Machine Learning - Project 2

Arthur Passuello, François Quellec, Lucas Strauss
Department of Computer Science, EPF Lausanne, Switzerland

***Abstract*—Natural Language Processing and more specifically Sentiment Analysis are subjects arousing a lot of interest in the scientific community. Due to the many practical direct applications that new discovery in this field could have, there already is a vast amount of literature and tools available for anyone willing to take a peek and work on it. In parallel, huge amounts of textual data in natural languages is one click away to anyone and can be used as laboratory mice, one of such in the *tweets* that are generated by billions of users everyday. This papers aims to describe the methodological process of defining an optimal model to predict sentiment in *tweets*. Exploring, using and combining existing literature and available software tools to create the best suited tool such as *GloVe* and *Word2Vec* to represent the data and *SVM* or *Recurrent and Convolutional Neural Networks* as model to be trained.**

I. Introduction

Natural languages are all languages that are intrinsically implicit and ambiguous, as opposed to *formal languages* that are always unequivocal and explicit. The study of solutions for defining an automatic process of extracting qualitative data from a spoken or written text is called *Natural Language Processing* (NLP) and it has been a very active topic in the recent years. The complexity of natural languages and their interpretation dependency on the context makes NLP as challenging as it is promising in the many direct applications any tool capable of such things could provide. Among the sources of data containing natural languages are the gigantic amounts of *tweets* generated every day by `Twitter` users, and among the useful applications of NLP on these data is predicting the expressed *sentiment* behind every tweet, known as *Sentiment Analysis*. Using a maximum of 280 characters, every `Twitter` user regularly express ideas or reactions to events and many parties show great interest in finding a way to globally understand, estimate and even predict the public reactions or opinions.

Over the years, many high-performing tools and methods were developed by renowned institutions and are freely available, for that reason, this paper detail the exploration several of these, from the pre-processing of the data to the final trained prediction model, including the creation of the features representing the *tweets*. Each will then be discussed, followed by the final results that were obtained and what could still be done to go further.

II. Dataset Preprocessing

Before being able to process the data, it must be cleaned, to remove any noisy inputs (such as typos), and homogenized, in order to ensure that all similar words are represented the same and the structural likeness is properly represented. Based on the Stanford pre-processing methods for feeding their glove embedding solution, we proceeded with the data-set as follow:

- All @ mention are replaced by the token $< user >$
- All number are replaced by the token $< number >$
- The remaining smileys are categorized into 3 tokens:
  - $< smile >$
  - $< sadface >$
  - $< neutralface >$
  - $< heart >$
  - $< brokenheart >$
- When a hashtag is detected, the # is removed and a $< hashtag >$ token is inserted in front of the word
- Capitalized word are lowered and an $< allcaps >$ token is added
- Repeated punctuation (more than twice) are truncated to 1 occurrences and a $< repeat >$ token is added
- Repeated letters are truncated to 1 occurrences and a $< repeat >$ token is added at the end of the word

Tweet1	"We loveeeeeeeee mr. JAGGI <3"
Tweet2	"We #loveeee mr. jaggi !!!!!!!!"
Result1	"love <elong> mr . jaggi <allcaps> <heart>"
Result2	"<hashtag> love mr . jaggi ! <elong>"

Fig. 1: Example of *Tweet* processing

One can easily see in the example in Figure1 the transformations made by our methods on some original *tweets*. It clearly illustrates the importance of such work as, for example, the two semantically similar words in the *tweets* in Figure1, initially represented quite differently, share now the same representation. Furthermore, the work on the *smileys*, significantly linked to emotions and the scope of this work, are now represented in a way that can be interpreted as any other word. Indeed, the dataset was initially freed of all basic smileys but a lot of them still remains. Finally, the concept of *hashtag* is now properly represented, like any part of a sentence in english and the words contained in the hashtags can properly be associated to real language.

III. Word embedding

As a way to provide the learning model a representation of *tweets* adapted to numerical processing, words and sentences can be turned into a vector of real numbers. A word would be represented by vector of fixed length, called *embedding* and a *tweet* can be represented as a composition of all the vectors from the words it contains.
In this project we will focus on comparing 3 different techniques of embedding :

A. GloVe

For "Global Vectors" is a method developed at Stanford to obtain a vector representation of every words in a provided corpus. Given only the desired length of each vector and the whole experiment corpus of text, a learning model is used to create a vector representation for every word specific to their usage and meaning in the provided text.

1 - Pre-Trained GloVe vector
A pre-trained corpus of vector/words is made available by Stanford. Trained on roughly 2 billions *tweets*, it is used as reference.

2 - Self-Trained GloVe vector
Using the experiment set of tweets and the stanford glove

learning algorithm, a custom corpus of word/vector is defined, more specific to the data at hand.

Both methods will be used in cross validation with the following sizes:

- 50
- 100
- 200

B. Word2Vec

`Word2Vec` is a selection of related models - developed and patented by *Google* [1]- used to generate word embeddings from a corpus of text and a desired vector length. The default models available through the `gensim` python library are *Continuous Bag of Words* (CBoW) and *Skip-gram*, only the first was used in this study. CBoW uses a 1 hidden layer neural network that is fed tokenized *tweets*. Using those, it tries to optimize predictions vectors for each word using the context in which it was provided.

C. N-Grams

N-Grams of texts are a set of co-occurring words (with every element of length N). We used the *gensim* library and the `Phrase` class to produce a set *Bi-grams* on the entire set of tokenized *tweets*. Then, for every tokenized *tweet* in the dataset, all tokens that generated a *bi-gram* are replaced by it, thus emphasizing further the semantic link between words. Furthermore, stacking together terms often used together can help avoid ambiguity inherent to word usage in common expressions. A good toy example could be the expression *"tough love"* and the word *"love"*, one has a negative connotation whereas the other has a positive one but, if used frequently enough, the `N-Gram` constructor is able to create the *bi-gram "tough-love"* which will then be considered as a word in itself, detached from the word *"love"*. The resulting set of tokenized *tweets* is used to create more decisive word embeddings, in this case using `GloVe`.

IV. Features Engineering

In an attempt to improve even further the training set of embedded *tweets*, some features were created but none of them brought a significant improvement in the final predictions and none were retained in the final model. Those were implemented as follow :

- ***Tweet* length** after noticing that "positive" *tweets* were significantly longer than "negative" ones, an additional feature was added for the length of each sample.
- **Number of Hashtag** a similar observation was made for the number of hashtag that was generally higher for "positive" *tweets* in comparison to "negative" ones. An additional feature was then added for the count of hashtag in each sample.

V. Model

Support-Vector Machines

Support-vector Machines [2] (SVM) are a set of supervised learning models used for linear classification and regression analysis providing higher accuracy performances than other algorithms [3] by simultaneously minimizing the empirical classification error and maximizing the geometric margin using only a few data points. The first significant parameter of a SVM is the choice of its kernel (e.g. linear, polynomial, Radial Basis Function (RBF), sigmoid, etc.) and it mostly depends on the type of data it would be used onto. Then, for each of those, inherent parameters must be tuned to reach optimal performances.

In this research, the `SVC` class from the library `scikit-learn` was used and simply using the desired kernel provides an already finely tuned set of parameters.
Finally, the SVM algorithm performs a linear regression and tries to minimize the energy function, with ϕ the kernel function, n the number of sample, λ the regularization term, w the weights, y_i the output data and x_i the input data points, is defined as follow:

$$\frac{1}{n}\sum_{i=1}^{n}\phi(1-y_i(w\cdot x_i-b))+\lambda\|w\|^2 \qquad (1)$$

Bidirectional Long Short-Term Memory

A basic *Long Short-Term Memory* (LSTM) is a *Recurrent Neural Network* (RNN) composed of cells with input and output gates as well as a forget rate which allows the model to capture time dependent series. This adaptive memory usage during recurrence with neural attention offers a way to induce relations between the input tokens, which is particularly relevant to the case of this study. To go even further in the attempt to extract semantic links among the input tokens (i.e. words in a *tweet*), a more specialized concept, BLSTM was considered.
A *Bidirectional LSTM* (BLSTM) is an extension of the *Long Short-Term Memory* [4] (LSTM) that connects two hidden layers of opposite directions to the same output as opposed to only one for the LSTM [5], as illustrated in Figure2. This additional connections allows exploitation of long-range contextual information which, in the scope of this research, significantly helps refining the prediction of past observation with new contextual data [6].

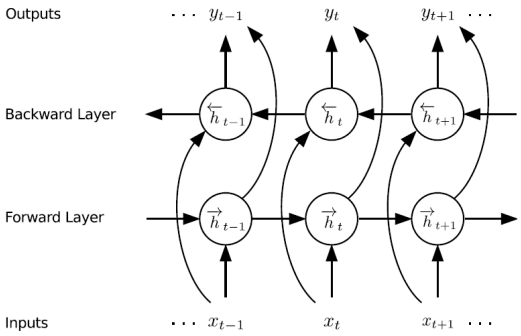

Fig. 2: BLSTM Network Schema [7]

Using this topology, the most significant parameters of the RNN are then the width of the input and output layers as well as the composition of hidden layers between them. In this research, the `LSTM` and `Bidirectional` classes from the library `tensorflow` were used, among other layer-specific classes.

Depth-wise Separable Convolutional Neural Network

Depthwise Separable Convolutions are convolutions that not only deal with the spatial dimension but also with the channel dimension, i.e. the number of channel composing the data. Such a convolution is composed of two kernels,

each with his own convolution function. First the depthwise (i.e. channel-wise) convolution, and then the pointwise convolution, merging the channels together.

A *Separable Convolutional Neural Network* [8] (sepCNN) are composed of two Separable Convolutional Layers, i.e. each layer consists of two convolutional layers - layers in which each neuron is connected to every neurons in the next layer - stack behind each other -the two layers correspond respectively to the two convolution kernels. The first layer (i.e. depthwise convolution kernel) operates on the spatial dimension and acts on the channels independently whereas the second layer corresponds to the pointwise convolution kernel, acting only on the channel dimension [9].

In the scope of this study, the major strength of this model lies in the use of multiple kernels that operate the convolution in steps rather than all at once. This trick reduces the number of operations to carry out, thus the training time.

An implementation of the model is provided by Google [10]. Considering the complexity of the model and the work that was already done on the model that was used, most parameters were kept as is and only the dropout rate parameter was considered.

VI. Method

Due to the complexity of the models considered in this study, it was decided to rely on the literature when looking for optimal values for most of each model hyper parameters.

A. Fine-tuning of hyper parameters

SVM

Among the several kernel that were considered, only the RBF was retained as it seemed well adapted to the data. It indeed non-linearly maps samples into a higher dimensional space, as opposed to linear kernels and it presents less numerical difficulties than other kernels.

BLSTM & sepCNN

Most parameters were kept as found in the relevant literature. However, upon realization that the models over-fitted quickly after a few epochs, it seemed relevant to explore the *dropout* parameter - designed specifically to address that matter. For both models, test were made for the values 0.2, 0.4 and 0.6.

B. Model tweaking

In an attempt to improve accuracy performances for the neural networks, the models found in the literature were modified by adding hidden layers. Among other attempts, adding `Flatten` layer between other hidden layers to increase the dimension of the data and potentially enable an opportunity for the two classes to further set themselves apart was a promising lead. However, all attempts resulted in reduced accuracy and were thus discarded.

C. Embedding engineering

This is where most exploration efforts were placed as this is where this study had the most potential to improve the case-specific parts of the problem, i.e. the data. As detailed in Section III, several embedding methods were considered, i.e. *GloVe* - trained on the data at hand and the pre-train Stanford embeddings, *Word2Vec* and *Bi-gram* (a derivation of *GloVe*).

Initially, the performances were evaluated with embedding of dimension `N=50` for the 4 methods on all 3 models in the hope that a small dimension would be enough for the best candidate to stand out and avoid high computational time for cross-validation.

After this first phase, higher dimensions were tested, up to $N = 200$, on the best model candidate.

1) Final Results Evaluation: As the final step of exploration, given that accuracy was the main goal of this study, predictions on the unlabeled data were computed and uploaded to the *AICrowd* platform to assess the performance scores from an external, and thus unbiased, source on unknown data.

VII. Results

A. Fine-Tuning of Hyper-Parameter

As a result from the work described in section VI, a single significantly improving modification from the values found in the literature was determined. Indeed, modifying the `dropout rate` parameters from the *BLSTM* and *sep-CNN* models, in order to reduce the most over-fitting. As the graph in Figure 3 shows, a value too small, here 0.2, limits accuracy performances no matter the number of epochs. However, when doubling this value and setting it to 0.4- that is randomly dropping close to half of the network's units during training- not only is the accuracy directly improved but it keeps improving with the number of epochs. Besides, it also shows that increasing further this rate up by the same to 0.6 does not decrease the accuracy performances in comparison to the previous rate, it shows however no significant improvement either.

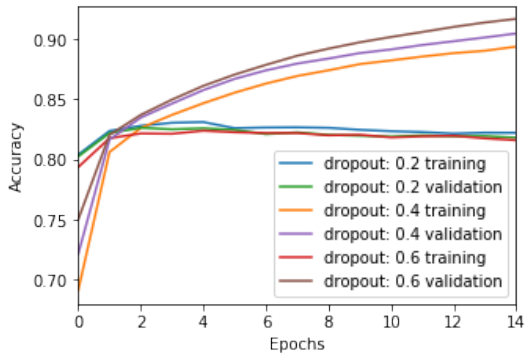

Fig. 3: Variation of the dropout parameter on a BLSTM model using the pre-trained GloVe and dimension of 50

B. Embedding engineering

Here are we used the optimal dropout rate of 0.2 to test the different embedding dimensions and n-grams.

Model	n-Gram	Embedding	Train acc	Val. acc.
SVM	1-gram	pre-train GloVe	-	**0.8036**
	1-gram	self-train W2V	-	0.7869
	1-gram	self-train GloVe	-	0.7943
	2-gram	self-train GloVe	-	0.7914
BLSTM	1-gram	pre-train GloVe	0.9169	0.8223
	1-gram	self-train W2V	0.905	0.8106
	1-gram	self-train GloVe	0.8786	0.8374
	2-gram	self-train GloVe	0.8807	**0.8431**
sep-CNN	1-gram	pre-train GloVe	0.8733	0.8295
	1-gram	self-train W2V	0.8232	0.8284
	1-gram	self-train GloVe	0.8820	**0.8389**
	2-gram	self-train GloVe	0.8725	0.8321

Fig. 4: Training and Validation Accuracy of the 3 Models for Different Embeddings (N=50)

We can see here that, BLSTM and sep-CNN performs both very well compare to SVM, both Neural Networks need to be considered for the final predictions. Using bigrams is also very promising.

C. Final Results

Based on the previous results we have retained the following two models: BLSTM and sep-CNN. We then trained them using an embedding size of 200 for the Unigram self trained sep-CNN and BLSTM, and 100 for the other methods because the computation was much slower.

Model	n-Gram	Embedding	Results
BLSTM	1-gram	self-train Glove 200	0.865
BLSTM	2-gram	self-train Glove 100	0.871
BLSTM	1-gram	pre-train Glove 100	0.854
sep-CNN	1-gram	self-train Glove 200	0.875
sep-CNN	2-gram	self-train Glove 100	0.870

Fig. 5: Final Accuracy Scores for *BLSTM* and *sep-CNN*

The model showing the best accuracy score was the *sepCNN*, using a 1-gram pre-processing of the *tweets* and embedded with an embedding matrix of dimension $N = 200$ using the *GloVe* on the dataset provided for the study.

VIII. Discussion

A. Data-set Analysis and Embeddings

The comparison between the different state of the art methods for word embedding proposed in this paper gave interesting results, we saw that GloVe and Word2Vec provided quite similar output performance-wise, seeming best suited to feed to neural networks. One interesting fact is that despite the enormous number of tweet on which the GloVe embedding dictionary from Stanford was trained, our best results were given on a much smaller GloVe representation, trained solely on the samples provided for this study (roughly 2.5 millions i.e one thousandth of the Stanford training set). We can deduce that the word space generated by Stanford may reduce the distance between some discriminatory words needed for sentiment analysis, whereas in our case, we only work with tweet characterized by a negative or positive emotion.

B. n-Grams

Using bigrams seemed to significantly improve our results during the testing phase when used with a BLSTM. However, due to computational complexity we only managed to use a maximum of 100 features for the embedding with bigrams and the full dataset, indeed the generation of bigrams increases greatly the number of different "word" in the corpus. Since we saw that using a higher number of dimensions for the embedding improved significantly the results, and we managed to have 87.1% with only 100 features on aicrowd, we can speculate that with more computing power and thus using a larger dimension for the embeddings we could obtain better result with Bigram or Trigram.

C. BLSTM and sep-CNN

It appeared quickly that those models were the most adapted to the task at hand. However, the large amount of processing required to obtain results largely limited the depth at which it was practically possible to explore the exploration of their parameters and topologies. Still, the way both model were designed seemed well fitted to model the semantic structure of natural languages, especially with self-contained items of short length like *tweets*.

D. Possible Improvements

The more recent approaches for dealing with NLP are algorithms based on *Transfer learning*. Google, in particular, came out with an innovative model called *Bidirectional Encoder Representations from Transformers* (BERT) in 2018. [11] Indeed, as previous algorithms tended to treat queries as a 'bag of words' with no particular order, the Google `BERT` algorithm looks very closely at the order of words in the query, claiming it helps it offers a better understanding of the meaning behind the query.
Nevertheless, it needs a tremendous amount of computing time, hence in this study, it was only tested with a random selection of 250 *tweets* as a training set. As a result, an impressive 83% accuracy was obtained using 10% of the dataset for testing, which suggests very promising performances if trained of the full dataset.

References

[1] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[2] V. Vapnik and V. Vapnik, "Statistical learning theory wiley," *New York*, pp. 156–160, 1998.

[3] L. B. DURGESH K. SRIVASTAVA, "Data classification using support vector," *Journal of Theoretical and Applied Information Technology*, 2009.

[4] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[5] P. Zhou, W. Shi, J. Tian, Z. Qi, B. Li, H. Hao, and B. Xu, "Attention-based bidirectional long short-term memory networks for relation classification," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2016, pp. 207–212.

[6] M. Wöllmer, A. Metallinou, F. Eyben, B. Schuller, and S. Narayanan, "Context-sensitive multimodal emotion recognition from speech and facial expression using bidirectional lstm modeling," 01 2010, pp. 2362–2365.

[7] A. Mousa, "Architecture of blstm." 2016, [Online; accessed on 19.12.2019]. [Online]. Available: https://www.researchgate.net/figure/Architecture-of-BLSTM_fig1_307889752

[8] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[9] T. I. Denk, "Text classification with separable convolutional neural networks," 2017.

[10] "Google's implementation of a depthwise separable convolutional neural network," https://github.com/google/eng-edu/blob/master/ml/guides/text_classification/build_model.py, accessed: 2019-12-18.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018.