Emojify - Sentiment Analysis of Text
Natural Language Processing Challenge

Matteo Pozzan
matteo.pozzan@studenti.unipd.it

Giovanni Vedana
giovanni.vedana.1@studenti.unipd.it

Abstract

In this project, we were asked to experiment with a dataset consisting of written sentences and to explore how machine learning algorithms can be used to associate at each phrase an emoji that captures its content. After having well dug into the dataset, we decided to vectorize the sentences in two different ways, using TF-IDF vectorizer and GloVe. With both of them, we tested several classifiers to perform this sentimental analysis. We reported the results obtained with the various combinations of vectorization and classifier, and we focused in particular on the architectural choices and hyperparameters tuning of Neural Networks. However, the best test accuracy was achieved by a Logistic Regression with the TF-IDF vectorization.

1. Introduction

This study concerns the field of Sentiment Analysis, which is a Natural Language Processing technique that aims to extract subjective information from textual data. Its classic formulation consists of understanding whether a written sentence has a positive or negative meaning. In our project we want to catch a deeper significance of phrases, looking also for sentiments that cannot be simply translated into "good" or "bad". We want to build a classifier that captures the sentiment of a phrase and maps it into an emoticon. Summarizing the content of a phrase is precious information. For instance, in the context of online reviews, it could be useful to make a marketing analysis about the level of appreciation of a product. Furthermore, Sentimental Analysis is a fundamental tool in social networks, where everyone can express their opinion. Since it is impossible to manage the number of comments otherwise, summarizing phrases and catching their root meaning into sentiments become a powerful method to understand how people react to a content. In practice, it is not immediate to build an effective algorithm for sentiment analysis and there is not standard way to do it.

It depends mostly on three ingredients:

- The choice of the training set, i.e. the sentences on which we base the knowledge of our system, that need to fit the algorithm.
- The method of word vectorization, how we translate words into numerical values.
- The choice of the algorithm that classifies the data.

In our work, we looked for the right choice of these aspects, to maximize the accuracy of our model. For what concerns the word vectorization we chose between two different approaches: Bag of Words with *TF-IDF* [1] and word embedding with *GloVe* [2]. We tried several classifiers with both of these approaches: firstly we had some trials with classical labeling methods, such as *Support Vector Machines*, and then we built some *Neural Networks*. We noticed that the result obtained with the *NN* with *GloVe*, after the right choice of architectures and hyperparameters, led to the best validation accuracy, but despite this fact, this model did not provide the best test accuracy.

2. Dataset and Word Vectorization

The dataset, provided by *Kaggle*, consists of 188 phrases of maximum length equal to 10. They were already cleaned and ready to use, and divided into the training, validation, and test set. The training set, formed by 100 phrases, are the phrases used to training the models, while the validation set is composed of 32 phrases and is used for the hyperparameters tuning, to boost the performance of our models. The remaining phrases form the test set and are unlabeled written sentences that are used only at the end to test the best model performance. The five possible labels are integers from 0 to 4 and they represent the emoji we want to associate with each phrase: ❤️, ⚾, 😄, 😔, 🍴. Given the training set, in order to use it to fit the model, we had to translate it into numerical vectors: this process is called word vectorization. We chose to use two different approaches of word vectorization: Bag of Words with a *TF-IDF* vectorizer, which is a classic in the literature of Sentimental Analysis, and a word embedding with *GloVe*, which is more modern and implemented with neural networks. *TF-IDF* is a statistical mea-

sure that evaluates how relevant a word is for each document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document and the inverse document frequency of the word across a set of documents. *TF-IDF vectorize*r works associating to each sample a vector of features, which length depends on a particular choice. We chose these features to be the unigrams, i.e. the unique words that appear in the training set, and the bigrams, which are all the pairs of adjacent words. For each feature, the entrance is the TF-IDF score. We have noticed that the distribution of the words is long-tailored: the most frequent words in the training set are stopwords, for example 'I', 'and', etc., that are less relevant. For this reason, we remove them to reduce the noise. *GloVe*, instead, is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus. Words can be represented as n-dimensional vectors where the distance between points has correspondence for the similarity between word semantics (similar words are closer, while dissimilar ones are distant). In our case, a single word is represented by a vector of length 25. All the phrases are padded to the phrase of maximum length and each phrase is represented by the concatenation of his word embeddings (each phrase thus is a 10 * 25 = 250 dimensional vector). In this case, stopwords are kept: global vectors capture the meaning of a word depending on the context, so they could be meaningful. The procedure used to divide labeled sentences into the training and validation set is called *Hold-Out Validation*: we decided to maintain the original structure due to the unbalanced distribution of the classes. It means that the labels are not uniform, there is a predominance of examples relating to the third and the fourth emoticon. We confirmed our suggestion about the label distribution testing our models with a *K-Cross Validation*: indeed we obtained results less satisfying than the previous because the classes 0, 1, 4 risk to be not represented well in the train set. To overcome this problem more samples should be added to the training set.

3. Method

TF-IDF

Our first experiment was to use *Support Vector Machine* with a *linear* kernel. *Support Vector Machine* could be very much effective in this sentiment clarification because, having already removed some noise (stop words), it should give a little weight to outliers. Later we experimented with other suitable classifiers - *K-nearest neighbors, Logistic Regression, Decision Trees, multinomial Naive Bayes* - here are the best results archived with each of them.

	L-SVC	kNN	LogReg	DT	MNB
Train acc	1	0.88	1	1	1
Val acc	0.72	0.47	0.72	0.62	0.56

The best accuracy, i.e. the number of correctly predicted phrases out of all the phrases, is obtained with *Linear Support Vector Machines* and *Logistic Regression*.

GloVe

We initially bet on *K-Nearest Neighbors* because it assigns to the same class word-vectors that are close and, due to the *GloVe* representations, they could work well together. As we did with *TF-IDF*, we then tried next a bunch of different classifiers, *-SVC, LinearSVC, Decision Trees, Logistic Regression-*, to see if we find a better accuracy.

	kNN	SVC	DT	LogReg
Train acc	1	1	0.83	0.99
Val acc	0.62	0.62	0.56	0.66

KNN did not yield the desired results, on the contrary, the best accuracy is obtained with *Logistic Regression*. Besides, the performance of the models is worst using *GloVe* instead of TF-IDF, even if *GloVe* is a more complex word vectorization.

Neural Networks

The algorithms tested until now gave an adequate accuracy on the validation set, but those models could have some limitations. First, some of them, such as *Linear Support Vector Machines* and *Logistic Regression*, are linear classifiers. This approach could be wrong if the decision boundaries are non-linear. Moreover, the previous classifiers definitely perform well if the starting data are numerical, in such a way that the main features are already visible and the correlation between them is clear, but this is not our situation, because how we translated the training data is not so evident to understand which are the relation between each sample. Therefore we decided to try with an algorithm that should take better into account the interactions between all the features: this improvement could be reached with the use of *Neural Networks*. These models should bring satisfying results because, even if it is not apparent which are the characteristics that lead to a particular class rather than another, *Neural Networks* manage to extract well the main features for the label, thanks to the multiple numbers of layers. This type of classifier, also, build with dense layers, should allow learning from textual data more effectively, because it label each sentence considering how words interact with each other. A dense layer, indeed, is a layer that has an edge for each neuron in adjacent layers: in this way, in each step, the algorithm learns from all the features in the previous layer. This is fundamental for our purpose, because, neglecting some interactions, the meaning of a sentence can totally change. This is

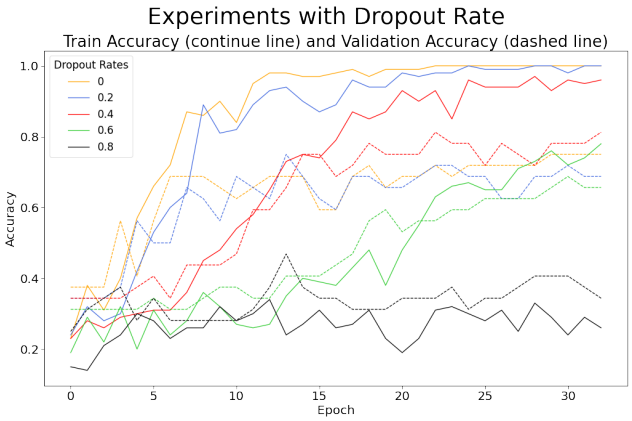
one of the main powerful advantages of *Neural Networks*. Another great aspect provided by this approach is that the *NN* depend on several different hyperparameters. Some of them are standard for multi-classification problems, such as the choice of the *Categorical Cross Entropy* as loss function and the choice of the *softmax* activation function for the output layer, because with this function the output values can be seen as probabilities to belong to a particular class. However, we can change the other parameters to find the optimal model and understand the nature of our problem by looking at the best combination of parameters. For both the word vectorization approaches at first we built some single-layer networks, changing the dimension of the layer, and then we built a more complex model looking for the optimal number of the neuron for each layer. We also tested different activation functions for the hidden layers and different dropout rates to prevent overfitting. To train the models we used a Gradient Descent with the batch size equal to 32: we had some trials with different batch sizes but 32 seemed to be a good one. In the following experiments, we do not mention the tune of this parameter since we preferred to focus on different parameters.

Finally, we think that this approach could perform better with the vectorization with GloVe, because it takes into account more information than TFIDF about the interactions between different words.

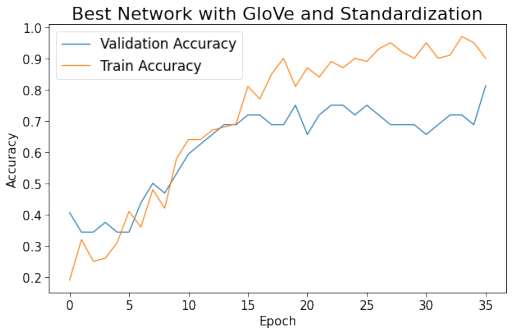
4. Experiments with Neural Networks

Starting with the *TF-IDF* vectorization, we initially searched among the easiest possible models, which are networks with only one hidden layer. We changed the dimension of the layer looking between the powers of 2 from 32 to 1024, noticing that the best performances are reached with an higher number of neurons, giving 0.75 as best accuracy on the validation set. That value, also, is obtained soon, always before 40 iterations. Thus we understood that is not necessary to test our models in a large number of epochs, and for each model, we looked for the epoch that maximizes the validation accuracy. Then we tried a more complex algorithm, changing the number of layers all with the same dimension: in this case, the accuracy does not improve, since it is again 0.75. Moreover, we used 3 different activation function (*relu*, *tanh* and *sigmoid*) and the models with the *relu* functions perform much better than the others. Therefore we took *relu* as our main function, also because it makes the model simpler and more effective, providing great computational advantages. With the previous models, we observed that for our purpose is not necessary to have a deep network, since the best values are reached with models with a small number of layers: after this consideration, we decided to fix that number to be equal to 3 and we changed the dimension of each layer. A *Neural Networks*, built with 128, 512, and 64 neurons for the three hidden layers, gave the best validation accuracy of 0.78125. However, his model could be overfitting, since the train accuracy is 0.96.

With *GloVe* vectorization we retraced the same steps of exploring the architectural choices and, as we expected we obtain better results. The best network gave a train accuracy of 0.96 and a validation accuracy of 0.8125, reached by a network with 64, 1024, and 128 neurons for the hidden layers. Again, the model could be overfitting, hence we tried to solve this problem by changing the *dropout rate*, but we observed that the initial rate of 0.4 is the best.



Finally, as another attempt to reduce the gap between train and validation accuracy, we applied standardization to the features. As can been seen in the graph below, in this case, the validation accuracy remained 0.8125 but we successfully reduced the training accuracy to 0.9.



5. Conclusions

After having enhanced our models on the validation set we checked how they performed on the test data: surprisingly the best test accuracy was achieved by the Logistic Regression with the TF-IDF vectorization. In particular, it is interesting to notice that the best model is neither given by the most complex classifier nor by the most sophisticated vectorization.

	LogReg	NN TFIDF	NN GloVe
Val acc	0.72	0.78	0.81
Test acc	0.68	0.66	0.64

References

[1] *How does bag of words and TF-IDF works in deep learning?*. 2020. URL: https://medium.com/the-programmer/how-does-bag-of-words-tf-idf-works-in-deep-learning-d668d05d281b

[2] *GloVe: Global Vectors for Word Representation*. 2014. URL: https://nlp.stanford.edu/projects/glove/