EKIN OSKAY 24215 CS445 PROJECT 3

REPORT ON TEXT CLASSIFICATION APPROACHES

Naïve Bayes

Files are taken from google drive and first they are lowercased, stopwords are removed and then tokenized. Count Vectorizer is used to fit_transform.

After that, parameters for Naïve Bayes are investigated and found that there are no Naïve Bayes hyper-parameters for Grid Search. More information can be found online:

https://www.reddt.com/r/MLQuestions/comments/6w196i/why_grid_search_is_not_performed_for_naive_bayes/

https://stats.stackexchange.com/questions/299842/why-grid-search-is-not-performed-for-naive-bayes-classifier

Model for Naïve Bayes ran with TfidfVectorizer and MultinomialNB (Gaussian was unable to perform because of RAM limitations of Google Colab).

Accuracy from the Grid Search was %71.7 (Precision ~%77, Recall = %71).

Model then afterwards ran without pipeline and by just using model.fit(train.text, train.label), model ended up getting the same accuracy since there were no hyper-parameter to fine tune by grid search.

Logistic Regression

There were a lot of hyper-parameters available for logistic regression and some are selected they can be found on code. Grid Search is applied for those parameters. Output is written to the code manually since it takes around one hour to run that code block. Best parameters were

```
	clf__max_iter: 20
	clf__penalty: 'l2'
	vect__max_df: 0.5
	vect__max_features: 5000
	vect__ngram_range: (1, 2)
```

And since maximum number of iterations were set to 100000 at the logistic regression, it stops after that many iterations. Best parameters that were found via grid search is applied to same pipeline and the resulting accuracy was found as %81 (Precision = Recall = %81).

Since logistic regression has hyper-parameters that can be fine-tuned via Grid Search, it resulted approximately %10 better accuracy, precision and recall.

In general, both in Naïve Bayes and in Logistic Regression, hardest part was finding the news that were in “turkiye” category. We can understand this from precision and recall (or their combination of f1-score) of the both models for that category is lower compared to others. (Precision NB: %72, Recall NB: %64 ; Precision LR: %59, Recall LR: %67).

Easiest category for both models was “spor” since precision and recall has higher values compared to others. (Precision NB: %89, Recall NB: %94 ; Precision LR: %91, Recall LR: %87)

CNN Classification

Data from the google drive is acquired in the same way and then one-hot encoding is done for labels of both train and test manually. Maximum length of words in any document is calculated.

Train data is tokenized and then padded with default padding parameter, padding = ‘pre’

Afterwards, sequential model is defined. In the first approach, embedding layer is added that would take maximum vocab_size words as an input and would yield 50-dimensional output of those words

Next layer is 1D Convolutional Layer with filters =32, kernel_size = 5 and stride is default (stride =1), softmax activation is selected for this layer.

After that, MaxPooling1D is applied and then Flatten()

Finally, 2 dense layers applied which the last one has units=5 (Since we have 5 classes for the classification.)

According to a result from the last dense layer, category of the news article is decided.

(Example result: [0.6 , 0.2, 0.1 , 0.05 , 0.05])

Gold standart labels: turkiye : [1, 0, 0, 0, 0] , spor: [0, 1, 0, 0, 0] …

For the example result, since 0.6 is the highest among all others, models decide that result belongs to “turkiye” class.

For the loss, “categorical_crossentropy” , “sparse_categorical_crossentropy” and “kullback_leibler_divergence” are used. Best fit is acquired with categorical_crossentropy hence others are discarded from the code.

For optimizer, Adam optimizer is used.

Model is then fitted with batch size of 128 and epoch of 12. Epoch of 2, 10, 12 are tried manually. 2 was not enough for model to fit hence resulting model was underfitting. After that 10 is used and model had validation accuracy of around %60.

Finally, epoch of 12 is selected to see the affect of overfitting if there is any and resulting model with the epoch = 12 had training accuracy of %80 and validation accuracy of %63 which shows overfitting.

For the 2nd model, pre-trained word-embeddigs (word2vec) are acquired from http://github.com/akoksal/Turkish-Word2Vec/

Word2vec was loaded using gensim library and since model was defined on keras it is transformed into keras embedding layer using. Train_emdeddings parameter is set to false because model is thought to be trained well and do not need to be trained anymore that much.

Word2vec.vw.get_keras_embedding(train_embeddings = false)

Then sequential model is created with first embedding layer that is acquired from outside, afterwards 1D convolutional layer with filters = 32, kernel size = 5 using softmax activation function. Then GlobalMaxPooling1D is applied and then same dense layers applied.

Model is compiled with same loss and optimizer. Model reach %67.80 validation accuracy which is %5 higher than the previous model (this could be considered as effect of well-trained word embeddings and we can have higher accuracy results with more data) and %80 training accuracy. Difference between training and validation accuracy is sign of overfitting.

Since our data for CNN is very limited (only 8000 instances for training and 2000 for testing). (Our first model has 42 million parameters that are trainable and next model has 165 million parameters that only 73 thousand of them are trainable). Model is basically memorizing the given data which results in overfitting. In order to get much better results, both models need a lot more data for training and testing.

References:

https://towardsdatascience.com/text-classification-using-naive-bayes-theory-a-working-example-2ef4b7eb7d5a

https://medium.com/analytics-vidhya/applying-text-classification-using-logistic-regression-a-comparison-between-bow-and-tf-idf-1f1ed1b83640

https://www.kaggle.com/shahkan/text-classification-using-logistic-regression

https://towardsdatascience.com/logistic-regression-model-tuning-with-scikit-learn-part-1-425142e01af5

https://medium.com/voice-tech-podcast/text-classification-using-cnn-9ade8155dfb9