Stochastic Accuracy Ascent in the text classification competition

Ciprian Baetu, Doru Musuroi, Joseph Vavala

***Abstract*—An important challenge in text classification tasks is obtaining the proper embeddings on which the *usual suspects* can be trained. We present in this report our work with three types of text embeddings - GloVe, FastText and Sent2vec. We used the mentioned methods to create vector representations for tweets, and afterwards trained, as a baseline, standard Machine Learning algorithms like Logistic Regression and SVM with linear kernel, on the representations. Finally, we came up to our best-performing model based on a meta ensembling pipeline composed of neural networks as base predictors on top of embeddings and a linear model as the stacking model.**

I. Introduction

Working with textual information in Machine Learning has been a challenge for a long time. Processing text is very different from the other types of information, e.g. images, because the "atom" should be represented by words, but also because there are almost no rules in how those words can be connected in a proper and logical sentence. Also, the usual encoding for images is very natural, such that if you change a single bit in the pixel-matrix representation, we still have a valid encoding, while changing a bit in textual information represented, for the sake of argument, in ASCII code, the whole text might become invalid.

Therefore, scientists tried to find a way to represent textual information such that we can apply usual Machine Learning techniques on top of it. A breakthrough has been realized by the creators of word2vec [1], which used matrix factorization opposed to neural networks as a main background algorithm for training efficient text representations. After their idea, multiple algorithms based on the same technique were developed, improving the performance of the initial algorithm.

The project which we chose to approach was the *Text Sentiment Classification* on Twitter data. The main task of the project is to come up with a solution which predicts if a tweet message contained a positive smile, denoted by *:)*, or a negative smile, marked in text by *:(*. Even though the name suggests that *sentiment classification* should be performed on tweets, the truth is somewhat different: a user might insert a happy face in an ironical manner, even though the sentiment is negative, or there could be just a typo in the tweet, or maybe, as we actually seen in the data, the user starts an enumeration with a colon sign *:* and then opens a bracket for a short explanation.

Our approach was a gradual one: the first methods tried are very simple, where we just tuned-in pre-trained existing models on our data and we tried to assess the accuracy using those models. Afterwards, we realized fine-tuning on each method, and finally came up with a solution which stacks multiple classifiers, using predictions from different methods.

II. Sent2vec baseline

III. FastText grid-search

In the same time, we tried was to use FastText [2], a method developed at Facebook for efficient text classification. As it was mentioned during the course, the method is among the best ones that exist nowadays, therefore that is the reason for trying it for the first time.

A. Motivation

As we already mentioned, word2vec represented a breakthrough in the area, allowing users to construct vector representations, called *embeddings* for words existing in a given text. Interestingly enough, the resulted vectors can also carry some semantics, allowing sum or difference operations over embeddings, with meaningful interpretations. Even though the results are even shocking at a first sight, we can see there is a small flaw: after training the algorithm on a dataset, we cannot retrieve representations for a word which was not previously encountered. Knowing that users tweets are usually not very correctly written, with many typos, slangs and jargons present, we believe that being able to construct representations for words which were not encountered in the training set would be a big plus in this context.

That is where fastText comes in handy, because it does exactly what word2vec couldn't handle. More specifically, it also has the option to construct the word representation for "*out-of-bag*" words, by constructing representations for different part of words too. Therefore, for a new, unseen word, a representation could be represented by an averaged some of the representations for the "sub-words" [3], i.e. small parts of the word.

B. How does it work?

But how is fastText working under the hood? It is very interesting to see that the method used is extremely easy, relying again on matrix factorization [4], as word2vec does.

Therefore, given $\mathcal{V}$ the set of all words used for training, and also a tweet written as:

$$tweet_n = (w_1, w_2, ..., w_m),$$

where $w_1, w_2, ..., w_m$ are the words of the tweet, we can construct the bag-of-words representation of the tweet as a vector $x_n \in \mathbb{R}^{|\mathcal{V}|}$. Using this representation, our goal is to find the matrices $W \in \mathbb{R}^{1\times K}$ and $Z \in \mathbb{R}^{|\mathcal{V}|\times K}$ that minimize the negative log-likelihood over the output classes:

$$\mathcal{L}(W, Z) = -\frac{1}{N}\sum_{n=1}^{N} y_n log(f(WZ^\top x_n)),$$

where:

- N is the number of tweets in the dataset
- $x_n \in \mathbb{R}^{|\mathcal{V}|}$ is the bag-of-word representation of tweet n
- $y_n \in \{-1, 1\}$ is the label of tweet n
- f is the softmax function

C. fastText on our dataset

The results of fastText on our problem are very promising. As we started with an accuracy of around 80%, we decided to try some parameter tuning in the model. We performed a grid-search on different parameters of the model, as mentioned below:

- *dimension of the embeddings*: values between 10, 30, 50 and 100
- *number of epochs*: values between 5, 10, 15, 20, 50
- *minimum number of word occurrences to be counted*: 1, 5, 10
- *"sub-word" size*: (1, 6), (1, 7), (2, 6), (2, 7), where a pair (x, y) means that we considered all subwords of length between x and y.
- *n-gram size*: all between 1-7. A n-gram means a pair of n consecutive words, which is actually needed to establish the context for words.

With the results from the grid-search, we decided that the best parameters are: *dimension of the embeddings*: 30, *number of epochs*: 12 (we fine-tuned between 10 and 15), *minimum occurrences*; 1, *subword size*: (2, 7) and *n-gram size*: 6.

IV. Stacking model

V. Results

VI. Summary

References

[1] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

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

[3] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *arXiv preprint arXiv:1607.04606*, 2016.

[4] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *In NIPS*. MIT Press, 2000, pp. 556–562.