Machine Learning Project 2019/2020
Emojify - Sentiment Analysis of Text Natural Language Processing Challenge

First Author
Flavio Petruzzellis

Second Author
Marco Francesco Sommaruga

1. Introduction

Natural language is one of the most important means to have insights in opinions, preferences, ideas of people around the world. Since the amount of tweets, posts, speeches generated everyday is enormous, the role played by artificial intelligence is fundamental . In particular, the branch of AI that deals with it is called Natural Language Processing (NLP). NLP, through specific machine learning algorithms, allows machines to extract meaningful information from the human language. One field of NLP is sentiment analysis: it consists in gathering feelings and opinions from a sentence. Thanks to machine learning algorithms we are able to do it automatically and to classify sentences according to their topics or expressed feelings. The aim of this project is to build a 5-label classifier that is able to summarize the sentiment of phrases in the Text to Emoji dataset from Kaggle. This dataset is a small extract from a larger dataset. To accomplish this task, we have chosen to implement a Logistic Regression Model (LR), a Support Vector Machine (SVM) model, and a Feed-Forward Neural Network (FFNN). Due to the scarcity of data, we have mainly focused on reducing as much as possible overfitting for all considered models. In particular, the results achieved with these models are:

Model	Validation Accuracy	Test Accuracy
Log. Reg.	0.71	0.64
SVM	0.73	0.75
FFNN	0.88	0.85

2. Dataset

The dataset we considered is formed by 188 English phrases, with a total of 312 words and 5 different classes of emoji. The dataset comes already split into 100 sentences as train data, 32 as validation data and 56 as test data. Furthermore, all the data contain a vectorized version of the sentences, where each word is represented by a GloVe word embedding of 25 elements. This representations adds significant information about the relationships between the words in the short phrases, which can be successfully exploited in classification. Since each sentence is composed by a different number of words, the dataset has been preprocessed, pooling each representing vector up to a maximum length of 10 words per sentence. Hence, the final train dataset is represented by a tensor (100,25,10). The same procedure was applied on the validation and test dataset. In order to obtain satisfying results, the preprocessing phase is critical. Firstly, we decided to remove the stop-words from the sentences, so that the models were able not to focus on terms that were not meaningful. Due to the embedded representation of words, removing the stop-words required to set to zero all the components of those particular words. Secondly, we standardized the word vectors. Although this modifies the similarity relationships among words, we have observed better behaviors of the models. We justify this observing that the regularization procedure works better with standardized features. Since the amount of available data is not huge, we decided to apply a 10-fold cross validation on the dataset obtained merging the train and validation sets together, evaluating the performance of models with tuned hyperparameters on the test dataset. In order to apply on each validation fold the same preprocessing procedure which was applied to the corresponding training fold, we developed a pipeline to standardize the folds in the cross-validation process. Finally, we exploited the cross-validation procedure together with a grid search to analyze the performances of the models with different hyperparameters values and find the best ones.

3. Models

3.1. Logistic Regression

At first, we decided to approach the classification problem with a Logistic Regression model, which forms a simpler decision boundary with respect to the other two. In order to obtain the best combination of hyperparameters, we performed a grid-search over a selected subset of values for each one. As a result, we were able to obtain a table reporting accuracy values averaged over the 10 folds, together with the corresponding hyperparameters [Figure 1].

CV Test Acc. mean	CV Test Acc. std	CV Train Acc. mean	CV Train Acc. std	CV Acc. difference	solver	penalty	tol	C	class_weight	fit_intercept	max_iter	dual
0.165934	0.024176	0.166657	0.002820	0.000723	liblinear	l1	0.00010	0.010	balanced	False	3500	False
0.165934	0.024176	0.166657	0.002820	0.000723	liblinear	l1	0.00001	0.010	balanced	False	3500	False
0.710440	0.092662	0.799651	0.008511	0.089211	liblinear	l2	0.00100	0.001	balanced	True	3500	False
0.710440	0.092662	0.799651	0.008511	0.089211	liblinear	l2	0.00010	0.001	balanced	True	3500	False
0.710440	0.092662	0.799651	0.008511	0.089211	liblinear	l2	0.00001	0.001	balanced	True	3500	False

Figure 1. DataFrame Log-Reg

Firstly, we considered the best values of average accuracy over the validation sets; then, we tried to gain a deeper insight into the generalization capacity of our models. To do so, we combined the results obtained from the grid search with the study of learning curves obtained by plotting the averaged accuracy over the folds against the number of samples considered. We consequently modified the grid search in order to find the combination of hyperparameters yielding the highest validation accuracy and the lowest difference between training and validation accuracy, so as to find the model presenting the minimal possible symptoms of overfitting. In particular, a great contribution in reducing overfitting has probably been given by the right choice of the regularization parameter C, as shown in [Figure 2].

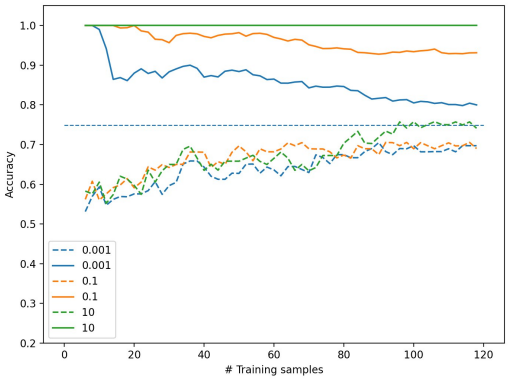

Figure 2. Log-Reg Learning Curve varying Param. C

We finally selected the following parameters for our Logistic Regression model:

solver	liblinear
penalty	l2
max iter	3500
tol	0.001
class weight	balanced
dual	False
fit intercept	True
C	0.001

In particular class weight parameter allows to assign a weight to the instances depending on their classes. The 'balanced' setting of this parameter automatically assigns to each class a weight which is inversely proportional to the frequency of instances of the class, thus helping to deal with class unbalance. This model obtained 71% average accuracy over the validation sets in the 10-folds validation procedure, and 64% accuracy over the test set. The results obtained show that the model is not able to predict with high accuracy the labels on the test set. Analyzing the learning curve we can see that, even with the best choice of hyperparameters, there still is high overfitting, as suggested by the considerable gap (about 10 points) between the train score and the validation score.

3.2. Support Vector Machine

The second model we chose to train was a Support Vector Machine. This family of models is best known for its abilty to form complex decision boundaries although being initially created to solve linear classification problems, thanks to the so-called "kernel trick". Also in this case, we exploited a grid search and the analysis of learning curves to select the best combination of hyperparameters for the model. In particular, we have considered different combination of kernels, C parameter, gamma, r, sample weights, and degree of the polynomial for polynomial kernels. As in the case of logistic regression, the sample weighting scheme can be used to deal with the slight class unbalance in our dataset. The final parameters we chose for our Support Vector Machine are:

kernel	sigmoid
gamma	auto
class weight	balanced
r	1
C	10

The hyperparameter kernel permits to operate in a high-dimensional feature space with the aim to make easier the separation of the data. In particular, using a sigmoid kernel we selected a non-linear kernel of the form $\tanh(\gamma\langle x, x'\rangle + r)$ where r is the intercept constant and γ is such that the higher is its value, the more the model tries to exactly fit the data of the training set. Setting gamma to 'auto', SVM uses the value $\frac{1}{\text{n features}}$. The results we obtained were quite similar to those of Logistic Regression model. The C parameter, analogous to the lambda regularization parameter, was strongly related to the reduction of overfitting, as we can observe in the learning curves in [Figure 3].

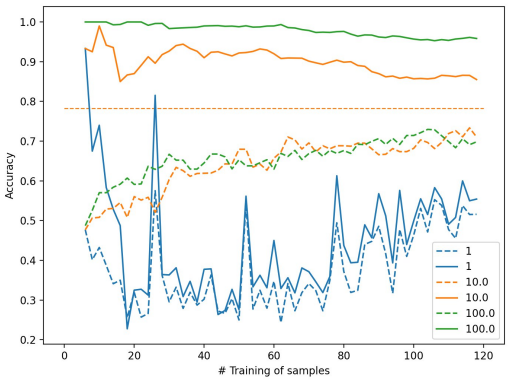

Figure 3. SVM Learning Curve varying Param. C

As in the case of Logistic Regression, as much as overfitting can be reduced, we still observe signals of its presence. However, we find that the average accuracy obtained on the validation sets in the cross validation process are more consistent with what we observe on the test set. Indeed, we obtain about 73% average accuracy on the validation sets and 75% accuracy on the test set. We justify this behavior with the model's capacity to form more complex decision boundaries using a sigmoid kernel, which can be expected to have high overfitting, yet is also able to capture more complex and general trends in the data distribution.

3.3. Feed Forward Neural Network

As a third and last model we implemented a Feed Forward Neural Network. Our aim was to exploit the ability of these more complex models to deal with the nonlinearity we found out to be present in our data. The network takes as input the 250-dimensional vector representing a phrase and outputs a 5-dimensional vector representing the probability distribution over the classes for the input phrase. In order to be consistent with this label representation, we have chosen the Categorical Accuracy as evaluation metric, and the Categorical Cross-Entropy as Loss Function. Furthermore, we used a 1-hot encoding of the training set labels. As a first step, we have searched for the best number of hidden layers via grid search. We considered network structures with one, two and three hidden layers. Our final choice was a model with the following structure:

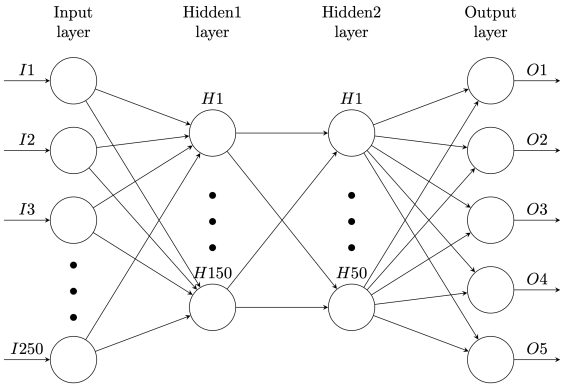

Figure 4. Scheme Feed Forward Neural Network

We have used ReLU activation function for the second hidden layer and a Softmax activation function for the output one, in order to have class probabilities as outputs. Sigmoid activation function was also considered for the hidden layers but did not lead to significant improvements.

The accuracy of the models on k-fold cross-validation was often very high (over 85%) since the very beginning of the training phase, both on training and on the validation set. This is reasonable, since this model is very complex and is indeed able to capture the nuances of our small dataset quite easily. Nevertheless, looking at the loss learning curves [figure 5], we have focused on reducing the model's overfitting, suggested by the training and test loss curves spreading apart.

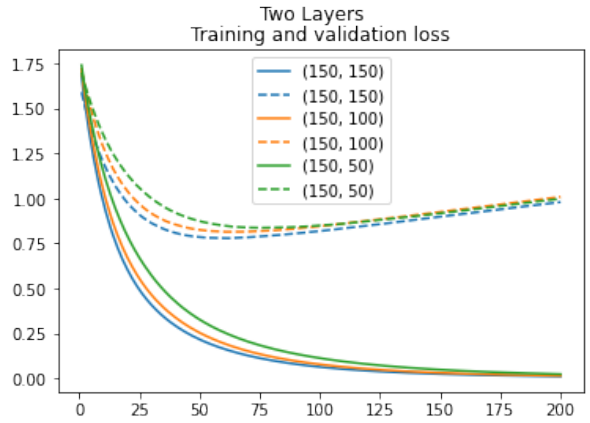

Figure 5. Loss FFNN without regularization parameter varying the number of neurons in the second hidden layer

In order to deal with this severe overfitting, we have added an L-2 regularization term to the second hidden layer weights, performing a grid search over the regularization parameter for each layer. Results in [figure 6].

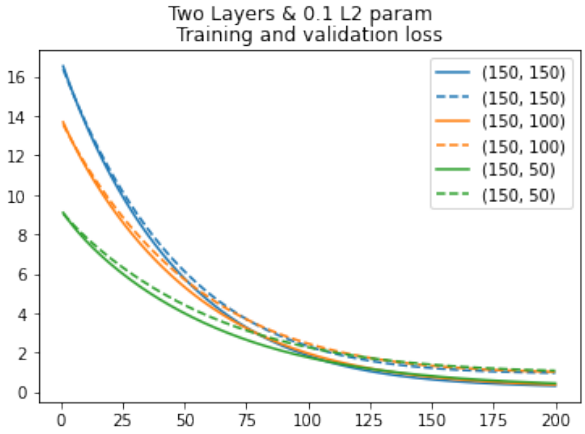

Figure 6. Loss FFNN with regularization parameter varying the number of neurons in the second hidden layer

We also tried to add a Dropout layer to avoid considering some nodes' weights in the backpropagation phase and thus to reduce overfitting. However, this procedure did not lead to significant results. Finally, we tuned the learning rate of the solver, in order to avoid the possible oscillating behavior of the loss function caused by high levels of this hyperparameter, as it can be seen from [figure 7].

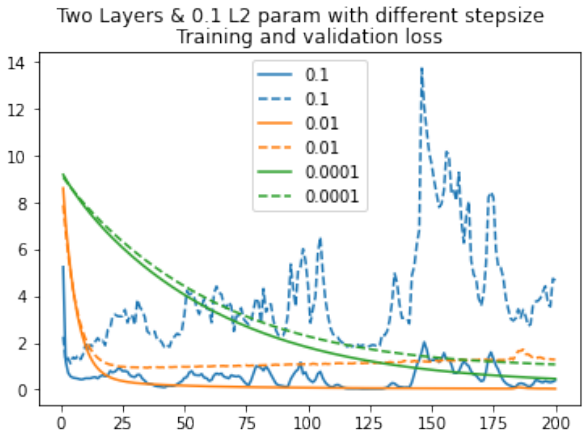

Figure 7. Loss FFNN with different stepsize

At the end, we obtained the best performances with the following hyperparameters:

N input	250
N hidden 1	150
N hidden 2	50
N output	5
L-2 Regularizer hidden 2	0.1
batch-size	10
epochs	200
activation hidden 2	Relu
activation output	softmax
learning rate	0.0001

Our final model reached around 88% average accuracy on the (stratified) k-fold cross validation and, consistently with other models, a slightly lower accuracy (0.85%) on the test data on the Kaggle website.

4. Conclusion

The results we obtained on the Text to Emoji Kaggle dataset are quite consistent under several aspects. Firstly, the predictable nonlinearity of the data is confirmed by the good performance of the SVM and FFNN models. Secondly, we were able to reduce overfitting for all models, applying both data manipulation, (stopword removal, standardization), class balancing and regularization techniques. However, for none of the models it was possible to go below a fixed threshold of overfitting. This is confirmed both by the prediction performances on the test set and on the shapes of the learning curves of all models. In particular, a typical high-variance problem can be detected from these plots, which we assume is due to the scarcity of training data and could be therefore solved by increasing the training set size.

5. Additional Graphs

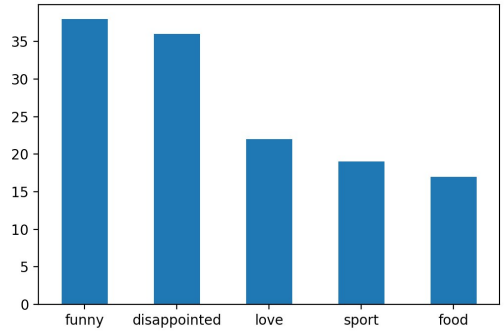

Figure 8. Classes Distribution

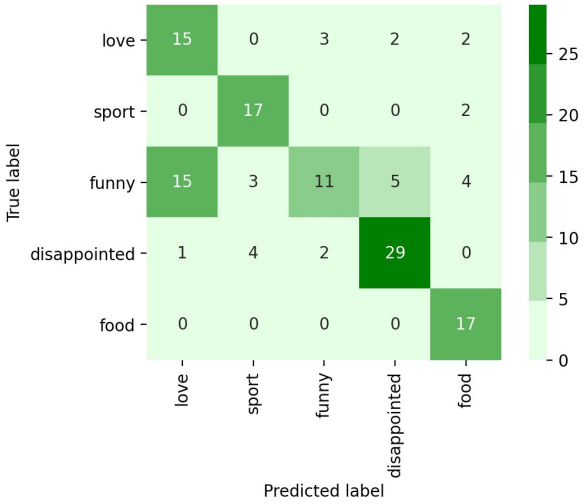

Figure 9. Confusion Matrix Logistic Regression

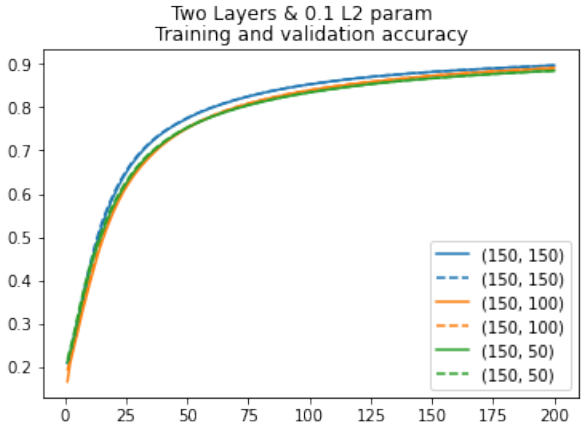

Figure 10. Accuracy FFNN with regularization parameter varying the number of neurons in the second hidden layer