Machine Learning Project II

Ariel Alba, Saleh Gholam Zadeh, Diego Iriarte
Team Name: *SAD*
School of Computer and Communication Sciences, EPFL, Switzerland

***Abstract*—This report summarizes our findings and results about machine learning and deep learning models developed to estimate the likelihood that a given tweet, taken from a real-world dataset, expresses a positive or negative sentiment. The report gives an overview of the machine learning models implemented for making the predictions and argues on the results obtained. It provides a description of the text preprocessing, text representation, cleaning methods and feature engineering applied in order to have better understanding of the tweets information and get their best numerical representation. Based on this and comparing the results obtained, the deep learning model that achieves the best classification accuracy is a Convolutional Neural Network fed by bigram word representation.**

I. Introduction

Natural language processing (NLP) consists of creating systems that process or understand language in order to perform certain tasks. NLP tasks applicability to daily life make it a very exciting field and has a growing interest of big research companies. Applications to NLP tasks have been one of the biggest areas of deep learning research. Deep learning allows algorithms to understand sentence structure and semantics. The model is built as a representation of the entire sentence based on how the words are arranged and interact with each other.

One of the most exiting NLP fields of study is Sentiment Analysis, which can be thought of as the exercise of taking a sentence, paragraph, document, or any piece of natural language, and determining whether that text's emotional tone is positive, negative or neutral. Twitter is a social networking and microblogging service that allows users to post real time messages, called tweets. The challenge arises in the fact that tweets are written by an heterogeneous group of people, which may have different ways to express their feelings by using many variations of words, abbreviations and misspellings.

In that sense, two data sets are provided: a train set and a test set which are a collection of real-world tweets expressing a positive or negative tone. It is important to note that a neutral tone is not considered. The train set is composed of a total of 2.500.000 tweets, properly labeled and divided in positive or negative tone. For training purposes a small train set of 200.000 tweets, half positive and half negative, is provided which is a subset of the complete train set. The test set consists of 10.000 unlabeled tweets and our task is to predict whether they express a positive or negative sentiment by implementing machine learning techniques able to have consistent understanding and interpretation of the tweets and to accurately classify them. Each tweet of the datasets is limited to 140 characters length.

The report summarizes the different preprocessing steps performed for cleaning the tweets, text and word representation options examined as well as some deep learning models for classifying the tweets. The model that gave us the higher score in Kaggle competition [1] is based on a Convolutional Neural Network implemented using the framework Keras in python, fed by a bigram word representation, and outputs an accuracy of 0.8670.

II. Data Preprocessing

This section summarizes all the steps performed for preparing the raw data provided in order to feed our learning models with more consistent and trustful data. Due to the nature of tweets (quick and short messages), people abuse of informal language, use acronyms, make spelling mistakes, use emoticons/emojis and other special characters. Some of the most used terminologies in tweets are: a) emoticons/emojis: facial expressions pictorially represented using punctuation and letters, that may express the user's mood. b) hashtags: a word or phrase preceded by a hash sign (#), used to identify messages on a specific topic. c) target: to refer to other users by using the symbol @. We specially aim to emphasize the words, phrases or symbols that represent any kind of sentiment such as happiness, sadness, anger, or other moods and can help to identify whether the tweet has positive or negative tone.

It is important to note that preprocessing steps were applied differently according to the model and numeric representation chosen, meaning that different combinations of preprocessing were tested while trying to get the best accuracy of the classifier. The most important insights and preprocessing steps applied are detailed in the following lines:

a) **Raw Data:** The provided raw train dataset (positive and negative tweets) has passed through a previous preprocessing consisting of: replacement of target users by the tag <user>, all tweets only contain lower case letters, there is a single space between words in each tweet and there is a <url> tag replacing specific webpages urls (e.g. <user> why is she so perfect <url>).

b) **Punctuation handling:** In most cases punctuation does not express or represent any sentiment, so they were removed from every tweet. There are three special punctuation marks that we considered that needed to be kept which are hash symbol (#), exclamation mark (!) and apostrophe (′). Hash symbol may represent a hashtag which is followed by a message, exclamation may mean an augmented significance of the message and can be useful for sentiment classification, and apostrophe is kept because of word contractions (e.g. "today, i played soccer!.." converted to "today i played soccer!").

c) **Repeated letters:** There are many words that contain repeated letters, it may mean that people wanted to emphasize the word, but they are semantically incorrect. If

[1] https://www.kaggle.com/c/epfml17-text/leaderboard

there are many repetitions of a letter they are reduced to a maximum of two repetitions and if the word is misspelled it can be corrected more easily later (e.g. "loveeeee" converted to "lovee").

d) **Split text and numbers** Another finding corresponds to words which contain numbers mixed within the word. We use a function to split numbers and letters, it is useful because in many cases a word representing a sentiment is trapped between numbers (e.g. "789love99" split to "789 love 99"). We found out that there are also many meaningless words or letters trapped between numbers but other preprocessing steps handle them (e.g. "00xx89" split to "00 xx 99").

e) **Filter numbers:** Numbers don't represent any positive or negative sentiment. One initial approach was to replace numbers by the tag <number> (e.g. "00 hello 99" converted to "<number> hello <number>").

f) **Remove stop words:** It usually refers to the most common words in a language. We used Natural Language Toolkit (nltk) library [2] stopword's list as a reference for words removal in the tweets. Some words expressing negation were removed from the list (e.g. don't, not, nor). Another try was to include in the stopword list the created tags <user>, <number> and <url>, since they appear very often in the tweets and they don't express any sentiment either (e.g. "I want to play soccer" converted to "want play soccer").

g) **Expand contractions:** Contractions are widely used in tweets due to informal abuse of language. One approach was to expand the contractions and try to capture the word that represents a positive or negative sentiment (e.g. "don't" converted to "do not"). In this case we found out many contracted words that are not correctly spelled, mainly missing the apostrophe and some of them were included in our expand list (e.g "didnt" converted to "did not").

h) **Emoji interpretation:** Emojis or emoticons are a pictorial representation of a feeling by the use of punctuation marks. They are widely used in tweets and may be very useful for sentiment classification. We use a function to look for possible combinations of punctuation and letters that may represent an emoticon and replace them by the respective tag such as <smile>, <sad>, <neutral> or <heart> (e.g. "thankful :-) <3" converted to "thankful <smile> <heart>").

i) **Hashtag interpretation:** Hashtags are expressions or sentences expressed in one word without spaces and usually preceded by the hash symbol (#). They may encapsulate a message that denotes a sentiment, so the task is to split the hashtag into meaningful words. The difficulty with hashtags is that there may be two or more different split possibilities from which we should guess what the user wanted to express. The function used is based on nltk words and finds all possible combinations of the hashtag, in all cases we take only the first hashtag split (e.g. "#gohomesill" outputs two possibilities "go home sill" or "go homes ill").

[2]http://www.nltk.org/

j) **Lemmatize and Stemming:** Lemmatize is the algorithmic process of determining the lemma of a word based on its intended meaning. It depends on correctly identifying the intended part of speech and meaning of a word in a sentence, as well as within the larger context surrounding that sentence, such as neighboring sentences or even an entire document. Lemmatize was performed using nltk WordNetLemmatizer (e.g. "children" converted to "child"). Stemming was also explored but provided bad results since it usually cuts the last letters of a word, leaving them meaningless.

k) **Correct misspelling:** Tweets contain an abuse of abbreviations, informal grammar and misspellings. It may happen that an important word for describing the sentiment is misspelled, missing a letter or having a repeated one. We aim to find the correct spelling of such words and replace them. The difficulty is to correctly interpret the misspelled words since it may have different options such as "lates" can be corrected to "late" or "latest" or "lattes", so the function used is based on probabilities to find the correction, out of all possible (e.g. "thy are plying" converted to "they are playing").

l) **Test dataset:** Each tweet of the test set starts with its id number. Since these id numbers have no sentiment representation they are removed. By applying one of the previous methods for removing numbers, we are able to handle with them and depending on the model used all other preprocessing steps are applied similarly as to the train set (e.g "1, tweet number one" converted to "tweet number one").

III. Data Representation

For building a good text classifier, it is crucial to find a good feature representation of the input text. This section describes the algorithms used to convert the tweets into numerical representations. For it, the first step is to build a vocabulary, which represents the link between a word and its numerical representation. Usually the words that have higher occurrence in the whole document are taken as part of the vocabulary. The numeric representation methods explored are:

- **Word Vectors:** This first approach consists of generating a word embedding for each word of a given tweet. The word vector is calculated using a custom embeddings generated by a coocurrence matrix, a vocabulary and Glove model. The first step to get the word embeddings is to generate the matrix of coocurrences, which counts the number of times that two words X and Y appear together. This matrix is generated using the vocabulary and the train dataset. Then Glove's model is applied to coocurrences matrix defining the embedding's dimension and generating an embedding matrix, where each row represents a word embedding. Then, each word is paired with its corresponding numerical representation from the embedding matrix. To get the tweet representation, our simplest approach consisted of getting the feature representation of each training tweet by averaging the word vectors over all words of the tweet. Other alternatives, different from getting the average may be applied to get the tweet representation.

- **TFDIF Representation:** TFDIF or term frequency inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection. This method takes into account the occurrence of a word in the entire corpus. Common words like is, the, a etc. tend to appear quite frequently in comparison to the words which are important to a document. TFIDF works by penalizing these common words by assigning them lower weights while giving importance to other relevant words in a particular document.

- **N-gram Extension:** N-gram is a representation usually used in language models. A n-gram model of order n represents maximum n words in a sequence of contiguous words. In this project, the tweets matrix is extended using a contiguous sequences of 2 words in every tweet (bigram). This representation lets a length of a tweet vector representation grow to 2 * len(tweet) - 1. When using a pretrained word embeddings, the vocabulary is modified in order to achieve the same result as the n-gram extension. Each n-gram in the vocabulary entry is set to have the mean vector of the words used to generate it [1].

- **Pretrained Word Vectors:** Pretrained word vectors can lead to substantial performance gains as long as the model is configured and optimized to take advantage of this initial structure [2] [3] [4]. The most popular pretrained word embeddings are Google's word2vec [3], Facebook's fasttext [4] and Standford's Glove [5]. Standford's Glove pretrained word vectors were used as input to the Neural Networks and Convolutional Neural Networks models described in section IV. Glove combines global matrix decomposition and local context window. Glove's vectors were trained with 2 billion tweets containing 1.2 million words in its vocabulary and a collection of around 27 Billion tokens, each of them represented by a fixed-size vector of 200 features.

IV. Models

Once the word representations are generated, they are fed to a classifier model for its training. In this section, we present the different methods and combinations of word representation and classifiers applied in order to achieve the best accuracy. Some of the methods explored for classification were Logistic Regression, Support Vector Machines (SVM), Naive Bayes, Neural Networks (NN) and Convolutional Neural Networks (CNN). The most promising results are summarized in the following lines:

1) **Model 1: Word vectors - SVM:** This is our baseline model, it uses global vectors for word representations (glove), based on the cooccurrence matrix and the vocabulary of the data set, as explained in section III. For classification, we trained with SVM using scikit learn library [5] and predicted the labels for the test set. For this model, we achieved better results after applying preprocessing to the raw data as described in Section II. For simplicity and to get an idea of the model's performance, we applied a split of train data set into train/validation with a ratio of 0.9/0.1 and the accuracy achieved with this model is 0.59 and Kaggle's competition accuracy is 0.60580.

2) **Model 2: TFIDF - SVM/Naive Bayes:** This model consists of using TFIDF for text representation as described in section III with scikit TfidfVectorizer in python. The number of features defined per word was 1200. For this model we also tried generating the word vectors with preprocessed dataset and achieved better results. We trained the dataset with the linear classifier SVM and Naive Bayes to predict the labels for the test set. As in model 1, the validation method for the model consists of splitting the training set into train/validation with a ratio of 0.9/0.1 and the accuracies obtained with these models are: a) with SVM: validation accuracy 0.76, Kaggle: 0.72920. b) with Naive Bayes: validation accuracy 0.81, Kaggle: 0.6854.

3) **Model 3: Pretrained word embeddings - CNN, Tensorflow:** This model consists of training a CNN with one layer of convolution on top of word vectors obtained from an unsupervised neural language model (pretrained vectors). The first layer embeds words into low dimensional vectors. The next layer performs convolutions over the embedded word vectors using multiple filter sizes (3,4,5). Next, we max-pool the result of the convolutional layer into a long feature vector, add dropout regularization, and classify the result using a softmax layer [6]. For the pretrained vectors, we tried with both Google's word2vec and Standford's glove pretrained vectors described in section III. Pretrained word embeddings contain vector representations for the most used words and symbols on tweets, therefore we only include few preprocessing steps to the datasets (such as adding tags for numbers, emojis and hashtag as described in section II). CNN was implemented using Tensorflow, a highly sophisticated framework in Python developed by Google for Machine Learning [7]. Figure 1 shows the train and validation accuracies of every step obtained using this model. We use a 0.1 sample percentage for model evaluation and the results are: Using pretrained word2vec, validation accuracy: 0.8016 and Kaggle competition accuracy: 0.8206. Using pretrained glove embeddings, validation accuracy: 0.8041 and Kaggle competition accuracy: 0.8226.

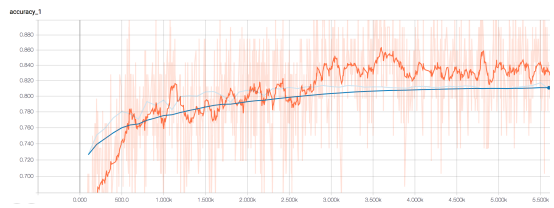

Fig. 1: Train (orange line) and Validation (blue line) accuracies

4) **Model 4: Bigram - NN with pretrained data:** This model uses Neural Networks as the classifier. To model

[3]https://nlp.stanford.edu/projects/glove/
[4]https://fasttext.cc/
[5]https://code.google.com/archive/p/word2vec/

this abstraction there are several python frameworks that have many features and give some advantages about stability, comprehension and other features. For this approach we used Keras [6], a python high level neural network API built over other frameworks as TensorFlow, CNTK and others. NN model is composed by 1 fully connected layer, that has the sigmoid as activation function, a Pooling layer and an embedding layer where it takes the pretrained data and length as input. Standford's Glove pretrained embedding was filtered using the current vocabulary, in order to have only the words that appear in our vocabulary and reduce processing times. While using n-gram extension, pretrained embedding must be modified to have the same length as the vocabulary, in this case vocabulary contains also entries for bigram representations. Therefore, we should fill empty rows with some representation. For this goal we examined two options, the first approach was filling the gaps with a vector of zeros, and the second approach was to to get the numerical representation from the vocabulary of the words that generate the bigram and average the values of those vectors. First approach provided better results. To optimize the training, Adam optimizer was used due to its efficiency while using large datasets and high-dimensional parameters [8]. Once the NN model is trained with the given dataset, the test set is fed to the model to get the predictions of their labels. The model outputs two probability vectors, one belonging to the train dataset and the other to the test set. The vectors contain the probability of each tweet to represent a negative sentiment (labeled -1). These two probability vectors are feed to another classifier called XGBoost [7]. XGBoost is an implementation based on boosted tree algorithms [9] that offers high efficiency in the use of memory, thus in performance. To get the accuracy of the model, we split the train set into train/validation sets and obtained a validation accuracy of 0.8550. The accuracy obtained in Kaggle competition is 0.8530.

5) **Model 5: Bigram - CNN without pretrained data:** For this model we use Convolutional Neural Networks. Unlike NN, CNNs are not fully connected, in this case a neuron is connected just to a few neurons of its previous layer, learning data specific kernels and translate many low level features to compressed high level representation. For this approach, Keras is the framework selected to model CNN. The layers used are: 2 Fully connected layers, one with sigmoid activation function and the other with relu activation function; and 1 Convolutional Layer with 32 filters and a kernel with size 3. As the previous model the optimizer chosen was Adam due to its high efficiency. After the model is trained, it outputs probabilities for each tweet to be negative tweet (labeled -1) for the train and test sets. These probabilities are fed to the XGBoost classifier, which then outputs the predicted labels for the test set. Due to its better computation performance, we were able to feed this model with a larger train set of 2.4 million tweets. For the sake of completeness and to get a more precise validation accuracy, we performed k-fold cross validation with a k=10 in a smaller dataset (due to processing time) and large batch size, which outputs a value of 0.8587 +/- 0.0008. This model gave as the best results in Kaggle competition with an accuracy of 0.8670.

6) **Model 6: Combination of CNN and NN models:** This model consists of the combination of the previous two. Since previous models output two probability vectors (train and test), each containing the probabilities of the tweets to express a negative sentiment (labeled -1). we gather both set of probabilities from previous models and feed the classifier XGBoost with both set of probabilities. This model outputs our second best accuracy in Kaggle competition of 0.85999.

[6]https://keras.io/

[7]https://github.com/dmlc/xgboost

V. Results and Summary

The results obtained with the six models are summarized in table I. The model that achieved the best accuracy is model 5 implemented using bigram word representations and Convolutional Neural Networks. CNN models have shown to be effective for text classification and sentiment analysis. CNN requires little hyperparameter tuning and while being simple they outperform remarkably well. CNN's run time showed up to be a good advantage compared to other methods, which allowed us to use a larger train dataset for model 5.

For models 3 and 4 it is shown that unsupervised pretraining of word vectors allows us to get good word representations and improves classification performance. Since glove pretrained word vectors were trained over a big set of tweets, its dictionary contains most common used words, signs, expressions, abbreviations in tweets. We implemented and tested many preprocessing steps and some were successfully used only for the first models which provided better accuracies.

The large amount of data and the complexity of the models required very long training times. The best machine learning model evaluation is k-fold cross validation. However, due to its computational expense it is often not used for evaluating deep learning models. As such, for the model evaluation in most models we used a simple separation of data (data split) into training and validation datasets. For the sake of completeness and to get a robust estimate of the performance of our best model 5 on unseen data, we implemented k-fold cross validation with k=10, over a large batch size and only in 1 epoch and provided a precise estimated accuracy compared to the one obtained in Kaggle.

TABLE I: Accuracy for each model

No	Model	Accuracy	Comments
1	Word Vectors/SVM	0.6058	Trainset: 200.000 tweets
2	TFIDF/SVM	0.7292	Trainset: 200.000 tweets
3	Pretrained/CNN tensorflow	0.8226	Trainset: 200.000 tweets, glove word embeddings
4	Bigram/NN with pretrained	0.8530	Trainset: 600.000 tweets, glove word embeddings
5	Bigram/CNN w/o pretrained	0.8670	Trainset: 2.400.000 tweets
6	Combined CNN/NN	0.8599	Trainset: 1.200.000 tweets Combines models 4 and 5

REFERENCES

[1] C. T. Amir Eftekhar, Walid Juffali. (2014) Ngram-derived pattern recognition for the detection and prediction of epileptic seizures. [Online]. Available: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0096235

[2] G. C. J. D. Tomas Mikolov, Kai Chen, "Efficient estimation of word representations in vector space," 2013. [Online]. Available: https://arxiv.org/abs/1301.3781

[3] W. C. Hanxiao Liu, Ruslan Salakhutdinow, "A comparative study of word embeddings for reading comprehension," 2017. [Online]. Available: https://arxiv.org/abs/1301.3781

[4] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[5] S. learn developers. (2014) Support vector machines. [Online]. Available: http://scikit-learn.org/stable/modules/svm.html

[6] Y. Kim, "Convolutional neural networks for sentence classification," 2014. [Online]. Available: https://arxiv.org/pdf/1408.5882.pdf

[7] T. F. TM. (2017) Convolutional neural networks tensor flow. [Online]. Available: https://www.tensorflow.org/tutorials/deep_cnn

[8] J. L. B. Diederik P. Kingma, "Adam: A method for stochastic optimization," 2015. [Online]. Available: https://arxiv.org/pdf/1412.6980.pdf

[9] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," 2001. [Online]. Available: https://statweb.stanford.edu/~jhf/ftp/trebst.pdf