Tweets Sentiment Classification

Elisabetta Fedele, Lorenzo Liso, Davide Maioli, Ravi Srinivasan
Department of Computer Science, ETH Zurich, Switzerland

***Abstract*—Thanks to the advancements in technology there has been a huge growth of the volume of data available on the web. Specifically, web engines and social media have made it possible to collect labeled and unlabeled data on unprecedented scale. In particular, Twitter is one of the biggest micro-blogging platforms which allow users to share thoughts as "Tweets". In this paper we focus on sentiment analysis of Tweets extracted from Twitter and labeled in an automatized (and noisy) manner. We show that despite the noise in the labels it is possible to classify the sentiment of Tweets with high accuracy by using simple and complex models.**

I. Introduction

The rise of social media platforms, used daily by millions of people all around the world, created a constant stream of data which offers data analysis opportunities as never seen before: more then 800 million tweets are posted on Twitter every day. Online social interactions offer companies a large volume of sentiment rich data in different forms. It also provides businesses with a platform to connect with the customers and especially to understand the customer needs for better advertising. Due to the large volumes, there is a growing need of techniques to automatise data analysis.

Sentiment analysis can be defined as the process of analyzing opinions, views, and emotions from text [1] and it can help businesses to understand whether a product is satisfactory, and to predict future trends. This process involves the classification of opinions into categories such as "positive" or "negative" and, to achieve this, different Machine Learning and Deep Learning techniques can be used. Classification tasks usually require data to be labeled, which also represents another challenge: the large amount of data makes it difficult to analyze and label a large number of tweets. The dataset on which we perform classification was automatically labeled using distant supervision, as explained in detail in Section III.

In Section II we discuss the previous work that has been done for this tasks. Section III provides a detailed description of the dataset as well as the preprocessing techniques that we used. In Section IV we discuss the various models that we used which include ML techniques such as Support Vector Machines as well as DL models such as Transformers. Section V details the experimental results that we obtained, which show that accurate classification can be achieved despite noisy labeling of the data.

II. Related work

Sentiment analysis is a classical problem in natural language processing and as such it has been studied extensively [2], [3].

Traditional machine learning approaches have been based on manually extracting features from the text, in order to obtain suitable representations. These features were then fed as input to a classification model, which gave as output the sentiment (positive or negative). A few examples of this approach can be found in [4].

In recent years, the rise of deep learning has given birth to another technique for text representation, which is neural embeddings. It consists of automatically learning a mapping between text (words, documents) and vectors using a neural network. The classification task is also usually done with deep learning models. Word2Vec [5] was one famous technique for learning word embeddings that introduced negative sampling as a computationally cheaper way to do it. A variety of different architectures have been used in sentiment analysis, for example convolutional [6] and recurrent [7] neural networks.

The current state of the art in a lot of NLP-related tasks, including sentiment analysis, is achieved by models implementing the Transformer architecture [8], which uses an attention mechanism to learn relationships between different parts of the text. These models are usually pre-trained on a huge amount of data and then fine-tuned for the specific task at hand. [9].

III. Data

The training data provided in this project was divided into:

- a small training dataset containing 200000 tweets
- a complete training dataset containing 2500000 tweets

All these tweets were labeled using distant supervision so that every tweet which contained a :)ẃas labeled as positive and, conversely, every tweet containing :(ẃas labeled as negative.

This labeling approach was introduced by [10] and it is known to be a noisy labeling with respect to the sentiment: text (tweets in particular) may contain mixed sentiment, sarcasm, and spelling mistakes.

As a consequence, the first step of our work was to analyze the dataset in order to identify the possible sources of noise and try to remove them.

In particular, inspired by the literature [11] and by our experiments, we present three steps of preprocessing, each applied on top of the previous one:

1) Deletion of tweets containing both positive and negative emotions: these tweets were removed because we did not wish positive features marked as part of a negative tweet, and vice versa.

2) Deletion of duplicate tweets: we removed duplicated tweets in the following way: if a tweet was present in both positive and negative dataset, we detected in which of the two datasets it was occuring more frequently and we have kept that occurrence. Otherwise, (in case it was present only in one of the two datasets), we simply removed the duplicates.

3) Segmentation of hashtags: as the 11% of the tweets provided in the training set was containing hashtags, we decided to exploit their meaning by segmenting them. In order to do that, we have used the library WordSegment, which segmented each hashtag extrapolating all the different words contained in it. For example, '#iloveyou' is processed as 'i love you'.

4) Deletion of repeated letters: we have considered the tweets containing more than 3 repeated consecutive letters as all belonging to the same semantic class, by truncating the successive consecutive occurrences of the character after the third one. For example, the tweet 'hellooooo how are uuuuu?' after our segmentation becomes 'hellooo how are uuu?'. As all such similar tweets will carry the same semantic meaning, this processing will be helpful to exploit their associated sentiment.

In contrast to some literature approaches as [11], [12] we decided not to implement the correction of the misspelled tweets. This choice was made in order to avoid losing the expressions of slang which often characterize tweets. As a matter of fact, trying to use some state of art methods for spelling correction as TextBlob, SymSpell and Bk-Tree incorrectly transformed many tweets containing colloquial expressions.

IV. Models and methods

A. Baselines

1) Word2Vec: Our first approach was to apply classic machine learning algorithms to classify the embeddings produced by the Word2Vec model. First we trained Word2vec with Continuos Bag of Words (CBOW) [5], we choose CBOW as it trains faster than Skip-Gram, and can better represent more frequent words. After having trained the model, for each tweet we average the vectors embedding of each word in the tweet, obtaining a vector embedding representing the single tweet. Finally, we apply a standardization of this "tweet embeddings". To classify the embeddings we use a voting classifier taking the argmax of the sum of the predicted probabilities. The voting classifier is composed of four well know algorithms: RandomForest, LogisticRegressor, Extreme Gradient boosting and Extreme Gradient boosting with Random Forest.

2) DV-ngrams-cosine: The Document Vector by predicting n-grams [13] (in short DV-ngram) is a method for computing semantic embeddings of documents. It uses a neural network to train documents and words embeddings simultaneusly. The rationale behind the method is that the document vector should be informative for predicting which words and n-grams belong to it or not. Moreover, DV-ngram uses negative sampling for more efficient training. It has been shown in [14] that performance could be improved by substituting the dot product with the cosine similarity in the loss function. On top of it, [15] argues that by applying Naive Bayes sub-sampling, the model improves again. In our experiments, we used the last version of DV-ngrams (called DV-ngrams-cosine because of the usage of cosine similarity) by [15].

3) TFIDF + SVM: TF-IDF (term frequency-inverse document frequency) is a method that allows to determine the importance of a word to a document in a collection of documents. TF-IDF requires the entries of the document (tweets in this case) to be represented as list of words.

As TF-IDF creates a matrix with size $(num.tweets, vocab.size)$, we focus on limiting the size of the vocabulary: we apply further pre-processing to the already pre processed dataset, in particular we (i) remove the stopwords, (ii) stem the words using PorterStemmer from nltk, (iii) lemmatize the words using WordNetLemmatizer from nltk, (iv) remove the digits in the tweets and (v) create 2-grams of the words. To further limit the size of the vocabulary and to improve the performance of the classification we decided to exclude from the vocabulary the words that appeared in more than 50% of the sentences (generic words) and in less than 100 sentences (words that are too specific). With this pre-processing we obtain a vocabulary size of 20854. The columns of the obtained matrix represent the features that are used by the SVM classifier.

To perform the classification using the previously generated matrix we opted for a linear support vector machine (LinearSVC from sklearn). To deal with the sparsity of the feature vectors, we opted for L1 regularization, which helps with selecting the important features in the training phase: Fig. 1 shows the absolute values of the feature weights.

4) LSTM: Before the advent of the Transformer [8] based methods, Long Short Term Memory [16] Neural Networks were one of the most used architectures for natural language processing tasks such as machine translation, language modeling, and text classification.

For this reason, we have decided to take a LSTM among the baselines of our project and to investigate how different types of combination of this simple architecture may lead to improvements.

In particular, we have tried three different types of LSTM:

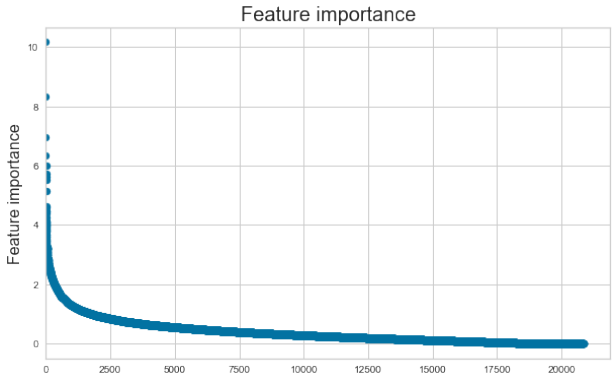

Figure 1. Absolute values of feature weights

1) Vanilla LSTM: it consists on the following layers: Embedding, LSTM, Dropout, FC, which outputs the probabilities of belonging to the positive or to the negative class
2) Bidirectional LSTM: here we have substitued the LSTM layer of the previous architecture with a BidirectionalLSTM, which allows to capture the dependencies from both previous and subsequent words
3) Stacked Bidirectional LSTM: since a common approach in NLP is to stack multiple recurrent layers to improve performance [17], we decided to stack 2 Bidirectional LSTM and to evaluate also the performance of this architecture

5) FastText: FastText [18] is a command-line tool developed by Facebook AI Research. It is able to compute word embeddings exploiting sub-word information. It does it by computing embeddings of character n-grams and representing a word vector as the sum of its character n-grams vectors. Other than computing word embeddings, FastText includes also the implementation of Bag-of-tricks [19], a classification model similar to Bag-of-words but considering also n-grams and with a very efficient implementation. We used FastText both to compute and classify the embeddings.

6) BERT: In the last years Transformer-based techniques have dominated Natural Language Processing as this architectures seems to well capture long-term and short-term dependencies in sequences [8]. In particular we focused our attention to the BERT architecture [20]. Our approach consisted on fine-tuning an already trained BERT architecture on our task, adding a fully connected layer with dropout to compute the class probabilities. First, we tried with BERT small, a reduced version of BERT. Then, encouraged by the promising results, we increased the size of the model. We tried three different "large" architectures : BERT base [21], BERT with talking heads [22], that use a modified version of the multi-head attention mechanism that showed better results [23] and RoBERTa which propose a different training for BERT [24], [25]. BERT with talking heads and RoBERTa showed an accuracy improvement over BERT base, but the increased computational complexity and training time of the new attention mechanism lead us to focus our attention to RoBERTa.

B. BERTweet

Given the promising results of RoBERTa we looked into BERT models pretrained on datasets containing tweets and on task similar to sentiment analysis. BERTweet [26] uses the same architecture of BERT base, but with the improved pre-training of RoBERTa on three different tweet NLP tasks. To improve our predictions we trained BERTweet on two different datasets, one is the original proposed and the other is the proposed one but with hashtag segmentation and without duplicates. The predictions of the two models are then averaged and the class is given by the argmax of the two classes logits.

V. Experiments

We splitted our dataset in two parts: the training set and the validation set. We ran our experiments on different platforms: Google Colab, the Euler cluster or a commodity laptop depending on the training complexity and necessity of GPUs.

All our results are summarized in Table I. Unless stated otherwise, the scores reported are achieved on the Kaggle public test set.

Excluding the dv-ngrams-cosine model, which performed significantly worse than all the others, we can identify two groups of models. The ones not based on the Transformer architecture, even when being quite different one from the other, all perform between 80% and 83.5% of accuracy. Models based on BERT, meanwhile, go roughly from 88.5% to 92%.

All the BERT variants perform better than the original. It is especially evident how BERTweet can exploit having been pre-trained on a dataset similar to the one we had, in contrast to the other variants which have been pre-trained on general text corpora. The best score is achieved by the ensemble of 2 BERTweet trained on different datasets, as detailed in Section IV.

VI. Discussion

We performed experiments on a range of different models, with interesting results.

Using the preprocessing explained in Section III we tried several approaches and we have drawn the following conclusions.

First of all, we can see that DV-ngrams-cosine is performing poorly. This can be partially explained by the dataset we used. DV-ngrams-cosine has been tested [13], [14], [15] on the IMDB dataset, which contains movie reviews. Movie reviews are long and well written documents, very different from the short and sometimes incomprehensible tweets from

Model	Evaluation Acc. %
Word2Vec + classifiers ensemble *	81.92
DV-ngrams-cosine + logistic regression *	51.65
TFIDF + SVM	80.64
Vanilla LSTM	83.32
Bidirectional LSTM	83.52
Stacked bidirectional LSTM	79.88
FastText	83.58
BERT base	88.72
BERT with talking heads	89.74
RoBERTa*	90.07
BERTweet	91.58
Ensemble of 2 BERTweet	**92.04**

Table I
RESULTS FROM OUR EXPERIMENTS. ROWS MARKED WITH AN ASTERISK REPORT THE SCORE ON THE VALIDATION SET.

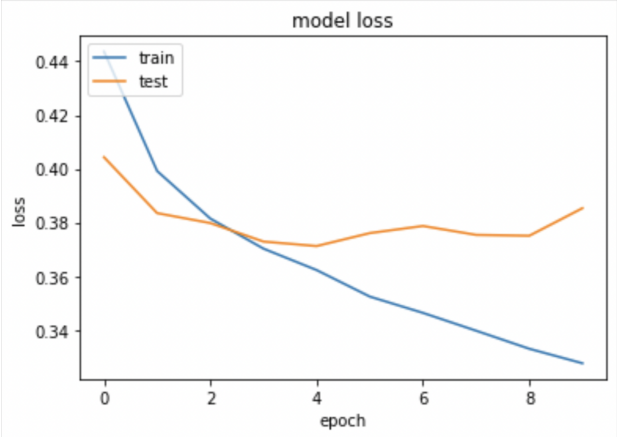

Figure 2. Stacked LSTM's loss during training experiments

our data. Thus, it is possible that, missing clear sentences whose semantics can be captured, the method fails. In future work, it could be interesting to test it against other datasets to assess this hypothesis.

We found that TFIDF + SVM and FastText both achieved good results. This is remarkable, given that both models are simple and train really fast compared to the others. The tweets in the dataset were short and often unclear, so this makes these simpler techniques relatively more effective compared to trying to uncover the meaning of the sentence. Moreover, FastText creates embeddings using subword information, resulting in representations more robust to the typos present in lots of tweets.

Among the three types of LSTM we tried, the vanilla LSTM and the Bidirectional LSTM achieved similar scores, while the stacked LSTM performed drastically worse. This may be due to the fact that our dataset is of small size, compared to the most popular ones, and thus our model is likely to overfit. This tendency to overfitting has been noticed also by observing the training figures (Fig. 2), in which training accuracy always overtakes the evaluation's one after three steps of training. Regarding the BERT-based models, we found a confirmation that they achieve state-of-the-art results in tweets sentiment classification, especially when pre-trained on similar data. We also discovered that two different BertTweets trained on datasets with and without hashtag segmentation and duplicates are different enough to make their ensemble improve the score of the original BERTweet alone. This is interesting because by segmenting the hashtags, we generated additional information which would not have been captured otherwise without the cost of finding additional data.

VII. SUMMARY

In this work we showed how different models can learn to classify the sentiment of a noisily-labeled dataset of tweets. We compared the scores and found that the Transformer architecture confirms itself as the leading method for this type of task, outperforming all the other baselines. Finally, we proposed an ensemble based on augmented data and showed that it improves the top score achieved by a single BERTweet model.

REFERENCES

[1] O. Y. Adwan, M. Al-Tawil, A. Huneiti, R. Shahin, A. Abu Zayed, and R. Al-Dibsi, "Twitter sentiment analysis approaches: A survey," *International Journal of Emerging Technologies in Learning (iJET)*, vol. 15, no. 15, p. 79, 2020.

[2] A. Yadav and D. K. Vishwakarma, "Sentiment analysis using deep learning architectures: a review," *Artificial Intelligence Review*, vol. 53, no. 6, pp. 4335–4385, 2020.

[3] N. C. Dang, M. N. Moreno-García, and F. De la Prieta, "Sentiment analysis based on deep learning: A comparative study," *Electronics*, vol. 9, no. 3, p. 483, 2020.

[4] W. Medhat, A. Hassan, and H. Korashy, "Sentiment analysis algorithms and applications: A survey," *Ain Shams engineering journal*, vol. 5, no. 4, pp. 1093–1113, 2014.

[5] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," no. arXiv:1301.3781. [Online]. Available: http://arxiv.org/abs/1301.3781

[6] Y. Kim, "Convolutional neural networks for sentence classification," 2014.

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

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need." [Online]. Available: http://arxiv.org/abs/1706.03762

[9] C. Sun, X. Qiu, Y. Xu, and X. Huang, "How to fine-tune bert for text classification?" in *China national conference on Chinese computational linguistics*. Springer, 2019, pp. 194–206.

[10] J. Read, "Using emoticons to reduce dependency in machine learning techniques for sentiment classification," in *Proceedings of the ACL Student Research Workshop*. Ann Arbor, Michigan: Association for Computational Linguistics, Jun. 2005, pp. 43–48. [Online]. Available: https://aclanthology.org/P05-2008

[11] A. Go, R. Bhayani, and L. Huang, "Twitter sentiment classification using distant supervision," *Processing*, vol. 150, 01 2009.

[12] U. Naseem, I. Razzak, K. Musial, and M. Imran, "Transformer based deep intelligent contextual embedding for twitter sentiment analysis," *Future Generation Computer Systems*, vol. 113, pp. 58–69, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X2030306X

[13] B. Li, T. Liu, X. Du, D. Zhang, and Z. Zhao, "Learning document embeddings by predicting n-grams for sentiment classification of long movie reviews," *arXiv preprint arXiv:1512.08183*, 2015.

[14] T. Thongtan and T. Phienthrakul, "Sentiment classification using document embeddings trained with cosine similarity," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, 2019, pp. 407–414.

[15] Z. Bingyu and N. Arefyev, "The document vectors using cosine similarity revisited," *arXiv preprint arXiv:2205.13357*, 2022.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, nov 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[17] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," 12 2013.

[18] 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.

[19] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *arXiv preprint arXiv:1607.01759*, 2016.

[20] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," no. arXiv:1810.04805. [Online]. Available: http://arxiv.org/abs/1810.04805

[21] TensorFlow hub. [Online]. Available: https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/4

[22] TensorFlow hub. [Online]. Available: https://tfhub.dev/tensorflow/talkheads_ggelu_bert_en_base/2

[23] N. Shazeer, Z. Lan, Y. Cheng, N. Ding, and L. Hou, "Talking-heads attention." [Online]. Available: http://arxiv.org/abs/2003.02436

[24] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach." [Online]. Available: http://arxiv.org/abs/1907.11692

[25] RoBERTa. [Online]. Available: https://huggingface.co/docs/transformers/model_doc/roberta

[26] D. Q. Nguyen, T. Vu, and A. Tuan Nguyen, "BERTweet: A pre-trained language model for english tweets," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, pp. 9–14. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.2