Twitter Sentiment Classification

Zhecho Mitev Kylian Do Nascimento Anthony Yazdani
zhecho.mitev@epfl.ch kylian.donascimento@epfl.ch Anthony.Yazdani@etu.unige.ch

***Abstract*—Understanding the sentiment of a text plays a key role in Natural Language Processing. Such task is usually very trivial for a human; however, algorithms struggle to recognise whether certain sequence of words contains a positive or a negative sentiment. To tackle this problem, neural networks have been successfully introduced to the field of sentiment analysis. To facilitate the deep learning methods researchers have started using Twitter data, because of its large quantities and social value. In our paper, we present the full pipeline of generating an effective sentiment analysis. First, we discuss the importance of data pre-processing and its value to Machine Learning (ML) techniques. Then, we focus on the creation of word vector representations using the GloVe method. Finally, we compare different classification approaches and assess the ones that perform the best in the given setting. Our experiments show that deep learning methods significantly outperform the classical ML methods. In particular, the bi-directional LSTM network outputs an accuracy of more than 85% on both validation and test data.**

I. Introduction

Sentiment analysis is a field of Natural Language Processing (NLP) that aims to extract and quantify sentiments [1]. The extracted sentiment scores can then be used to produce statistics on the general feelings and opinions of a community. Recently, Twitter has become popular among researchers due to the vast amounts of data provided by the social network. In our research we focus on classifying tweets as either positive or negative. However, how can an algorithm recognise the sentiment of a word sequence?

The first step of achieving this goal is to produce a mathematical representation of qualitative data in a vector space. For this purpose we use the concept of word embeddings. We particularly focus on the GloVe method, which combine the properties of global matrix factorization and local context window methods to produce a vector space that accurately represents the corpus [2]. We explore two different GloVe word representations - one pre-trained GloVe model provided by Standford University that has used 2 billion tweets for training [2] and another model that we have trained specifically for our research using 2.2 million tweets.

Machine Learning (ML) and Deep Learning (DL) approaches has proven to be very effective in classifying text data based on a word embeddings model [3]. Therefore, we explore classical ML techniques such as Naive Bayes [4] and Logistic regression to create a baseline model. However, our research focuses on the field of neural networks as these have achieved major success in text classification tasks, recently. We particularly investigate the multilayered neural network (MLP), long short-term memory (LSTM) network and convolutional neural network (CNN).

II. Data processing

In order to take advantage of the large number of observations that have been provided to us, we perform several meaningful data processing steps. Indeed, we had 2.27 million unique training tweets with an equal proportion of tweets labeled as positive and negative. Additionally, the test set contains $10'000$ tweets on which we will be able to evaluate our models. We note that the average length of a tweet is 76.9 words. This means that the tweets are fairly long, which makes the sentiment analysis a difficult task. To improve the quality of the data we propose two data processing approaches.

A. Trivial approach

To reduce the tweets' length, but preserve their core meaning, we perform the following data cleaning methods:

- **Delete labels and tags**. Tags such as <user> and <url> are removed due to their neutral sentiment.
- **Delete non-alphabetic words**. Words that contain symbols which are not present in the English alphabet are removed to decrease the size of our vocabulary.
- **Words to lower case**. Upper case and lower case letters can produce numerous words with the same meaning. Thus, we normalize our corpus. (e.g. “eXtra” to “extra”)
- **White spaces**. Multiple white spaces do not carry any meaning thus we make sure that there is only 1 white space between words ((e.g. ”hey there” to ”hey there”)

These simple steps reduce the dimensionality of the data, thus we believe that such cleaning can be beneficial to our word embedding and machine learning implementations.

B. Advanced approach

In order to increase the correctly classified instances, to the first approach steps, we add several more sophisticated preprocessing methods:

- **Replace emoticons**. We define a list of happy and sad emoticons and if encountered in the text, they are replaced by the words “happy” or “sad”, respectively.
- **Squeeze repeating characters**. (e.g. “laaazy” to “lazy”)
- **Decontract words**. Often tweets contain shortened phrases, thus we define a set of such words and transform them to their full form (e.g. “won’t” to “will not”)
- **Delete stop words**. Remove common words which do not carry any sentiment (e.g. “I”, “it”, “of”, etc.)
- **Lemmatize**. Group together the inflected forms of a word so they are analysed as a single item (e.g. ”go” = ”goes”)

The first and second step are implemented to capture Twitter-specific text and convert it into meaningful information

dataset. Additionally, in [5], the authors demonstrate the value of abbreviations analysis, lemmatization and stop words removal by testing these approaches on 6 different datasets. To confirm of deny the usefulness of these approaches we compare their results in Section V.

III. Word embedding

Among all word embedding methods, the most commonly used are the GloVe method [2], the Word2Vec method [6] and the FastText method [7]. In our project, we opt for the GloVe method as it has shown a slight advantage when it comes to a more general tweet classification task in the article by Li et al. [8].

A. GloVe: Global Vectors model

The main goal of any word embedding model is to translate a text into vectors that accurately represent the meaning of the words. Methods in the family of matrix factorization such as Latent Semantic Analysis suffer from a poor structure of the matrix, while method like Word2Vec are unable to capture essential global information from large text sequences. As we note in Section II, a tweet can be quite long, thus it is important to understand the global idea of the text containing the sentiment. Therefore, we implement the Global vectors model proposed by Pennington et al.(2014) [2]. First, a co-occurrence matrix X is build based on corpus of words, where X_{ij} represents the number of times that word i and word j appear in the same context. The probability that word j occurs given that word i is in the context is annotated as follows:

$$P_{ij} = P(j|i) = \frac{X_{ij}}{\sum_k X_{ik}}$$

The main idea that is proposed in [2] is calculating the ratio $\frac{P_{ik}}{P_{jk}}$, in order to retrieve information about the relevance of the words. In case the ratio is much above 1 then k is much more relevant to i than to j and vice versa, if the ratio is close to 0 then k is more relevant to j.

The next step is to define the word embeddings w_i and w_j based on the following formulation:

$$F((w_i - w_j)^T \cdot \tilde{w_k}) = \frac{P_{ik}}{P_{jk}}$$

where the function F must be a homomorphism between the groups ($\mathbb{R}$,+) and ($\mathbb{R} > 0$, $\times$) [2]. For example, F can be an exponential function.

In order to compute vector space representation, the model is trained only on non-zero co-occurrence matrix entries, instead of iterating through the entire sparse matrix. This allows for efficient computation of all embeddings. Pennington et al. (2014) introduce an updated optimization problem which is specifically suited for the GloVe embeddings. To find the optimal vector representations the following function is minimized

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w_j} + b_i + \tilde{b_j} - logX_{ij})^2,$$

where f(x) should be a non-decreasing function, so that a small weight is allocated to rare co-occurrences. At the same time, frequent co-occurrence should not be overweighted either, thus for large values of x the output should be relatively low. Once the entire vectors representations are computed, a tweet t is represented as a concatenation of word embeddings:

$$t = \{w_1, w_2, ..., w_n\}$$

B. Training data

As it can be deduced from the GloVe method, the word embeddings highly depend on the data that they are trained on. We utilize two types of datasets :

- **Pre-trained embeddings.** 200-dimensional vectors produced using 2 billion tweets [2]. The embeddings are available on the following website: "$https://nlp.stanford.edu/projects/glove/$". The advantage of such vectors is that they have been trained on a very large data set for a long time, thus very good performance can achieved. However, this dataset is very general, therefore it might not correspond well to our dataset, which could have negative effect to the testing score.
- **Embeddings trained on specific data.** In order to capture any particular information from our dataset, we compute the 200-dimensional vector on the 2.27M available tweets. The number of data points is seemingly much smaller than the previous data, however this approach can capture any particular information of the training set, to improve the performance on the test set.

IV. Models

Machine Learning techniques have proven to have high efficiency in classification task based on large amounts of data. Our goal is to find the ML algorithm that can predict the sentiment of our tweet with highest accuracy.

A. Classical Machine Learning algorithms

Using the Python library scikit-learn, we train several common Machine Learning methods. To obtain our one dimensional input vector we take all words in a sentence and compute the average over the given vector representations of the words produced by GloVe. Given this input, each classifier aims to predict the sentiment of all test tweets after learning a function that accurately represents the data.

- **Naive Bayes** is a probabilistic approach that utilizes the Bayes' theorem to classify instances. It makes an assumption of conditional independence between every pair of features. Such assumption might seem oversimplified, however the approach yields surprisingly good results in different practical problems, such as document classification [4]. Another advantage of the method is that it is much faster than other classical ML approaches as it does not suffer from the curse of dimensionality.
- **Logistic regression** is statistical linear model that uses the non-linear logistic function to do binary predictions.

To ensure regularization of the output we apply l2 normalization. The cross entropy loss is used as a loss function. In general, this is a simple, but powerful model for classification tasks.

- **Multi-layered perceptron** (MLP) is a feedforward artificial neural network which connects multiple layers of nodes in a directed graph. In our specific case of binary classification, the algorithm learns a function $f(\cdot) : R^m \rightarrow R^1$, where m is the number of input dimensions. Each node of the MLP uses a non-linear activation function to process and pass the information. Compared to the logistic regression method, which utilizes a single activation function, the structure of the MLP is much more complex as the computes the composition of multiple nested activation functions.

All three algorithms use a stochastic gradient descent (SGD) to calculate the optimum of their loss functions.

B. Recurrent neural network (RNN)

RNNs have become a very popular solution to NLP problems in the recent years because of RNN's adaptability to variable-length text [3], [9]. The main idea of the RNN is to perform recursive iterations on the hidden state vector h based on the input text, in order to encode the full sequence information. Although, the iterations are performed recursively, the method exhibits a time complexity $\mathcal{O}(n)$, where n is the length of the text. To compute hidden state h_t the following function is applied:

$$h_t = \begin{cases} 0 & \text{if } t = 0 \\ f(h_{t-1}, x_t) & \text{otherwise} \end{cases},$$

where f is the activation function, x_t is input at time step t and h_{t-1} is the previous hidden state .

Unfortunately, simple RNNs cannot process longer sequences well, since information from the first time steps is barely retained within the final encoding of the sequence. This phenomenon is called vanishing gradient problem [10]. To solve this problem we implement a more complex network structure.

C. Long short-term memory (LSTM)

LSTMs [11] are neural networks that overcome the problem of vanishing gradients by introducing an memory cell for every step, which is expressed as a cell state vector c_t, making a second connection from each cell to the next one. In every memory cell there are three gates: input gate i_t, forget gate g_t and output gate o_t, which are defined as follows:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$
$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$
$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o),$$

where W_o, W_f, W_i are the weights to be learned by the network and b_o, b_f, b_i are the biases for each gate. To explain the function of the gates in short, the input gate takes care of the information passed from the previous cell. The forget gate controls whether the memory should be reset to 0, while the output gate is responsible for making visible the information for the next cell. The sigmoid activation always produces values in the range (0,1) and allows the model to remain differentiable. Additionally, a cell gate is introduced

$$g_t = \tanh(W_g[h_{t-1}, x_t] + b_g)$$

Because of the tahn activation function, the cell state information can float longer into the network, thus the vanishing gradient problem is prevented. Based on the input, forget and cell gates the next cell state c_t obtains a value :

$$c_t = f_t \circ c_{t-1} + i_t \circ g_t$$

,where $\circ$ is the Hadamard product (element-wise multiplication). Finally, the hidden state value is defined as:

$$h_t = o_t \circ \tanh(c_t)$$

As we implement the Bi-LSTM, the algorithm performs the equations above in the two direction of the text sequence (from left to right and from right to left).

D. Dropout

In order to apply regularization to the neural network we use a dropout technique. It is generally used for dealing with overfitting. To implement the technique, we randomly remove units from a neural network during training at a rate of 0.5.

E. Adam optimizer

Adam is one of the most widely used optimizers for training neural network models. Similarly to SGD, it performs gradient-based optimization, however the learning rate λ is not needed as an input. Generally, it is preferred to SGD, because of its speed of convergence and the little memory requirement [12]. Since our LSTM is very large, we use the Adam optimizer to find the optimal solution. In order to calculate our loss and learn the weights W_o, W_f, W_i, W_g and biases b_o, b_f, b_i, b_g we utilize the Pytorch class BCEWithLogitsLoss, which computes the binary cross entropy loss combined with sigmoid function for each batch of sentences j:

$$l_j = -w_j[y_j \log \sigma(x_j) + (1 - y_j) \log(1 - \sigma(x_j)),$$

This combination of functions makes the loss numerically stable and more robust.

F. Convolutional neural network (CNN)

CNN is a very popular neural network for image recognition, due to its pattern detection strengths. The architecture of the network allows every hidden layer to detect a different type of pattern. CNNs introduce the idea of convolutional filters, which iterate through a sequence and capture the information in a n-gram fashion. Instead of using 2-dimensional filters, which image require we adapt the filters to 1-dimensional, as the filter is moved in a single dimension to process the words. The convolution operation can be formally expressed as

$$x'_i = h(W_c \cdot x_{i:i+h-1} + b_c)$$

,where $h()$ is the hyperbolic tangent function and $x_{i:i+h-1}$ is the concatenation of word vectors from position i to position $i+h1$. An important advantage of CNN is that it can recognise semantics of phrases in a text using a max-pooling layer. We apply a max-over-time pooling operation on the feature map x' computed from the convolution operation.

$$x' = max\{x'_1, x'_2, ..., x'_{n-h+1}\}$$

The idea is that the maximum value has most influence on the output, thus it is the most important n-gram for determining the sentiment of the review. Using the backpropagation feature of the network, the maximum value is passed and the weights of the filters are updated. As we utilize 128 filters in total, the vectors generated by the max pooling operation are concatenated and passed to the dropout layer (see Section IV-D). After half of the units are disregarded, the remaining vectors are passed to the linear layer which returns a prediction for the label of the text.

V. Results

A. Pre-processing

In order to evaluate the effect of our pre-processing techniques we test the accuracy of our Machine Learning models, using the trivial pre-processing (see Section II-A) and the advanced pre-processing (see Section II-B). We use stratified 5-fold cross-validation to obtain the scores.

TABLE I: Accuracy of machine learning methods on sentiment with Glove 2.2M token word embeddings

Accuracy score - Glove 2.2M	NB	LG	MLP
Trivial pre-processing	59.71	**76.25**	**79.53**
Advanced pre-processing	**61.49**	73.69	76.60

TABLE II: Accuracy of machine learning methods on sentiment with Glove 2B token word embeddings

Accuracy score - Glove 2B	NB	LG	MLP
Trivial pre-processing	64.77	**76.66**	**79.92**
Advanced pre-processing	**66.09**	74.37	77.24

Based on the results of the two tables, we note that in general using pre-trained embeddings outperforms our self-generated embeddings. One of the major reasons is the size of the training set. As the pre-trained embeddings utilize 1000 times more data point, it is not surprising that the results are in favour of the 2 billion embeddings.

B. Machine Learning models

Furthermore, note that the logistic regression and the multi-layered perceptron produce better scores than the Naive Bayes method. These scores increase when the trivial pre-processing is applied. Several researchers have been questioning the pre-processing approaches especially when used in combination with deep learning techniques [13], [14]. Similarly to these research paper, we find that advanced pre-processing might be resilient for advanced machine learning method, since they rely on large amounts of data, while the pre-processing reduces the length of the text sequences. Still, we observe that the MLP generates the best result. Since MLP is a deep learning method we choose the trivial set of pre-processing for our tests of Bi-LSTM and CNN.

C. Deep Learning comparison

We have implemented 2 different neural networks which are designed to make a prediction on the sentiment of our test tweets. To compare their performance we run 3 epochs of each neural network[1] and show the result of the best epoch in Table 3. We run the methods using both of the proposed embeddings in our paper.

TABLE III: Deep learning models - accuracy

Accuracy score	LSTM	CNN
GloVe 2.2M	84.81	83.51
GloVe 2B	**85.23**	**84.11**

Apparently, the best results are yield using the Bi-LSTM and the 2B token embeddings rendered by Pennington (2014) [2]. Clearly, the both neural networks perform much better than the classical ML methods thus we can conclude that for this text sentiment analysis task DL algorithms outperform classical ML techniques.
Additionally, on Fig. 1 one can see the scores that have been produced by the AIcrowd platform on all of our methods. The bidirectional LSTM network seems to capture positive and negative features of text best of all.

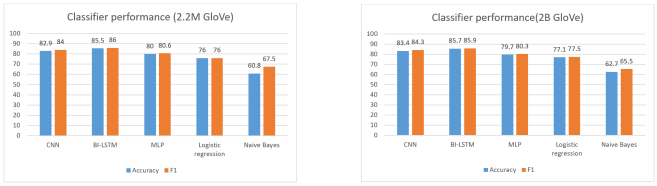

Fig. 1: Performance of all presented models on the test set (using trivial pre-processing)

VI. Conclusion

In this paper we have presented in detail a full pipeline approach to classify the sentiment of Twitter text. The results show that advanced pre-processing such as stop words removal and lemmatization can be harmful to Machine Learning methods and Deep Learning in particular. The reason for such phenomenon can be explained with the complex structure of the neural networks and the ability to capture various features from large amount of data. Additionally, we compare the existing Twitter embeddings trained on 2 billion tokens [2] to specifically produced embeddings based on 2.2 million positive or negative tweets. Even though, the pre-trained GloVe embeddings seem to allow methods to produce better score, we note that our best method - Bi-LSTM outputs very similar scores using both of the word representation options (see Fig.1).

[1]Each epoch takes around 30 minute with a 4GB GPU card

REFERENCES

[1] R. Feldman, "Techniques and applications for sentiment analysis," *Communications of the ACM*, vol. 56, no. 4, pp. 82–89, 2013.

[2] 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 (EMNLP)*, 2014, pp. 1532–1543.

[3] P. Liu, X. Qiu, and X. Huang, "Recurrent neural network for text classification with multi-task learning," *arXiv preprint arXiv:1605.05101*, 2016.

[4] A. McCallum, K. Nigam *et al.*, "A comparison of event models for naive bayes text classification," in *AAAI-98 workshop on learning for text categorization*, vol. 752, no. 1. Citeseer, 1998, pp. 41–48.

[5] F. Khan, S. Bashir, and U. Qamar, "Tom: Twitter opinion mining framework using hybrid classification scheme," *Decision Support Systems*, 01 2014.

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

[7] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[8] H. Li, D. Caragea, X. Li, and C. Caragea, "Comparison of word embeddings and sentence encodings as generalized representations for crisis tweet classification tasks," *en. In: New Zealand*, p. 13, 2018.

[9] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179 – 211, 1990.

[10] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

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

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[13] A. Krouska, C. Troussas, and M. Virvou, "The effect of preprocessing techniques on twitter sentiment analysis," in *2016 7th International Conference on Information, Intelligence, Systems & Applications (IISA)*. IEEE, 2016, pp. 1–5.

[14] K. Choi, G. Fazekas, M. Sandler, and K. Cho, "A comparison of audio signal preprocessing methods for deep neural networks on music tagging," in *2018 26th European Signal Processing Conference (EUSIPCO)*. IEEE, 2018, pp. 1870–1874.