Machine Learning Project 2: Sentiment Analysis Using *Twitter* Data

EMILE BOURBAN, TIMOTHÉE BRONNER, AND FLORIAN DELBERGHE

Compiled December 20, 2018

Sentiment analysis is a new and growing field of machine learning; it is an implementation of natural language processing that can determine the feelings conveyed in a text. In this project, we aimed at predicting through machine learning methods the sentiment, positive or negative, in tweets. This was part of a crowdAI contest on a dataset of 2.5 million tweets labeled for their attached smiley. Here, we implemented different classification algorithms and ways of processing the data to reach over 86 % accuracy at the end.

1. INTRODUCTION

The aim of this project was to build a machine learning algorithm that could accurately predict the sentiment conveyed in a tweet. This analysis is build on a set of 2.5 million tweets scrapped from the web. Each of them is classified into a category depending on the smiley it contains ':)' or ':('. There were no restrictions this time in the implementations of algorithms so we had the ability to explore many different ways of solving the problem. Thus we started by implementing a simple logistic regression so that we could explore all the preprocessing possibilities. Once this was done, we decided to use neural networks to improve the model even more. Figure 1 shows the process of our algorithm.

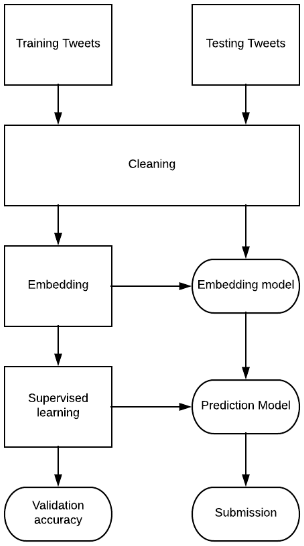

Fig. 1. General pipeline of the learning algorithm

2. OPTIMIZATION OF THE PREPROCESSING

One of the important parts of natural language processing (NLP) is to treat the data as best we can [1]. The aim of this first part is to remove as much as possible of all the useless words that will bring noise to the data without inferring much information, this data can then be more easily processed and its meaning will be more obvious to our different models. The first step is to clean all the '<user>' and '<url>' flags that would be agnostic to the sentiment associated to the tweets. There also seems to be picture frames included in the data in the form of DDxDD (where D is a digit), these were cleaned at first but we then tried to convert them to <img> flags which led to an increased accuracy so we kept it.

The next step should have been to delete all the non characters (punctuation, hashtags, numbers and such), this however showed decreased accuracy so we decided to only remove only the '#', since the word can still be meaningful in this context. We then chose to separate all the verb contraction ('ll, 've, '...).

Other good practices for NLP suggest to remove small stop words (less than 3 letters for example) and to do stemming on the data. This uses the fact that small words don't infer much sense to a sentence ('I want a dog'/'You want the dog' -> 'want dog'). The stemming process is used to remove the plurals and set the time concordance of the verbs to the present. This is another step that keeps the meaning of the tweets while simplifying its construction. In our case removing the word with less than 3 letters did not improve the accuracy and the stemming process made us lose some points as well.

The final preprocess removes verb contractions, letter repetitions and checks to see if the words are in one of the common misspelled dictionaries and corrects them. (All the tests were performed with the pre-trained embedding matrix, "glove.twitter.27B" built by Standford on a large amount of tweets as training data [2] then fitted with a logistic regression).

3. EFFECTS OF EMBEDDING DIMENSIONS

We were provided with a working implementation of the GloVe embedding for our dataset. After having tried it with 25 embedding dimensions, we decided to try and compare it with a pretrained GloVe [2] embedding matrix (with 25, 50, 100 and 200 dimensions). Since these matrices were trained on 2 billion tweets, we postulated that it would give us better results and would not require us to compute it ourselves. As can be seen in Fig. 2, the provided GloVe solution has a very poor accuracy. Its computation time was also much longer. Then, when using the pretrained GloVe for the same number of embedding dimensions the accuracy is over 10 points higher. Finally, when using higher dimensions (up to 200) the accuracy improves even further reaching up to 0.784. For this reason, we chose to use the pretrained version only for the rest of our code. One other way to create embeddings is with the Word2Vec algorithm [3].

4. WORD2VEC EMBEDDING

We also wanted to try a different approach for the embedding. This time we used the Word2Vec algorithm [3] and trained it on our dataset. We obtained an accuracy of 0.707 when working in 25 dimension and up to 0.783 at 200 dimensions. Computation time was also a lot smaller than with the provided GloVe solution. However as Fig 2 shows, the pretrained Glove is still better for all but the smallest tested dimension; even when the Word2Vec was trained on the large dataset. Using the pretrained glove solution with 200 dimensions thus seems the best option.

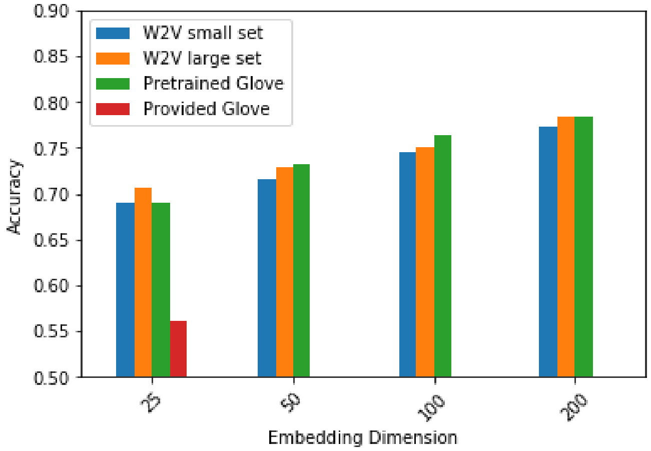

Fig. 2. Validation accuracy in function of the embedding dimensions. Accuracy obtained with a 5 fold cross validation

5. SUPPORT VECTOR MACHINE

An alternative algorithm to the logistic regression for binary classification is support vector machine. When we tested it on a small part of 1/10th of the large dataset with Word2Vec embedding. We obtained the accuracy of 0.673 with SVM, and an accuracy of 0.729 with logistic regression under the same conditions (Embedding dimension of 50). We thus decided to use only logistic regression for further analysis. Furthermore, the SVM has a computing cost that made it impossible to scale up and use on the full dataset.

6. N-GRAMS

N-grams are often used in text analysis to try to capture the meaning from groups of words. We implemented them by attributing unique tokens to each n-gram that we then used as new words for the embedding thereby augmenting the number of features. We implemented bi-grams and tri-grams and then compared their performances with the standard tweets with no-grams (or 1-gram) on both a neural network model and on a logistic regression. As seen in Fig. 3, the accuracy calculated with logistic regression on Word2Vec embedding doesn't significantly change with different n-grams. For the neural network (model 6) however, there is an increase of about 2% in accuracy between no n-grams and bi-grams / tri-grams. We also see that for this case, bi-grams and tri-grams have very close accuracy so we chose to only use bi-grams for our neural networks later on since using them makes for faster computations.

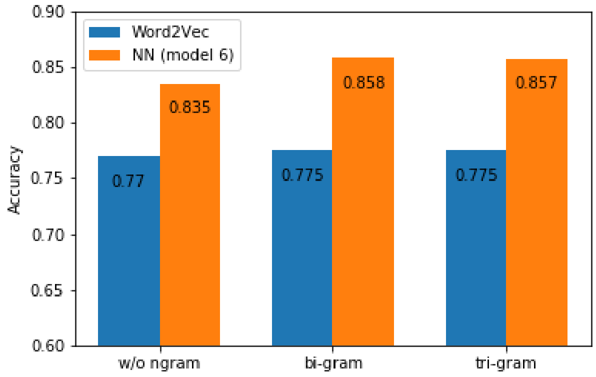

Fig. 3. Validation accuracy for different n-grams calculated either with Word2Vec and logistic regression or with neural network model 6 on the full dataset. Embedding dimension of 200.

7. NEURAL NETS

For the next phase of our classification system, we chose to implement neural networks. They are some if not the most powerful algorithms used in machine learning these days. Given our limited background on convolutional neural networks, we chose to search the web for some neural networks structures that were used for NLP and more particularly for sentiment analysis [4]. We had the chance to find some neural net structures that were used for the same purpose as this project [5]. We thus decided to implement those nets and see how they performed. For our predictions we chose to use ten neural nets with different layer combination. The first five were using the precomputed GloVe matrix [2] as an embedding layer and different combinations of hidden layers. The last five took advantage of n-grams and left the embedding to be learned by the neural net. From Table. 1, we can see that even though in theory n-gram should improve our predictions, the biggest influence on the results come from the embedding. Here, all five models with pretrained GloVe matrix had an accuracy of more than 7 points higher than the ones which left the embedding to the network.

Model	Accuracy
Model 1	0.831
Model 2	0.835
Model 3	0.837
Model 4	0.838
Model 5	0.839
Model 6	0.754
Model 7	0.743
Model 8	0.753
Model 9	0.756
Model 10	0.766

Table 1. Neural networks models performance comparison

Accuracy of each neural network calculated on small dataset

8. BOOSTING THE RESULTS

We have now trained different neural networks that each have good accuracy scores. The idea would be to try to find the strengths and the weakness of each classifiers to improve the overall accuracy. One technique to do this is boosting. It works on the principle of an ensemble; by training a set of weak learners on the data. Each of them will be making conclusion about the various features of each model and subsequently using these conclusions to build a new, stronger model capitalizing on the previous models' classification error.

9. CONCLUSION

Through all these analysis, we ended up with our best accuracy of 0.864 on our submission on CrowdAI. To obtain this result, we used the first 5 models with the pretrained GloVe with 200 embedding dimensions and then boosted the result.

We saw that we did not obtain satisfying results with the neural networks using the n-grams. Based on the results that others previously obtained[5], we expected higher accuracy with this methods, even higher than with the pretrained embedding matrix and we were not able to reproduce these result even when trying to optimize the different parameters. This discrepancy lead us to postulate that our n-gram implementation might not be optimal or that some other preprocessing would have been required in order to get better results.

10. DISCUSSION

By using different models and making small adjustments to the preprocessing, we have managed to optimize our model as much as we could for the basic classification techniques. Then upon switching to neural networks, using the most basic setups that we found the accuracy was increased by almost 10 points. This simple observation shows the shear power of neural networks when applied as machine learning algorithms. We did not have the chance, nor the time to explore and optimize the hyper-parameters and the layers much more. Given more time and more background, there would surely be a possibility for even further improvements.

REFERENCES

1. A. Kub, “Sentiment analysis with python,” https://towardsdatascience.com/sentiment-analysis-with-python-part-1-5ce197074184 (2018).
2. J. Pennington, R. Socher, and C. 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. T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in *Advances in neural information processing systems,* (2013), pp. 3111–3119.
4. M. Sundermeyer, R. Schlüter, and H. Ney, “Lstm neural networks for language modeling,” in *Thirteenth annual conference of the international speech communication association,* (2012).
5. V. B. Yassine Benyahia, Mohammed Hamza Sayah, “Sentiment analysis on twitter data,” https://github.com/Wronskia/Sentiment-Analysis-on-Twitter-data (2016).