Project Report 3

CS 445

Natural Language Processing

Fall 2020-2021

Instructor:

Reyyan Yeniterzi

Faculty of Engineering and Natural Sciences

Sabanci University

Efe Şencan

25083

Preprocessing:

As a preprocessing step, I lowercased all the letters in both train and test set and removed the common Turkish stop words. In addition to the built- in stop words that come from the python stop_words library, I created my own stop words lists and appended these lists to obtain larger stop words list. The additional stop words list are as follows:

additional_stopwords = ['acaba','ama','aslında','az','bazı','belki','biri','birkaç','birşey','biz','bu','çok','çünkü','da','daha','de','defa','diğer','eğer','en','gibi','hem','hepsi','her','hiç','için','ile','ise','kez','ki','kim','mi','mü','nasıl','ne','neden','nerde','nerede','nereye','niçin','niye','o','sanki','şey','şu','siz','tüm','ve','veya','ya','yani','nin','ın','in','nın','nde','den','dan','nda']

Train Data Split:

In order to create validation set, I splitted the provided train set into two sets: new train set and validation set with 8:2 size ratios using the random state “0”.

Number of data instances in each dataset after split:

Train Data: 6400

Validation Data: 1600

Test Data: 2000

Text Classification with Logistic Regression and Naïve Bayes

Term Weighting:

To represent the words into numerical format and then use them in the machine learning algorithms, I utilized two different term-weighting approach: TF and TF-IDF.

Logistic Regression:

Validation Accuracy Before Hyperparameter Tunning

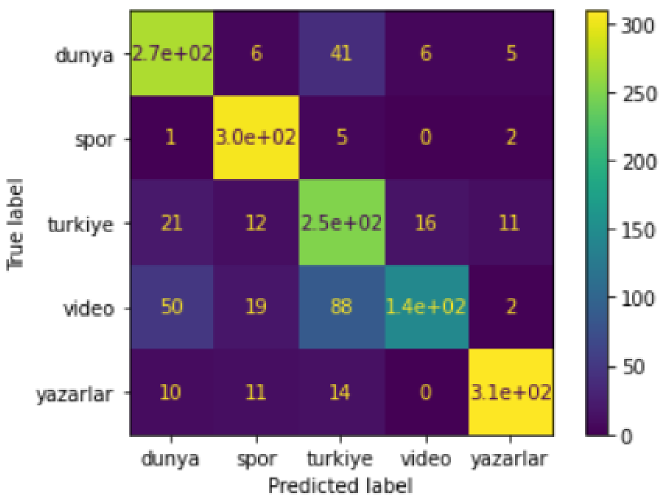
- **TF-IDF Approach:**

F1-Score:

The validation accuracy of the logistic regresison is 0.833125

	precision	recall	f1-score	support
dunya	0.82	0.81	0.82	329
spor	0.93	0.91	0.92	313
turkiye	0.73	0.73	0.73	311
video	0.75	0.80	0.77	303
yazarlar	0.94	0.90	0.92	344
accuracy			0.83	1600
macro avg	0.83	0.83	0.83	1600
weighted avg	0.84	0.83	0.83	1600

Confusion Matrix:



- **TF Approach:**

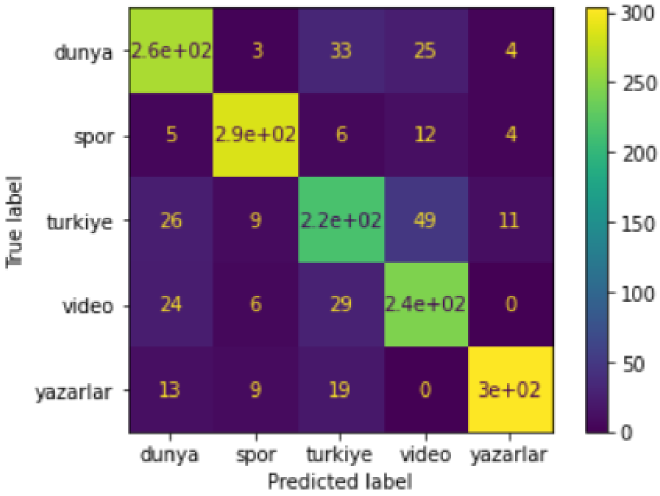
F1-Score:

```
The validation accuracy of the logistic regresison is 0.820625
              precision    recall  f1-score   support

       dunya       0.80      0.80      0.80       329
        spor       0.91      0.91      0.91       313
     turkiye       0.71      0.69      0.70       311
       video       0.74      0.81      0.77       303
    yazarlar       0.94      0.88      0.91       344

    accuracy                           0.82      1600
   macro avg       0.82      0.82      0.82      1600
weighted avg       0.82      0.82      0.82      1600
```

Confusion Matrix:



Hyperparameter Tuning in Logistic Regression:

I tried to optimize 2 hyperparamaters in logistic regression. These are penalty (l_1, l_2, None regularization) that specify the regularization type and C which determines the regularization strength, and fit_intercept.

Optimized Parameters:

{'C': 0.001, Penalty: 'l2', 'fit_intercept'; True}

Train accuracy after hyperparameter tunning: 0.857

Test Accuracy of the Logistic Regression:

- **TF-IDF Approach:**

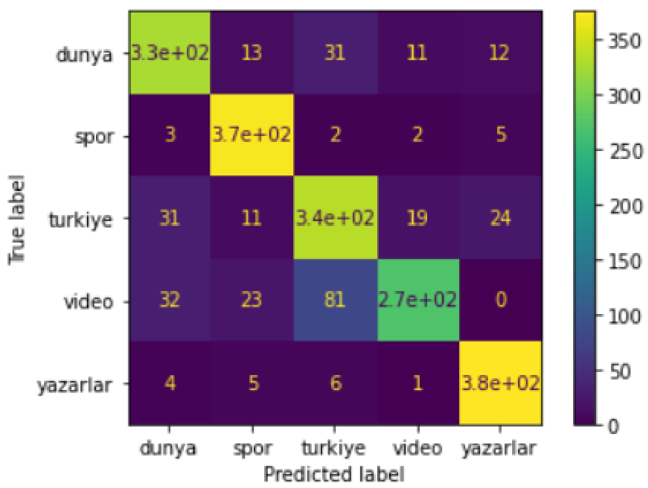
F1-Score:

```
The test accuracy is 0.8545
              precision    recall  f1-score   support

       dunya       0.84      0.83      0.84       395
        spor       0.96      0.92      0.94       384
     turkiye       0.80      0.72      0.76       421
       video       0.76      0.87      0.81       408
    yazarlar       0.94      0.95      0.94       392

    accuracy                           0.85      2000
   macro avg       0.86      0.86      0.86      2000
weighted avg       0.86      0.85      0.85      2000
```

Confusion Matrix:



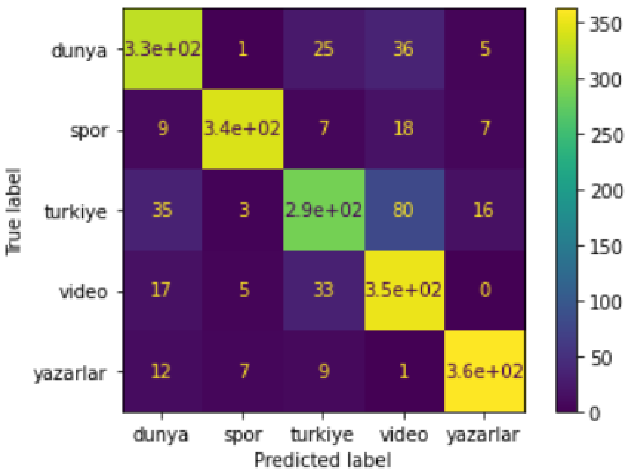
- **TF Approach:**

F1-Score:

The test accuracy is 0.8545

	precision	recall	f1-score	support
dunya	0.84	0.83	0.84	395
spor	0.96	0.92	0.94	384
turkiye	0.80	0.72	0.76	421
video	0.76	0.87	0.81	408
yazarlar	0.94	0.95	0.94	392
accuracy			0.85	2000
macro avg	0.86	0.86	0.86	2000
weighted avg	0.86	0.85	0.85	2000
weighted avg	0.84	0.84	0.84	2000

Confusion Matrix:



Naïve Bayes:

- **TF-IDF Approach:**

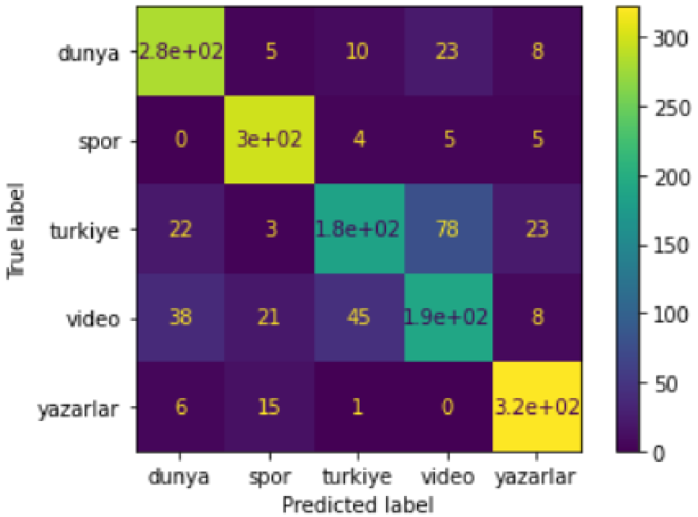
F1-Score:

```
The validation accuracy is 0.8
              precision    recall  f1-score   support

       dunya       0.80      0.87      0.84       329
        spor       0.89      0.92      0.90       313
     turkiye       0.71      0.64      0.67       311
       video       0.67      0.61      0.64       303
    yazarlar       0.89      0.93      0.91       344

    accuracy                           0.80      1600
   macro avg       0.79      0.80      0.79      1600
weighted avg       0.79      0.80      0.80      1600
```

Confusion Matrix:

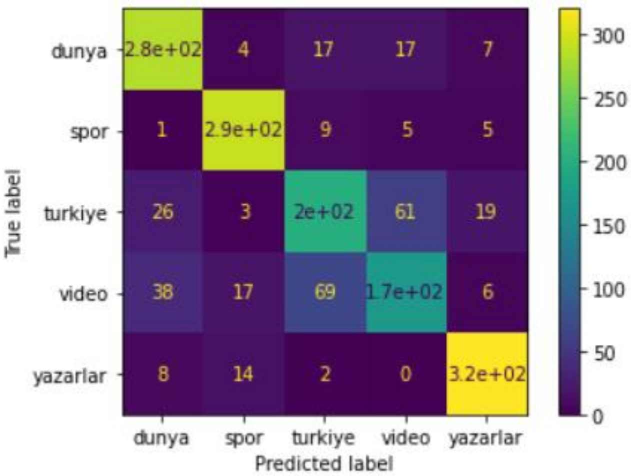


- **TF Approach:**

F1-Score:

	precision	recall	f1-score	support
dunya	0.80	0.86	0.83	329
spor	0.89	0.94	0.91	313
turkiye	0.68	0.65	0.66	311
video	0.68	0.57	0.62	303
yazarlar	0.90	0.93	0.91	344
accuracy			0.80	1600
macro avg	0.79	0.79	0.79	1600
weighted avg	0.79	0.80	0.79	1600

Confusion Matrix:



Hyperparameter Tuning in Naïve Bayes:

I tried to optimize 2 hyperparameters in Naïve Bayes. These are alpha which is the additive smoothing parameter and “fit_prior” which determines whether it learns the prior class probabilities or not.

Optimized Parameters:

{‘alpha’: 1.0, ‘fit_prior’: False}

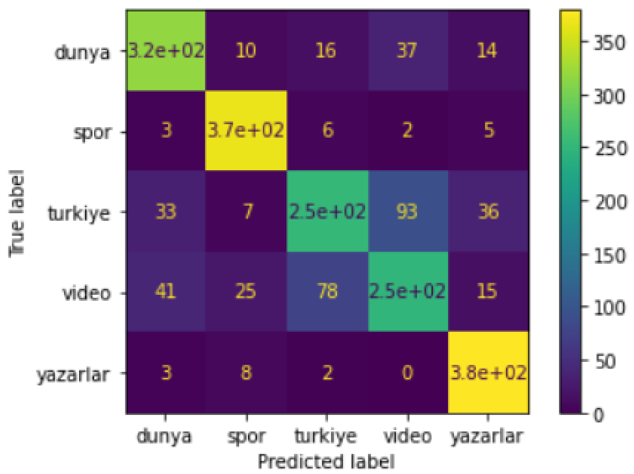
Test Accuracy of the Naïve Bayes:

- **TF-IDF Approach:**

F1-Score:

	precision	recall	f1-score	support
dunya	0.79	0.83	0.81	395
spor	0.90	0.93	0.92	384
turkiye	0.67	0.63	0.65	421
video	0.67	0.58	0.62	408
yazarlar	0.85	0.96	0.90	392
accuracy			0.78	2000
macro avg	0.78	0.79	0.78	2000
weighted avg	0.77	0.78	0.78	2000

Confusion Matrix:

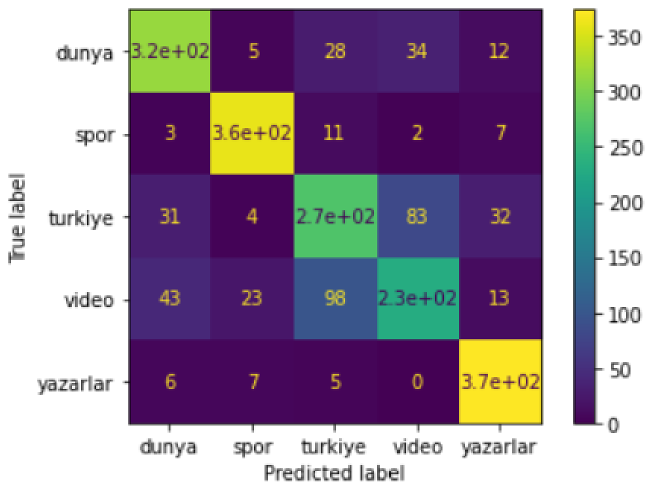


- **TF Approach:**

F1-Score:

	precision	recall	f1-score	support
dunya	0.79	0.80	0.79	395
spor	0.90	0.94	0.92	384
turkiye	0.66	0.64	0.65	421
video	0.66	0.56	0.61	408
yazarlar	0.86	0.96	0.90	392
accuracy			0.78	2000
macro avg	0.77	0.78	0.78	2000
weighted avg	0.77	0.78	0.77	2000

Confusion Matrix:



Effect of term weighting:

We utilized two different term weighting options (TF and TF-IDF) for the text classification. We observed very similar performances in terms of accuracy in the test dataset for both approaches in logistic regression and naïve bayes. Although TF and TF-IDF reached to the same accuracy performance in the test dataset, when we take a look at the weighted average of the scores, TF-IDF approach have %1 greater accuracy than TF approach in naïve bayes and logistic regression algorithms. The difference may due to that, TF-IDF gives more emphasize on words that are text specific and gives their weights accordingly.

Interpretation of the results:

We observed that logistic regression predicts the test labels with 0.85 accuracy while naïve bayes predicts with 0.78 accuracy. Logistic regression is a discriminative model, and it can determine the relation between words by assigning weights to the features and produce a probabilistic value using the sigmoid function for the class estimation part without using any prior probability. Although naïve bayes approach produced a hard to beat baseline, it could not reach to the accuracy level of the logistic regression. The main reason is that naïve bayes is a generative machine learning model and it assumes that features that are in the training set are conditionally independent with each other. Although this simplifying assumption leads to the significant decrease of total number parameters to be estimated, features are not fully independent with each other in the real-world scenario. Therefore, naïve bayes algorithm may produce false probabilistic estimates for the text classification part. When we compare the execution time of the algorithms, we observe that, logistic regression requires more time than the naïve bayes method. The reason is that logistic regression needs to calculate the weight coefficients of each features, while naïve bayes just calculates the prior probability of each class and the likelihood of each class being generated by that particular data.

Test Classification with CNN:

Training with Random Word Embeddings:

Effect of CNN architectures on Test and Validation Accuracy:

- **1 Conv1d Layer, 1 MaxPooling1d Layer, Dense Layer, Output Layer**

Non-Static Embedding:	**Static Embedding:**
Validation Accuracy: 0.845	Validation Accuracy: 0.756
Test Accuracy: 0.829	Test Accuracy: 0.755

- **2 Conv1d Layer, 2 MaxPooling1d Layer, Dense Layer, Output Layer**

Non-Static Embedding:	**Static Embedding**
Validation Accuracy: 0.853	Validation Accuracy: 0.768
Test Accuracy: 0.828	Test Accuracy: 0.767

- **3 Conv1d Layer, 3 MaxPooling1d Layer, Dense Layer, Output Layer**

Non-Static Embedding:	**Static Embedding**
Validation Accuracy: 0.789	Validation Accuracy: 0.780
Test Accuracy: 0.791	Test Accuracy: 0.777

  - **Optimizer**: Adam Optimizer

When we observe different CNN architectures, the test accuracy does not improve as we increase the number of layers. The reason of this situation is that, our dataset is small for the CNN approach and once our model gets more complicated, it is more likely that our model will overfit and hence get lower test accuracy results.

Training with Pretrained Word Embeddings:

Pretrained model of akoksal/Turkish-Word2Vec:

Non-Static Embedding:	**Static Embedding**
Validation Accuracy: 0.84	Validation Accuracy: 0.829
Test Accuracy: 0.836	Test Accuracy: 0.826

Pretrained model of Word2Vec model built in Project 2:

Model is built with **100k** news dataset with Skipgram, window size = 3 and dimension = 100.

Non-Static Embedding:	**Static Embedding**
Validation Accuracy: 0.846	Validation Accuracy: 0.8419
Test Accuracy: 0.836	Test Accuracy: 0.8335

Model is built with **5k** news dataset with Skipgram, window size = 3 and dimension = 100.

Non-Static Embedding:	**Static Embedding**
Validation Accuracy: 0.83	Validation Accuracy: 0.798
Test Accuracy: 0.82	Test Accuracy: 0.80

Experiment with CNN architecture published in public Kaggle Notebook:

- **CNN Architecture**
  - 3 Conv2d Layer, 3 MaxPooling2d Layer, Output Layer
  - Dropout rate 0.5

- Each convolutional layer has filter size 3,4,5 in the respective order

Credit: https://www.kaggle.com/marijakekic/cnn-in-keras-with-pretrained-word2vec-weights

- **Pretrained model of akoksal/Turkish-Word2Vec using that architecture:**

  Non-Static Embedding:

  Validation Accuracy: 0.86

  Test Accuracy: 0.84

- **Pretrained model of Word2Vec built in Project 2 using that architecture:**

  Non-Static Embedding:

  Validation Accuracy: 0.854

  Test Accuracy: 0.844

Interpretation of the results:

When we compare the test accuracy of random embedding and pretrained word embeddings, we can say that CNN with pretrained word embedding performed slightly better than the random embedding. The reason is that, pretrained embedding vector have already trained with corpus and obtained the weight of the features. Hence, it is more likely that it will learn and tune the weight of the new train corpus and produce better probabilities for estimating the class label. When we compare the performance of akoksal's pretrained word embedding with my own model that I trained in Project 2 with 100k dataset, both models reached the same test accuracy score with 83%. This result may seem counterintuitive at the first glance since akoksal's model had been trained with larger corpus. However, we should also consider the domain of the corporas when we are interpreting the results. We used the same domain the "Turkish news" in Project 2 and Project3, therefore the weights of the words that are learned in Project 2 may have more similar weight value with the words in Project 3. On the other hand, akoksal used Turkish Wikipedia texts while training his model. Therefore, the weight of a particular word may differ than the news dataset.

When we compare the performance of the pretrained word embedding model with 5k dataset and 100k dataset, we observed that model trained with 100k dataset have slightly better test accuracy performance.

I also experimented with the CNN architecture that I found in public Kaggle Notebook, and achieved better validation accuracy (%86) compared with my previous CNN architectures since that new CNN architecture is more complex. However, I still get %84 test accuracy using that model which are close to my previous CNN experiments since our dataset is not big enough and our model is more likely to overfit.

When we compare the performance of the static and non-static word embeddings, we can observe that non-static CNN outperformed the non-static models in all experiments in terms of test accuracy. We measured the highest accuracy different between static and non-static models in the random initialized embedding architecture, while the accuracy difference is quite small in the pretrained architectures. Overall, when the embeddings are non-static, the test accuracy of different word embeddings are very close to each other. The reason of this situation may be all of the embeddings are tuned during the training process and the weights that they learned are not kept static. Hence, they could reach similar test accuracy performance.

References:

For logistic and naïve bayes notebook:

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

For CNN notebook:

https://www.kaggle.com/marijakekic/cnn-in-keras-with-pretrained-word2vec-weights

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