BERT transformer to improve it. Using SVC instead of Linear SVC didn't improve much the results but using deep neural networks with a more complex input confirms the utility of using non-linear architectures for this type of task. This is also consistent with the literature that has shown artificial neural networks outperforming more traditional methods [13].

Even so, 10 percent of tweets were misclassified even with the most performing model, meaning there is still room for improvement in these models. One thing that would potentially improve those models would be more training data. While many text classification objective aim to label tweets as "positive" or "negative (and sometime neutral), this task differs in that tweets may possess a ":)" and be negative, or vice versa [14]. For example the following tweet's true classification is ":)"-

<user> i just want to let you know that i hate you .

It is reasonable to assume that some annotators may classify it as negative without the smiley. The inclusion of the smiley suggests potential sarcasm that turns it positive, and therefore is the minimal span that is altered to change label classification. Sarcasm detection is a whole other classification task, and confounds the sentiment classification goal [15]. The models for sentiment classification have been pretrained on a slightly different task, so therefore it is possible that further finetuning with more relevant data, pretraining a BERT model to begin with on more relevant data would improve performance.

Additionally, with regards to sentiment analysis, there is often disagreement on classifications of tweets between human annotators [14]. These tweets, on the other hand, used to contain the emoji of their class, and therefore the true classification is completely known and not subject to personal biases.

V. ACKNOWLEDGEMENTS

Special thanks to the following tutorial for how to parallelize data on TPUs https://colab.research.google.com/drive/1dVEfoxGvMAKd0GLnrUJSHZycGtyKt9mrscrollTo=FyTR9V5jJWcS

Thanks to Dr. James Henderson at IDIAP whose course introduced me to pretrained huggingface models, and which helped guide me with the code and provided the "compute metrics" function used to evaluate the finetuning process.

REFERENCES

[1] Hirschberg Julia and Manning Christopher D., "Advances in natural language processing," *Science*, vol. 349, no. 6245, pp. 261–266, Jul. 2015, publisher: American Association for the Advancement of Science. [Online]. Available: https://doi.org/10.1126/science.aaa8685

[2] A. F. Dugas, M. Jalalpour, Y. Gel, S. Levin, F. Torcaso, T. Igusa, and R. E. Rothman, "Influenza forecasting with google flu trends," *PloS one*, vol. 8, no. 2, p. e56176, 2013.

[3] S. Yang, Z. Ning, and Y. Wu, "Nlp based on twitter information: A survey report," in *2020 2nd International Conference on Information Technology and Computer Application (ITCA)*, 2020, pp. 620–625.

[4] J. Eisenstein, "Natural language processing," 2018. [Online]. Available: https://github.com/jacobeisenstein/gt-nlp-class/blob/master/notes/eisenstein-nlp-notes.pdf

[5] Y. Bao, C. Quan, L. Wang, and F. Ren, "The role of pre-processing in twitter sentiment analysis," *In International conference on intelligent computing*, pp. 615–624, 2014.

[6] S. Hochreiter and J. Schmidhuber, "Lstm can solve hard long time lag problems," *Advances in neural information processing system*, pp. 473–479, 1997.

[7] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," *In Proceedings of the 2014 conference on empirical methods in natural language processing*, p. 1532.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

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

[10] F. Barbieri, J. Camacho-Collados, L. Neves, and L. Espinosa-Anke, "Tweeteval: Unified benchmark and comparative evaluation for tweet classification," 2020.

[11] F. Barbieri, L. E. Anke, and J. Camacho-Collados, "Xlm-t: A multilingual language model toolkit for twitter," 2021.

[12] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019.

[13] M. Zhou, N. Duan, S. Liu, and H.-Y. Shum, "Progress in neural nlp: modeling, learning, and reasoning," *Engineering*, vol. 6, no. 3, pp. 275–290, 2020.

[14] I. Mozetič, M. Grčar, and J. Smailović, "Multilingual twitter sentiment classification: The role of human annotators," *PloS one*, vol. 11, no. 5, p. e0155036, 2016.

[15] A. Joshi, P. Bhattacharyya, and M. J. Carman, "Automatic sarcasm detection: A survey," *ACM Comput. Surv.*, vol. 50, no. 5, sep 2017. [Online]. Available: https://doi.org/10.1145/3124420