Machine Learning Project 2
Text Classification

Ahmed Ben Haj Yahia, Mahdi Ben Hassen, Nour Ghribi
Department of Computer Science, EPFL, Switzerland

***Abstract*—Machine learning has made a huge improvement this last decade. Thanks to the rapid enhancement in computational power and fast data storage, new tools and techniques have been combined to create models that will give better use to the data. In this paper, we present an extensive study on the binary text classification using a set of data from Twitter. Our main goal is to present different machine learning methods to predict if a tweet message used contains a positive or a negative smiley. We will go through all the feature engineering and model training using classical machine learning algorithms and also deep learning.**

I. Introduction

Text classification, tagging or categorization by their content has gotten the interest of several machine learning researchers to help them understand millions of texts. Unstructured texts are everywhere, such as social media interaction, websites, emails etc..and companies are willing to invest a lot money to get good usage for all the data collected to receive key insight on users' thoughts. Being one of the most popular social medias, Twitter registers more than 500 millions tweets per day[1], and we will try, in this paper, to analyse a set oh them using Natural language Processing(NLP) and decide whether contain a positive or a negative smiley.
The report will be organized as follows: in section II we will be exploring the structure of the given data. Section III will be about the preprocessing steps..... Section IV .....

II. Data Analysis

A data-set of 2.5 million tweets was given to us by EPFL for machine learning course as text classification competition at AIcrowd[2], consisting of negative or negative smiley equally distributed for training and 10,000 tweets for testing. When inspecting the data, we can clearly notice a lot of anomalies, such as, abbreviated English, non formal words, hashtags, urls, special characters, emoticons etc. Getting rid of these anomalies is not a good approach since they could be good indicators, that's why we need to take more advanced approaches. To see that, we did group the tweets containing <user>and <url>by sentiment. The results are in figure 1 and 2 below.

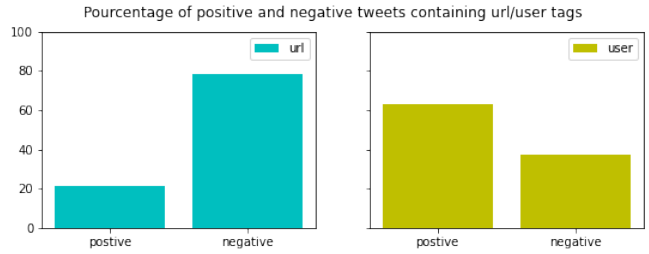

Figure 1. The correlation between the keyword <url>and<user>with the smiley

III. Data Cleaning

Although the data was found with a lot of inconsistencies, we did manage to clean it up and make it useful for our models. We describe below all the preprocessing steps for every single tweet:
Repetition handling: Words that are repeated in a row are removed since we can consider them as a mistake and they don't add much in the sentiment analysis.
Emojis translation: Since emojis are known to be used a lot in social media as important means of expression, we tried to map each one of them to its corresponding meaning. We did classify them into 7 different sets: happy, sad, laugh, love, kiss, wink and cry.
Dealing with hashtags: Hashtags are widely used on micro-blogging and photo sharing services such as Twitter that enables cross-referencing of content sharing. That's why it's more convenient to change the # by <tag>so that it's not considered as dummy word, and split the combination of words that comes after.
Removing numbers: We have chosen to remove the numbers since it does not play a significant role in sentiment analysis.
Punctuation removal: Punctuation is not important for the training of our models, since it's just a means of sending a coherent message, so we changed only the '..' and '...' by 'multistop' and '?' by 'question'.
Apostrophe treatment: Contraction is a shortened form of word that omits certain letters or sounds. In most contractions, an apostrophe represents the missing letters. We state as an example the contraction "i'm" that will be changed to "i am" for better training our models.
Stop words removal: Stop words such as [the, is, at etc.]

don't have a significant impact on sentiment so we decided to remove them.

Slang words treatment: Slang is very informal language or specific words used by a particular group of people. We dealt with it by importing a dictionary to change the slang words by their real meaning.

To deal with contractions, unpack hashtags and correct some spelling mistakes we used **EKphrasis** text processor and we feeds it **Pott's tokenizer**[3] : Pott's tokenizer conducts sentiment-aware Tokenization (i.e., by replacing emoticons and slangs with actual words). Since this process still had some inconsistencies, we downloaded some spelling correction dictionaries[4] and a Slang words dictionary[5] and used them to perform better correction.

IV. Feature engineering and vector representation

Most of machine learning algorithms are unable to process natural language, a conversion to a numerical representation is needed. Some of the techniques that we used for vector representation are:

TF-IDF[6]: Is an abbreviation for Term Frequency Inverse Document Frequency, it is a statistical measure that evaluates the importance of a word to a document in a collection of documents. TF-IDF values are proportional to the number of time a word exists in a document. The highest scoring words of a document are the most relevant to that document, and therefore they can be considered keywords.

Word embeddings[7]: They are in fact a class of techniques where individual words are represented as real-valued vectors in a predefined vector space. Each word is mapped to one vector and the vector values are learned in a way that resembles a neural network, and hence the technique is often lumped into the field of deep learning. We can always find pre-trained embeddings since computing them on our full data is unfeasible because it requires huge resources. That's why we decided to use **pre-trained Glove** embeddings which are trained in bigger data. But, Glove has a serious limitation. It does not take into consideration the context of words in a tweet since each word has only one fixed presentation. To get rid of this problem we used **ELMo** and contextualize word embeddings. ELMo looks at the entire sentence before assigning each word in it an embedding. It uses a bi-directional **LSTM**(an artificial recurrent neural network architecture(**RNN**)), trained on a specific task to be able to create those embeddings. Following our research on the different types of embeddings, we found the state of the art embedding model **BERT**[8], which stands for Bidirectional Encoder Representations from Transformers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks.

V. Models

A. Linear models

After preprocessing our data, we started by applying some linear models. Using *sklearn* library with the generated Glove embeddings or the pre-implemented TF-IDF Vectorizer. We first implemented Logistic regression using the generated GLOVE embeddings from the sample data. Secondly, we implemented Naive Bayes.This model applies Bayes theorem with a Naive assumption of no relationship between different features. We generate a dictionary containing all the words of our data as keys and we assign a value for each word. We then compute the sum of the values for the positive set of words, and respectively for the set of negative words. Then, we compute the log-likelihood of positive values (and respectively of negative values) for each word of the positive set and negative set data. Afterwords, knowing that the log-prior for positive and negative tweet is log(1/2), for a test tweet, we compute two values: the sum of the log-prior with the log-likelihood of each word ie:

$$sum\_pos = log(1/2) + log - likelihood\_pos.get(word)$$

$$sum\_neg = log(1/2) + log - likelihood\_neg.get(word)$$

Then by comparing these two values, we assign a sentiment to the tweet. Finally, we implemented Support Vector Machine using TF-IDF embedding.

See Table I that summarises the implemented linear models and their corresponding Accuracy and F1-Score from AIcrowd submissions plateform.

B. Neural Network models

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. Here is a list of the layers that we used in different models:

- **The Embedding Layer**: We initialise this layer with a minimum of 3 parameters: the maximum size of the vocabulary, the size of the vector space in which words will be embedded, and the length of the input sequence. By feeding this layer the embedding matrix from the pre-trained Glove word embeddings.

- **The Dense Layer**: Each neuron receives an input from every neuron in the previous layer, thus, is densely connected. We feed this layer a non linear activation function to map the resulting values into the desired range (between - 1 and 1, in the context of our project). We generally used sigmoid activation function since it led to better results.

- **The Dropout Layer**: In this layer, individual nodes are 'dropped out' of the net at random, with probability 1p,

so that a reduced network remains. This is necessary, as it prevents the problem of over-fitting.

*1) **Convolutional Neural Network models (CNN):*** A CNN is a class of deep neural networks, typically applied to analysing visual imagery. They are simply neural networks that use convolution in place of ordinary matrix multiplication in at least one of their layers.

- **The Convolutional Layer** : This is the core of CNN systems: it is used for feature extraction. Its parameters are the number of filters, the size of the filter and the activation function. In our case, we used the Convolution1D layer with 32 filters, a kernel size of 3 and Relu activation function

- **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.

- **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 a fully connected neural network classifier for the final classification.

*2) **Recurrent Neural Network models (RNN):*** A recurrent neural network is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior.

- **LSTM** : Long Short-Term Networks have feedback connections, unlike standard feedforward neural networks. They can equally process entire sequences of data, and not just single data points. We tried memory cells of size 30 or 100 , and dropout rates of 0.2 or 0.3

- **GRU** : Gated recurrent units are a gating mechanism in recurrent neural networks. Their performance on polyphonic music modeling and speech signal modeling was found to be similar to that of long short-term memory. They have fewer parameters than LSTM, as they lack an output gate. Similarly to LSTM, we used cells of size 30 or 100 , and dropout rates of 0.2 or 0.3

- **Bidirectional RNN** : Bidirectional Recurrent Neural Networks connect two hidden layers of opposite directions to the same output. With this form of generative deep learning, the output layer can get information from past (backwards) and future (forward) states simultaneously. We used as input to this layer, the output of either the LSTM layer or the GRU.

VI. RESULTS

Since the sample data can be representative of the full data, we decided, as a first step, to run our implemented models only on the sample data. Then, we used the full data on the ones that achieved better accuracy.

The table I shows all the linear models that we tested on sample data. Note that the best cleaning processes that we used were in this exact order: Removing repetitions, processing EMOJIs, numbers and hashtag, then tokenizing using Pott's tokenizer together with Ekphrasis (this will perform a better slang processing and spelling correction) then we use our dictionaries to perform another slang processing and spelling correction. As a last step we ommit stop words.

Models	Accuracy	F-1 score
Logistic Regression + GLOVE	0.609	0.631
Naive Bayes + GLOVE	0.75	0.758
SVM + TF-IDF	0.826	0.28

Table I
CLASSICAL MODELS TESTED ON OUR TEST DATA AFTER TRAINING THEM ON SAMPLE DATA

Having seen that SVM performed surprisingly high, we applied it to our full data-set of tweets. This leads to a 0.849 Accuracy and F-1 score of 0.853 (on AIcrowd). We will see that this score will outperform some of the deep network models that we implemented.

For neural network models, we tried several parameters for each layer and Table II summarise the chosen ones. [1]

Hyper-parameters	Value range	Best choice
Glove word embedding dimension	50,100,200,300	300
nbr of word per tweet	30, max_=49, 80	max_[1]
CNN number of filters	32, 64, 128	32
CNN Filter length	2, 3, 4	4
pool_length	2, 4	4
Dropout rates	0.1, 0.2, 0.5	0.2
Activation fn	relu, tanh, sigmoid	sigmoid , relu
Optimizer	Adam, SGD	Adam
Batch size	32, 64, 128	32,128

Table II
HYPER-PARAMETERS CHOICE

The RoBERTa [9] model is based on Google's BERT [8] model released in 2018. On the other hand, XLMRoBERTa [10] is based on Facebook's RoBERTa model released in 2019. It is a large multi-lingual language model, trained on 2.5TB of filtered CommonCrawl data.

The table III shows all the deep learning models that we tested. Note that multiple embedding techniques were used, and for each one, several models were tried. Again, the models that performed the better were submitted on AIcrowd

[1]max_ is the maximum number of words in a tweet from all tweets

Embedding	Models	Accuracy	Validation accuracy
Pretrained GLOVE	LSTM(30) + Relu	81.41%	79.50%
	BiLSTM(30) + Relu	81.36%	80.30%
	LSTM(30) + Sigmoid	83.67%	81.36%
	BiLSTM(30) + sigmoid	83.72%	81.48%
	GRU(30) + Relu	81.47%	78.82%
	BiGRU(30) + Relu	79.74%	80.25%
	GRU(30) + Sigmoid	83.12%	81.91%
	BiGRU(30) + Sigmoid	83.10%	82.10%
	Conv1D+MaxPool1D(GRU)+ Flatten	83.84%	81.54%
	Conv1D+MaxPool1D(GRU)+ BiLSTM(30)	84.19%	81.58%
	Conv1D+MaxPool1D(GRU)+ BiGRU(30)	84.28%	81.14%
Pretrained ELMO	Elmo default signature + Dense	79.90%	78.89%
	Elmo tokens signature + Dense	82.36%	82.34%
	Elmo tokens signature concatenated with Glove pretrained embeddings	81.89%	82.34%
Pretrained BERT	XLMRoBERTa For Sequence Classification	87.05%	88.47%
	BERT For Sequence Classification	87.33%	88.45%
	RoBERTa For Sequence Classification	x	x

Table III
DEEP LEARNING MODELS TESTED ON OUR TEST DATA AFTER TRAINING THEM ON SAMPLE DATA

Again, only the most interesting models were run on the full data. On AICrowd, the best model using pretrained Glove was Bidirectional LSTM(30) with Sigmoid activation function resulting in an accuracy of 84.9% and F-1 Score of 84.8%. We achieved the best scores using BERT models. XLMRoBERTa produced an accuracy of 87.7% and F-1 score of 87.8% on AICrowd. The basic BERT outperformed it with an accuracy and F-1 score of 88.3%. Finally, the best score on AICrowd resulted from RoBERTa for sequence classification with an accuracy of XXX and F-1 score of XXX

VII. CONCLUSION

Throughout this project we implemented a baseline model using classical ML methods. Following this, and searching for more advanced features representations, we discovered more interesting embeddings such as Elmo and BERT. We devised some deep learning models using these embeddings, although we didn't train the embeddings on our data (which requires more computational resources) we got some promising results with the best model being BERT having 88.3% as accuracy and F-1 score on the provided test set. We took a shot at playing with some of these models to try and fit an ensemble model as we have seen in the paper Transformer based Deep Intelligent Contextual Embedding for Twitter sentiment analysis **(DICET)**[11] by just combining between ELMO and GLOVE embeddings but it didn't outperform BERT. Provided more time and computing power would be necessary to explore some advanced models such as bagging or combining trained models. Recent advances in NLP are very promising such as **GPT-3**[12] as it uses deep learning to produce human-like text. We are looking forward to explore its limitless expansion.

REFERENCES

[1] Twitter. https://fr.wikipedia.org/wiki/Twitter.

[2] Aicrowd. https://www.aicrowd.com/challenges/epfl-ml-text-classification.

[3] Pott's tokenizer. http://sentiment.christopherpotts.net/code-data/happyfuntokenizing.py.

[4] Spelling correction dictionary. http://people.eng.unimelb.edu.au/tbaldwin/etc/emnlp2012-lexnorm.tgz.

[5] Slang correction dictionary. http://luululu.com/tweet/typo-corpus-r1.txt.

[6] Tf-idf. https://monkeylearn.com/blog/what-is-tf-idf/.

[7] Word embeddings. https://machinelearningmastery.com/what-are-word-embeddings/.

[8] Bert. https://arxiv.org/abs/1810.04805.

[9] Roberta: A robustly optimized bert pretraining approach. https://arxiv.org/abs/1907.11692.

[10] Unsupervised cross-lingual representation learning at scale. https://arxiv.org/abs/1911.02116.

[11] Transformer based deep intelligent contextual embedding for twitter sentiment analysis:. https://doi.org/10.1016/j.future.2020.06.050.

[12] Gpt-3:. https://openai.com/blog/openai-api/.