Project Documentation - Kaggle Competition

Sora Ioana-Georgiana, 242

Having to establish a model for a binary classification by dialect task, after thorough research, I have come to the conclusion that using a support-vector machine might be the most efficient way to classify the train data.

Algorithm:

1. Data exploring:

We are given 3 .txt samples(train_samples, test_samples, validation_samples) and 2 .txt labels(train_labels, validation_labels).

I chose to open the files by using pandas read method for fixed width files, using 2 parameters: the file path and header = None(because it interprets the first line as a header).

Afterwards I explored the data in order to observe what is taken into consideration, containing multiple columns amongst which the first one had multiple sentence IDs, the second one containing text(for .txt samples) or 0/1 labels(for .txt labels) and all of the other columns had the values NaN. Therefore, I opted for the exclusive use of the text column and transformed it into an array.

2. Data preparation

The fact that we are working with text instead of numerical data requires us to pre-process the given data. Consequently, I chose the bag-of-words algorithm which includes 2 important steps :

1. To create a vocabulary of known words.
2. To measure the presence of known words.

I created a class named BagOfWords which has a constructor that initializes an empty dictionary and an empty list and 2 methods, one for creating the vocabulary and the other to measure the existence of the known words.

So as to normalize the features that resulted from the previous step (bow), I created a method (normalize_data) which contains 4 parameters:

- Train_data
- Validation_data
- Test_data

- Normalization type – which could be: standard, MinMax or L1/L2 distances. If the normalization type is not mentioned, then it returns the unmodified data.

3. Choosing a model

As previously stated, the model chosen for the analysis was the support-vector machine, because it seemed fitter for text classification.

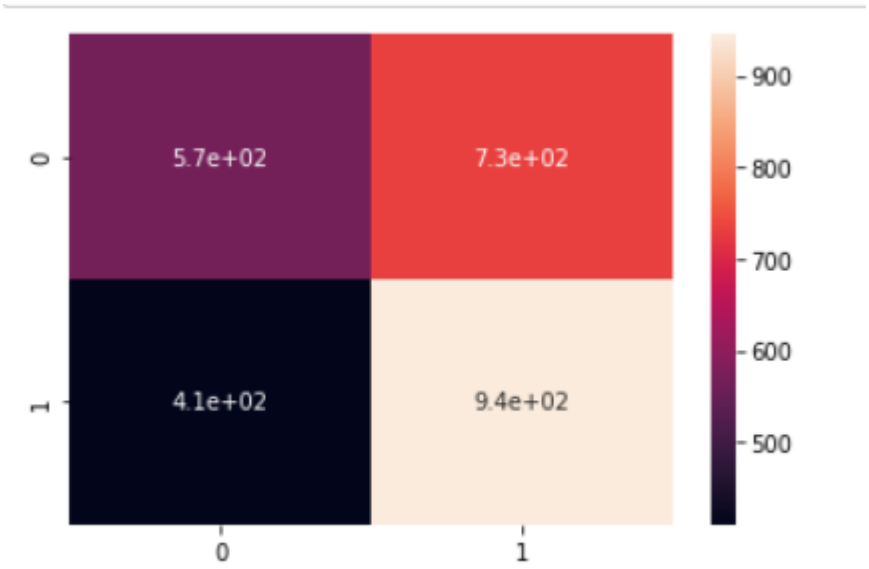
4. Training

In the training part, we teach our model to interpret given data. First of all, I am training the model on train data & labels and I am evaluating it on validation data & labels. Nonetheless, for the final prediction, for test samples, I am concatenating train data with validation data and train label with validation label, knowing that, if I increase the amount of data I am training the model on, it will result in a higher accuracy score.

5. Evaluation

In order to evaluate the model I am considering the following: accuracy_score, f1_score, confusion_matrix and classification_report.

Confusion matrix :



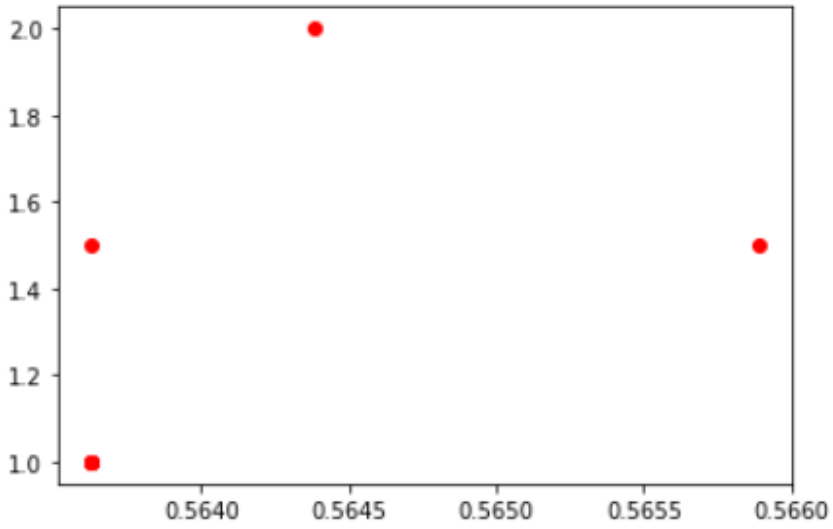
Classification report:

```
Classification report :
              precision    recall  f1-score   support

           0       0.56      0.53      0.54      1301
           1       0.57      0.60      0.59      1355

    accuracy                           0.57      2656
   macro avg       0.57      0.57      0.56      2656
weighted avg       0.57      0.57      0.57      2656
```

6. Parameter Tuning

For improving the results, I endeavoured fluctuating between different types of kernels, normalizations and values for the model's parameters and came to the conclusion that the best accuracy score is obtained when using rbf kernel, L2 distance normalization and C = 1.5.

Also, I created a plot, where I can see the maximum value obtained.



Labeled after C value(y-axis) and accuracy_score(x_axis). I observed that the majority of the tests had the same accuracy score.

7. Prediction

All aspects considered, after running all the mentioned tests, I trained on test_samples + validation_samples(as mentioned above, for better accuracy) and then predicted the labels for test_samples. I created a csv file where I introduced the IDs of the sentences and the predicted outcome.