Machine Learning Project 2 : Twitter sentiment analysis

Mehdi Zayene, Augustin Henry, Juliette Challot
Department of Computer Science, EPFL, Switzerland

***Abstract*—Natural Language Processing (NLP) is the understanding, manipulation and generation of natural language by machines. Sentiment analysis is one of the problematics tackled by NLP: it consists in identifying subjective information from a text to extract the author's opinion. This paper presents preprocessing and machine learning techniques that can be used to perform sentiment analysis on data extracted from Twitter.**

I. Introduction

This projects explores different ways to perform sentiment analysis on natural text data extracted from Twitter and compares the performances of each model. Twitter is a famous social network actively used by more than 313 millions users monthly, there are around 500 millions tweets published each day in more that 40 languages.The data used for this project contains English tweets in which an emoji :) or :(used to be present but got removed. The goal is to recover this information by applying different machine learning algorithms on the text data. In this paper are presented the exploratory data analysis, preprocessing steps, feature extraction and machine learning methods explored to recover the sentiment (positive for :) and negative for :( ) with the highest accuracy possible. Finally, the solution that gives the best accuracy on the testing set is presented.

II. Exploring data Analysis

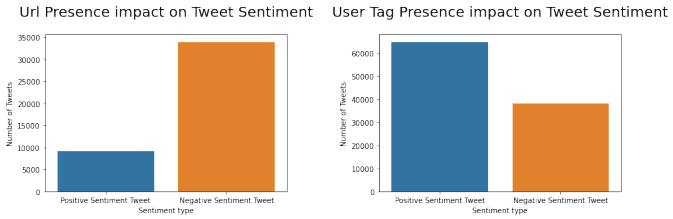
A necessary first step is exploring and understanding the data given. In this case the data is split into 5 sets : two small datasets containing 100'000 tweets each and two large datasets containing 1.25 million tweets each; for both, one contains positive tweets and the other contains the negative ones. Finally we have a testing set containing 10'000 tweets. Taking a closer look at the data was the basis of the data processing. Indeed, as twitter is a social network that limits the number of characters, it was predictable that the text contained non formal words and structures. Notably slang words, special characters such as emojis, abbreviated or misspelled words, hashtags and more lexically incorrect structures were noticed. The following section describes all the preprocessing operations that were intended on the data.

III. Processing and transforming the data

Preprocessing can be key to reveal the information that lies within the data, especially in NLP. In the following sections, we describe all the techniques that were used, tried and possibly dropped to improve the quality of the information fed to our models.

A. Data cleaning

1) ***user and url****:* Many tweets contain the word ¡user¿ or ¡url¿, before removing them altogether, the number of occurrences of these words in the positive (resp negative) dataset was counted. The following graph show the results :



Given these results, we decided not to remove them.

2) ***Punctuation****:* Punctuation can sometimes emphasis an emotion. Thus, we made a function to turn some punctuation repetitions into words. We also made a function that removes all punctuation signs.
ex: 'Wait ... how ?' -¿ 'Wait consecutiveStop how'.

3) ***Character Repetition****:* Many words are misspelled intentionally by repeating one or several characters. Thus, a function was implemented to check that a word contained at most two consecutive occurrences of the same letter. If not, the repetition gets replaced with only two repetitions.
ex: lllooooooove $\rightarrow$ lloove

4) ***Emojis****:* Emojis commonly used in tweets and can help in the sentiment analysis. A function was therefore made to turn the most current ones into one of the following words: 'Positive', 'Negative' or 'Love'.

5) ***Hashtags****:* Hashtags are common practice in social networks. There are a way to underline a particular word or combination of words by writing '#'+ 'word or word combination without space'. A function was designed to remove the '#' sign and split the words written afterwards, if necessary, with the most likely word combination.
ex: '#MLisfun' $\rightarrow$ 'ML is fun'

6) ***Apostrophe****:* Apostrophes in English is used for contractions of words. A function was designed to remove these apostrophes and recover the full expression.
ex: 'hasn't', 'hasnt' $\rightarrow$ 'has not'

7) ***Slang Words****:* Slang is an informal type of language commonly used in tweets. During data exploration, we established a dictionary to map the most common ones to their formal equivalent. A function was designed with this

dictionary along with another one using gingerit library [1]. While this library achieved far better result than textblob and was more complete than our own dictionary, it was also extremely computationally costly and some common twitter abbreviation were not corrected.
ex: first result is with the dictionary, second is with gingerit
'idk I luv dat ML stuf' → 'do not know i love dat ml stuf'
→ 'Idk, I love that ML stuff'

*8) **Short Word**:* We estimated that words of length one could not carry much weight in the sentiment analysis, a function to remove them was therefore designed.

*9) **Numbers**:* Similarly a function was written to remove numbers as they don't carry much sentiment information.

*10) **Misspelled Words**:* To correct the spelling of the words, a spell corrector dictionary from ekphrasis was used.
ex: 'wrds misspleled' → 'words misspelled'

*11) **Non alphabetic Words**:* A function that removes none alphabetic word ie words containing a number or a special character was designed.
ex: '4real you go at 6am ?' → 'you go at'

*12) **Stop Words**:* Some stop words can help identify the sentiment, for example 'not' is a stop word and removing it could have a bad impact (ex : 'not happy' would become 'happy' and the tweet may be misinterpreted as Positive). Thus a list of 'useful' stop words was established and a function was implemented to remove all stop words except the ones in that list.
ex: 'You are not happy as today' → 'You not happy today'

*13) **Stemming**:* Stemming is the process of reducing a word to its root. This operation uses nltk stem 'PorterStemmer' [2].
ex: studies studying studied → studi studi studi

*14) **Lemmatizing**:* Lemmatizing is another way to reduce word only it reduces the word to its smallest existing form. Two libraries were tested : wordNet [3] and Pattern lemmatizer [4]. As shown in the following example the Pattern lemmatizer yield better result but it was also way more computationally costly.
ex: first result is from WorldNet and second is from Pattern
'studies studying studied' → 'study studying studied'
→ 'study study study'

*15) **Emphasising sentiment**:* To emphasis sentiments, an idea that was explored is to used positive and negative word dictionaries by writing the word 'positive' (resp negative) next to a word that belong respectively in those dictionaries. Dictionaries from harvard [5], Hu and Liu [6] and Loughran McDonald [7] were used to implement the function.
ex: 'Progress makes us happy' → 'positive progress makes us positive happy'

*16) **Ekphrasis**:* A final attempt to improve the quality of the dataset was made using Ekphrasis [8] text preprocessor. This performs multiple actions including unpacking hashtags, contractions, correcting spelling mistakes and replace word with root. This option was very efficient in term of run time and performances.

IV. Vector representation

Machine learning algorithms can not perform directly on natural language, they need the data to be passed through an embedding layer. Word embedding reduces the dimension and capture the context, semantic and syntactic similarities (gender, synonyms,...) of a word. There exist word embedding techniques, the following subsection briefly describe the one implemented in this project.

*1) **Tf-Idf**:* This stand for Term Frequency-Inverse Document Frequency. Term frequency counts the frequency of a word in a document, this is highly depends on document length. In Tf-Idf, this value is multiplied by the logarithm of the inverse document frequency which measures the informativeness of term (low value for word that appears very often and higher value for a word that is rare). We used scikit-learn [9] to implement it.

*2) **Word2Vector**:* This algorithm is implemented using gensim [10]. It consists of the following steps :

- Create data tuples of input and output words, each word is represented as a one hot vector. The difficulty consist of finding the good vector dimension and the window's size to get the context. Having a larger window often yields better performances but the computational cost increases tremendously.
- Define a model that can take the one hot vectors as inputs and outputs, to be trained. We use an embedding matrix of size $[vocabulary\_size * embedding\_size]$ that stores the words found in the vocabulary.
- The neural network will simply train as follows : for a given input, it will find the corresponding word. Then try to predict the output word. By comparing the prediction, compute the loss. Using the loss, the model optimise itself with a stochastic optimizer.
- For the loss function we used the cross entropy defined in the TensorFlow library.

*3) **Glove [11]**:* The word2vec algorithm aforementioned relies only on local statistics ie the local context information of words. GloVe is a word embedding technique that also incorporates the global statistics ie the word co-occurrence. Our training datasets might not be large enough to capture all the nuances of tweet sentiment analysis so we decided not only to build a word embedding with our data but also to load a pre-trained twitter dictionary from the glove database to see if the performances changes. We ran a 4 layer convolutional network on both word embeddings (pretrained and our own) for the a few preprocessing steps combinations to try to optimise our results. The best results we achieved are on validation set are summarised in the Table I. As one the can see, using the pretrained corpus yields better results that creating our own word embedding. Note that the vector dimension used to do these tests was 50 and the pretrained

	2nd Best preprocess	Best preprocess
pretrained glove	83.21%	83.31
glove	82.65%	82.52

Table I: Results for glove embbedings and CNN algorithm

corpus exist in versions 25, 50, 100 and 200 so we can still improve our result by using a larger vector size (but at computational cost).

V. Machine learning techniques

A. *Models*

Using sklearn library, we tried to perform different machine learning algorithms with different preprocessing schemas. We used Tf_Idf vectorizer and classifiers used briefly presented in the following subsections. The best processing method and a summary of the results obtained is presented in Table II.

*1) **Logistic Regression**:* Logistic Regression is a predictive analysis algorithm and based on the concept of probability, it uses the sigmoïd function as cost function.

*2) **SVM**:* A Support vector machine (SVM) is a supervised learning method that outputs a map of the sorted data with the margins between the two as far apart as possible.

*3) **SVM with L1-based feature selection**:* This model is the same as the previous one but the norm used in the penalisation is different and leads to sparse coefficient vectors.

*4) **Multinomial NB**:* Multinomial Naive Bayes is a supervised learning classification algorithm that implements the naive Bayes algorithm for multinomially distributed data. It assumes feature independence.

*5) **Bernoulli NB**:* BernoulliNB implements the naive Bayes training and classification algorithms for data that is distributed according to multivariate Bernoulli distributions.

*6) **Ridge Classifier**:* This classifier first converts the target values into -1, 1 and then treats the problem as a regression task.

*7) **AdaBoost**:* An AdaBoost classifier fits a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

*8) **Perceptron**:* A perceptron is a single-layer neural network used for supervised learning of binary classifiers.

*9) **Passive-Aggresive**:* A Passive Aggressive Classifier is an online learning algorithms that responds as passive for correct classifications and as aggressive for miscalculations.

*10) **Nearest Centroid**:* In Nearest centroid, each class is represented by its centroid, with test samples classified to the class with the nearest centroid.

B. *Choice of processing and N-gram optimization*

Several tests were performed to find the processing combination that yielded the best results while having a reasonable computational complexity. We ran further tests on our two best preprocessing method (according to validation accuracy). Especially we tried to train the model on different N-gram (for 1-gram to 5-gram) and on all the previously cited classifiers to see the difference in accuracy : the results are shown in Tables 2. In the end, we achieved better results by combining : removing character repetitions, removing apostrophes and using ekphrasis text processor. Although some 4 or 5-gram achieved great accuracy, we chose to only work with N-grams up to 3 because the change in computational cost seemed unreasonable compared to the accuracy improvement. For the two best preprocessing methods, we ran 2-gram and 3-gram on the full dataset. In doing so we noticed that on the full dataset, 3-gram achieved slightly better performances than 2-gram. The best accuracy values are shown in Table II for each model. .

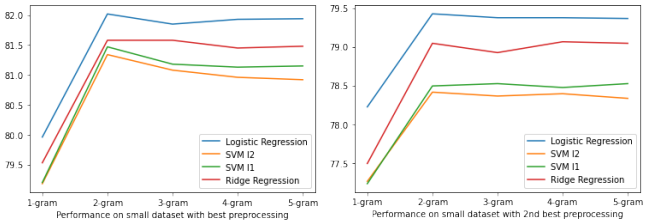


Figure 2: Accuracy achieved with different N-grams

C. *Results*

Validation accuracy and run time of all ML algorithms implemented are shown in the following table. Validation set represents 20% of the total training set. Our most promising model; Logistic Regression, was submitted on AIcrowd and we obtained a 85.4 accuracy.

algorithm	accuracy	time
Logistic Regression	84.38%	426.56s
Linear SVM l2 norm	84.23%	268.48s
Linear SVM l1 norm	84.24%	568.57s
Multinomial NB	80.23%	235.66s
Bernoulli NB	78.26%	257.96s
Ridge Classifier	84.19%	253.74s
AdaBoost	73.03%	1133.33s
Perceptron	78.32%	271.25s
Passive-Aggresive	81.83%	308.70s
Nearest Centroid	69.82%	224.43s

Table II: Results of tested ML algorithms

VI. Neural Networks

Deep and large pre-trained language models are suitable for various natural language processing tasks. In this part of the project we took advantage of existing models and did transfer learning. After several investigations, we noticed

that the best performances came up with state-of-the-art models BERT and Xtreme on raw data without any further pre-processing (except removing duplicate tweets as well as empty ones). However, the huge size of these kind of models could be a deterrent to use them in practice, thus we decided to use distilled models with a good trade-of between speed and average accuracy.

A. *CNN LSTM*

The Convolutional Neural Network (CNN) Long Short-Term Memory network (LSTM) is an LSTM architecture specifically designed for sequence prediction problems with spatial inputs. We use an embedding layer to represent our data, every tweet has a padding so they have the same length, this is our spatial inputs. Then a pooling layer, trying to represent a maximum the context of the words. The next layer is the LSTM layer with memory units (smart neurons). Finally, because this is a classification problem we use a Dense output layer with a sigmoid activation function to make 0 or 1 predictions for the two classes (positive or negative) in the problem. The accuracy achieved with this method is written in Table III.

B. *DistilBERT*

DistilBERT is a transformer model, smaller and faster than BERT, a state-of-the-art transformer model used in NLP. DistilBERT was pretrained on the same corpus in a self-supervised fashion, using the BERT base model as a teacher.

1) ***DistilBERT base model (uncased)*** *[12]:* In order to take advantage of its full potential, we re-trained DistilBERT 4 outer layers adding also additional Linear layers with Relu as activation functions (along with dropouts with a rate of 0.3). The last layer has output size of 2 and as we use a CrossEntropy Loss (probabilistic approach), we take the highest value of the two as prediction. The same logic was applied to all the other Deep Learning models.

2) ***DistilBERT base model (cased)*** *[13]:* Unlike the uncased version of DistilBERT, the cased version keeps uppercase letters without transforming them with BERT encoder. According to the documentation this is the only difference with the previous model so we decided to keep the exact same architecture of our hybrid model too.
Not surprisingly, capital letters play a significant role in sentiment expressions, a better performance was achieved (see Table III)

C. *XtremeDistil*

XtremeDistilTransformers is a distilled task-agnostic transformer model that leverages task transfer for learning a small universal model that can be applied to arbitrary tasks and languages.

1) ***XtremeDistil-l6-h384*** *[14]:* This model variant comes up with 6 layers, 384 hidden size, 12 attention heads corresponds to 22 million parameters with 5.3x speed-up over BERT-base. To compute our transfer learning we re-train its last layer adding two additional Linear layers with Relu as activation functions (along with dropouts with a rate of 0.3).
The training was done over 5 epochs.

2) ***XtremeDistil-l12-h384*** *[15]:* Unlike the previous variant Distil Model, this one was training over only 3 epochs due to its long training time.

Methods	Retrained	Epoch	l. rate	acc
DistilBERT base model (uncased)	4 layers	3	1.e-04	0.825
DistilBERT base model (cased)	4 layers	3	1.e-04	0.85
XtremeDistil-l6-h384	1 layers	5	5.e-05	0.889
XtremeDistil-l12-h384	1 layer	3	5.e-05	0.895
CNN LSTM	-	3	1.e-04	0.83

Table III: Comparison of tested pretrained models and the CNN LSTM

CONCLUSION AND FUTURE WORK

To conclude, sentiment analysis can be done with many different techniques, some however achieve far better results that others. Nevertheless, this often comes at a price in terms of computational complexity : some models or even preprocessing steps couldn't be run of the full dataset due to their high complexity and we used google Colab for the most promising algorithms to make the process faster. Overall we achieved better results with state-of-the-art neural network techniques although Logistic regression achieved great results as well. One thing we noticed is that sometimes, learning without any or very few preprocessing steps on the data achieves better results than learning on data where we try to correct as many mistakes as possible. This was indeed the case for Bert and XtremeDistil algorithms for example. In the end we achieved a result a 89.5 accuracy on AIcrowd. Future work could involve trying models trying out more variations of theses state-of-the-art methods and fitting them more to our data.

REFERENCES

[1] T. K. ¡tim.kleinschmidt@gmail.com¿. [Online]. Available: https://pypi.org/project/gingerit/

[2] [Online]. Available: https://www.nltk.org/howto/stem.html

[3] [Online]. Available: https://www.nltk.org/howto/wordnet.html

[4] Pattern for python, journal of machine learning research (2012). [Online]. Available: https://github.com/clips/pattern

[5] [Online]. Available: http://www.wjh.harvard.edu/~inquirer/homecat.htm

[6] Opinion mining, sentiment analysis, and opinion spam detection. [Online]. Available: https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html#lexicon

[7] Lougham and McDonald. [Online]. Available: https://sraf.nd.edu/textual-analysis/resources/#Master%20Dictionary

[8] C. Baziotis, N. Pelekis, and C. Doulkeridis, "Datastories at semeval-2017 task 4: Deep lstm with attention for message-level and topic-based sentiment analysis," in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Vancouver, Canada: Association for Computational Linguistics, August 2017, pp. 747–754.

[9] [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[10] [Online]. Available: https://radimrehurek.com/gensim/models/word2vec.html

[11] [Online]. Available: https://nlp.stanford.edu/projects/glove/

[12] [Online]. Available: https://huggingface.co/distilbert-base-uncased

[13] [Online]. Available: https://huggingface.co/distilbert-base-cased

[14] [Online]. Available: https://huggingface.co/microsoft/xtremedistil-l6-h384-uncased

[15] [Online]. Available: https://huggingface.co/microsoft/xtremedistil-l12-h384-uncased