Disaster Tweets Project Report

Rebecca Di Francesco
rebecca.difrancesco@studenti.unipd.it

Stefanija Galevska
stefanija.galevska@studenti.unipd.it

1. Introduction

This project focuses on predicting which tweets are describing disasters, so in another sense it is a binary classification problem. For the purpose of the project, we experimented with text cleaning functions, and after vectorizing the data we applied the following ML models: Logistic Regression, Support Vector Machine, Random Forest and Neural Networks.

2. Dataset

The original dataset in .csv format consists of a collection of tweets with 5 columns (id, keyword, location, text and target). For the aim of the project, we decided to select only the text column as a feature for our predictive model. We decided to discard the location column given the many missing values. Also, we found that it could add noise to our prediction as it is only a user input information and we do not think it is meaningful for predicting whether a tweet is about a disaster or not. Regarding the keyword column instead, we thought for simplicity to avoid using it since it is already present in the text variable, and we wanted to avoid overfitting.

```
id             0
keyword       61
location    2533
text           0
target         0
dtype: int64
```

```
id             0
keyword       26
location    1105
text           0
dtype: int64
```

Figure 1. Missing values in training and test data

We then check for data imbalance and we noticed a slight difference between the two classes, as the figure 2 shows.

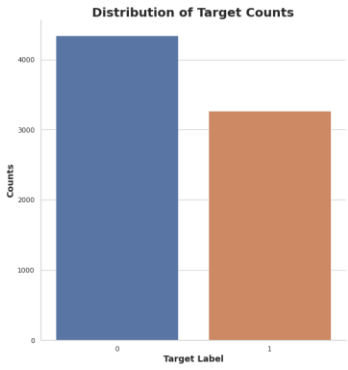


Figure 2. Number of samples for class in the training set

The train.csv dataset contains more tweets labeled as non-disaster tweets (label 0) than disaster tweets (label 1). The imbalance does not require resampling given the difference in number between the two labels is not so big. However, we will use F1 score as a metric to evaluate our model instead of accuracy since accuracy is not reliable when there is a class imbalance. Besides, when we split the dataset in training set and validation set we used the stratification parameter to preserve the same proportions of examples for each class.

2.1 Preprocessing

As a first step, we had to clean the corpus and did that by using some functions to remove url, html, emojis and punctuations as well as lowering all capital letters. Then

we deleted the English stop words that would bias our prediction and we tokenized the textual data by using the Keras Tokenizer method. After separating the tweets into words, we processed our data using the stemming technique to bring all the words in a common base form and finally we vectorized the text using the CountVectorizer.
For the first model, the logistic regression, we further processed the dataset by scaling the features to make the algorithm converge faster and for this we used MaxAbsScaler. Whereas for the NN model the StandardScaling worked better.

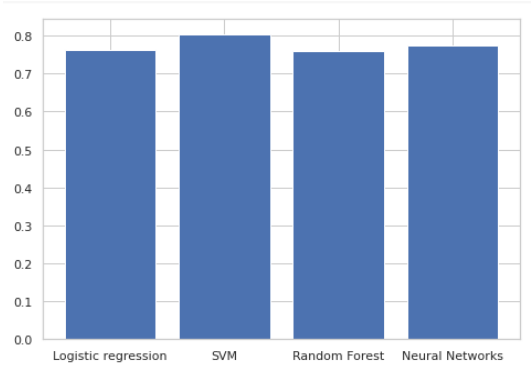


Figure 3. Bar chart of the F1 score calculated on validation sets for different models

3. Methods comparison

We created 4 machine learning models and we experimented with hyperparameter tuning using the sci-kit learn library. The results of the experiments with different parameters are displayed in Table 1 where we have specified the best parameters for each model.

Table 1. F1 scores on training and validation set for different models

Models	Parameters	F1 score training	F1 score validation
LR	'C': 0.5, 'fit_intercept': False	0.8561	0.7621
SVM	'C': 1, 'kernel': 'rbf'	0.9242	0.8035
RF	‘criterion:: entropy, ‘max_depth’ : None, ‘n_estimators’ : 200	0.9841	0.7608
NN	Input layer: 2000 neurons 0 hidden layers Output layer: 2 neurons	0.9180	0.7754

The first method used is a simple Logistic Regression in which we applied a GridSearch to find the best hyperparameters. In this case the model worked best with the regularization parameter tuned as default and with the y-intercept set to 0. The Support vector machine was the next method used in which we tried two kernel types: linear and rbf. The rbf kernel was the best kernel according to the grid search results in combination with default regularization parameter again. This model performed the best out of all four models but based on the difference between the F1 score on the training and validation sets it seems to overfit. Also, if we look at the mean of the F1 scores resulting from the 5-fold splits performed during the grid search this is only 0.73 for these parameters and it has the highest standard deviation among all the folds which means that this model may lead to different results based on different samplings.
The Random Forest is the third model that we used and we trained it for different values for the maximum depth of trees as well as with different numbers of trees. The model with the best parameters got a satisfactory validation score but this is the one which overfits the most with the training score reaching almost 1 as F1 score.
Lastly, we decided to experiment with the most complex model, neural networks. We used a function to create the neural network architect in which we set for the input layer the same number of neurons as the feature shape of the training set. Then we tested it with one optional hidden layer and the results show that in our prediction task it is better to use zero hidden layers, which usually is the case when data is linearly separable. While for the output layer, we set 2 nodes as the number of classes to predict. We used a sigmoid activation function for the output layer and ‘relu’ for the input and hidden layer when fitted.
Eventually we decided to use neural networks as the final model for the submission because apart from SVM it is the best performing model on the validation set. Regarding SVM we already stated our concerns that it may be a misleading model on new unseen data.

4. Conclusion: Neural Networks (NN)

To train our neural networks model we used the stochastic gradient descent optimization algorithm and we used binary cross entropy to minimize the loss function. In the fitting phase, we set 500 for the number of epochs which was controlled by the early stopping mechanism. As for the batch size we tried with the typical batch size: 64, 128, 256, and 512. By increasing it, the F1 score on validation set was getting better but the change from 256 to 512 was so slight that it was not convenient to leave 512 for the computational time involved.
In Figure 4 we plotted the accuracy curves and loss curves for the train and validation sets to see how their shapes changed accordingly to the increasing number of epochs. From 0 to 10 epochs there is a big increase in the accuracy performance. Then, while the accuracy in the training set continues to improve, the accuracy in the validation set keeps fluctuating between 0.70 and 0.80. This could also be a sign that the validation set may not be representative enough of the distribution in the training set since Keras uses its splitting mechanism to compile the model. There is also a clear overfitting from the 20th epoch. The curve for the model loss is smoother both for the training and validation set and it represents a steady decline in the error of the cost function. Even though also the loss curve reaches overfitting from a certain epoch.

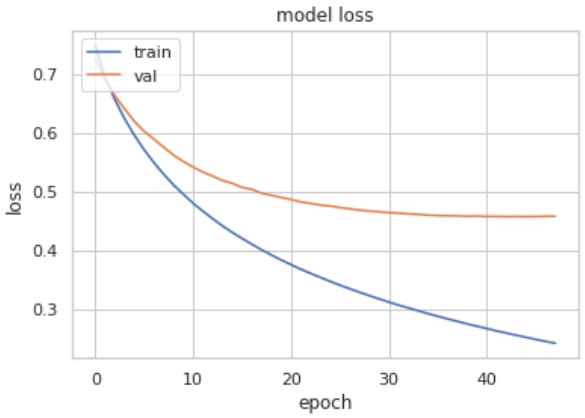
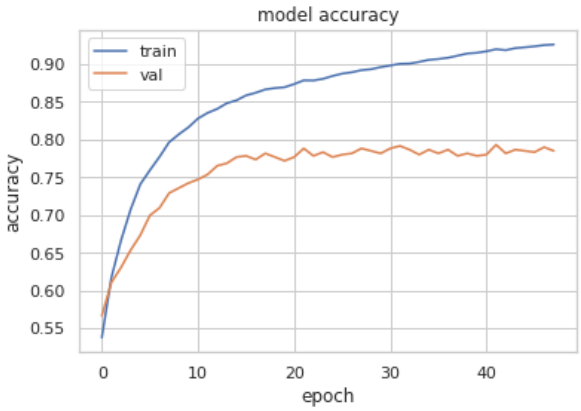


Figure 4. Accuracy and loss curves

To understand better the predictions provided by our neural network models we plotted a confusion matrix (Figure 5). From this we can notice that the model tends to produce more False positives than False negatives, this may be connected to the class imbalance that we saw at the beginning. However, the performance of the model is good enough to be tested on all training plus validation set and see the final F1 score on the test set. After submitting our prediction file on Kaggle we got 0.78 which is a satisfactory score for this task.



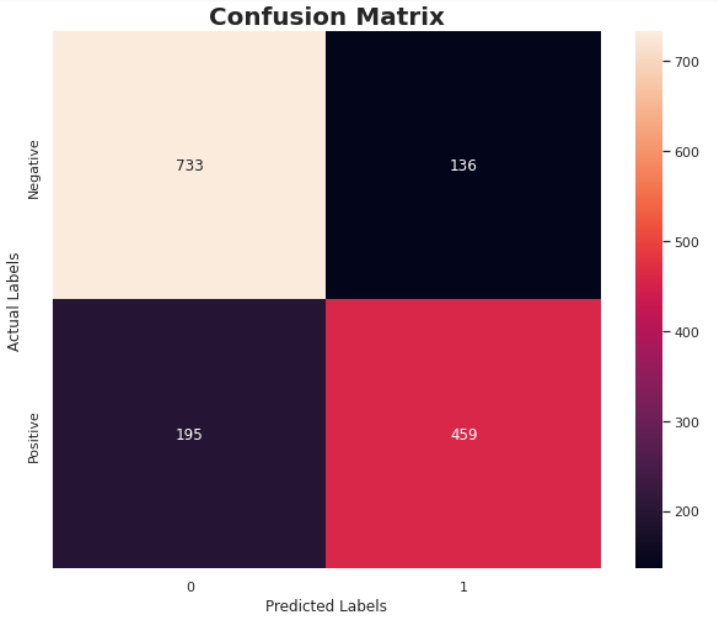


Figure 5. Confusion Matrix

5. References

[1] Pedregosa et al., Scikit-learn: *Machine Learning in Python*, JMLR 12, pp. 2825-2830, 2011
[2] Bengio, *Practical recommendations for gradient-based training of deep architecture*
[3] Bishop, *Pattern recognition and machine learnin*g
[4] Heaton, *The number of hidden layers,* www.heatonresearch.com