# CS 445 Natural Language Processing
# Project 3: Text Classification V.2

Cavit Çakır

23657

# Table of Contents:


Naïve Bayes
- Preprocessing:
- Classification Pipeline:
  - Grid Search:
    - TfidfVectorizer Parameters:
    - MultinominalNB Parameters:
    - Best Parameters:
- Evaluation:
  - Classification_report:
  - Confusion_matrix:

Logistic Regression
- Preprocessing:
- Classification Pipeline:
  - Grid Search:
    - TfidfVectorizer Parameters:
    - LogisticRegression Parameters:
    - Best Parameters:
- Evaluation:
  - Classification_report:
  - Confusion_matrix:

CNN
- Preprocessing:
  - Text:
  - Labels:
- Classification:
  - Final Model with Random Embeddings:
    - Classification Report
    - Confusion Matrix
  - Model with Pre-trained Embeddings from akoksal[2]:
    - Classification Report
    - Confusion Matrix
  - Model with Pre-trained Embeddings from homework2:
    - Classification Report
    - Confusion Matrix

Summarize:

References:

# Naïve Bayes

## Preprocessing:

At first I lowered and removed punctuations & stopwords and stemmed the data with TurkishStemmer[1] then ran the classification pipeline and got 77% classification accuracy, then I tried to lower the data and remove stopwords then executed the pipeline again and I got 78% accuracy. So I decided to keep punctuations and not stem the data.

## Classification Pipeline:

I used TfidfVectorizer from Sklearn in order to represent my text data with vectors. Then give the vectorized data to MultinominalNB from Sklearn.

### Grid Search:

#### TfidfVectorizer Parameters:

```
'use_idf':[True,False],
'norm': [None, 'l1', 'l2'],
'binary': [True, False]
```

#### MultinominalNB Parameters:

```
'alpha': np.linspace(0.5, 1.5, 3),
'fit_prior': [True, False],
```

#### Best Parameters:

Following parameters give the best result according to GridSearchCV from Sklearn. Especially use_idf from TfidfVectorizer helps us to see the difference between different weights.

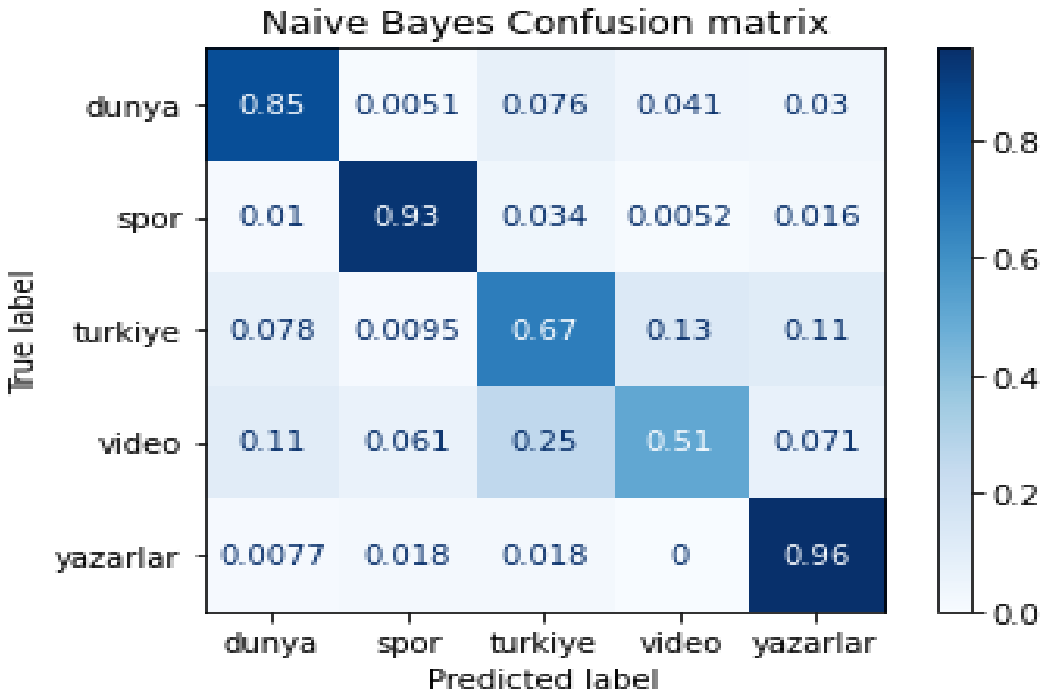
```
'alpha': 1.5,
'fit_prior': True,
'use_idf': True,
'norm': None
'binary': True
```

## Evaluation:

### Classification_report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| dunya | 0.80 | 0.85 | 0.82 | 395 |
| spor | 0.90 | 0.93 | 0.92 | 384 |
| turkiye | 0.65 | 0.67 | 0.66 | 421 |
| video | 0.74 | 0.51 | 0.60 | 408 |
| yazarlar | 0.80 | 0.96 | 0.87 | 392 |
| accuracy | | | 0.78 | 2000 |
| macro avg | 0.78 | 0.78 | 0.78 | 2000 |
| weighted avg | 0.78 | 0.78 | 0.77 | 2000 |

### Confusion_matrix:




As we can see from classification_report from sklearn, we predicted “dunya”, ”spor” and “yazarlar” classes good but news about “turkiye” and “video” are not good.

Also confusion_matrix shows us the model predicted “turkiye” as “video” a lot which supports results in classification_report.

# Logistic Regression

## Preprocessing:

At first I lowered and removed punctuations & stopwords and stemmed the data with TurkishStemmer[1] then ran the classification pipeline and got 83% classification accuracy, then I tried to lower the data and remove stopwords then executed the pipeline again and I got 86% accuracy. So I decided to keep punctuations and not stem the data.

## Classification Pipeline:

I used TfidfVectorizer from Sklearn in order to represent my text data with vectors. Then give the vectorized data to LogisticRegression from Sklearn.

### Grid Search:

#### TfidfVectorizer Parameters:

```
'use_idf':[True,False],
'binary': [True, False]
```

#### LogisticRegression Parameters:

```
'C': np.logspace(-3,3,3),
```

#### Best Parameters:

Following parameters give the best result according to GridSearchCV from Sklearn. Especially use_idf from TfidfVectorizer helps us to see the difference between different weights.

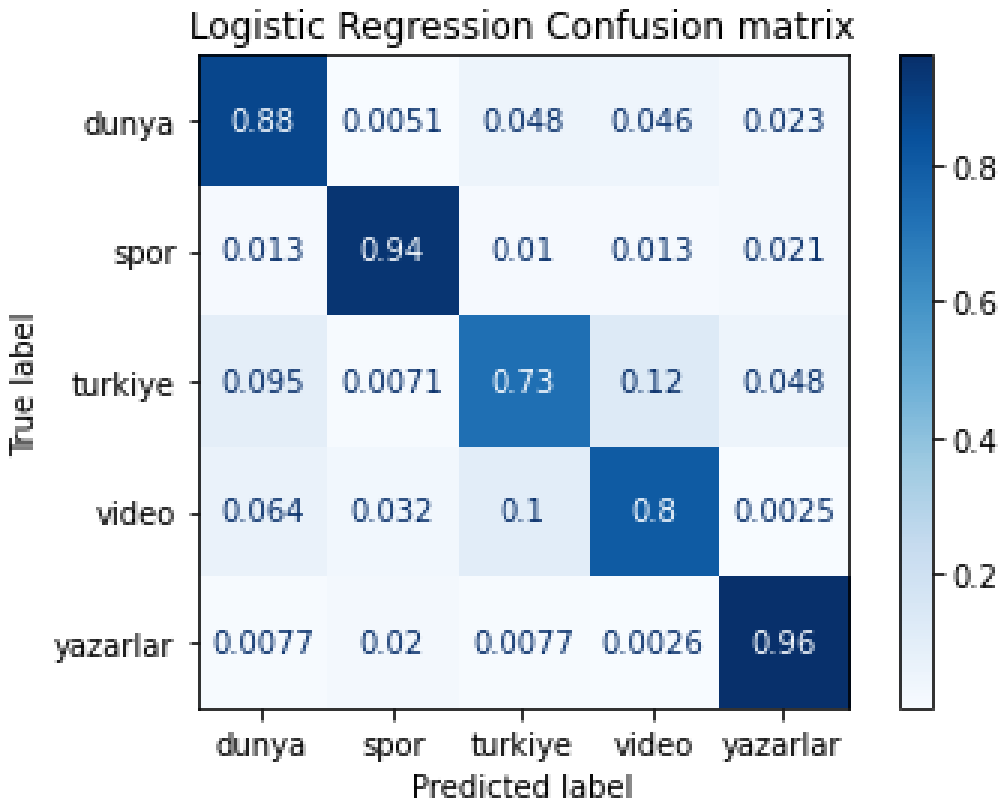
```
'C': 1000,
'use_idf': True,
'binary': True
```

Evaluation:

**Classification_report:**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| dunya | 0.82 | 0.88 | 0.85 | 395 |
| spor | 0.93 | 0.94 | 0.94 | 384 |
| turkiye | 0.82 | 0.73 | 0.77 | 421 |
| video | 0.81 | 0.80 | 0.81 | 408 |
| yazarlar | 0.91 | 0.96 | 0.93 | 392 |
| accuracy | | | 0.86 | 2000 |
| macro avg | 0.86 | 0.86 | 0.86 | 2000 |
| weighted avg | 0.86 | 0.86 | 0.86 | 2000 |

**Confusion_matrix:**



# CNN

## Preprocessing:

### Text:

I extracted digits and lowered all the text then removed stopwords. As an experiment I tried stemming with TurkishStemmer[1] but again it did not improve my overall score.

I created a Tokenizer object and fit my train data into it. Then with the help of the Tokenizer object, I created numeric representation of text and also got the vocabulary size information.

After these steps, I calculated estimated max_token per sample size with the following equation: max_tokens = np.mean(num_tokens) + 2 * np.std(num_tokens) where num_tokens is the total number of tokens. This max_token has covered 97% of my data and helped me to reduce overall complexity which caused by noise.

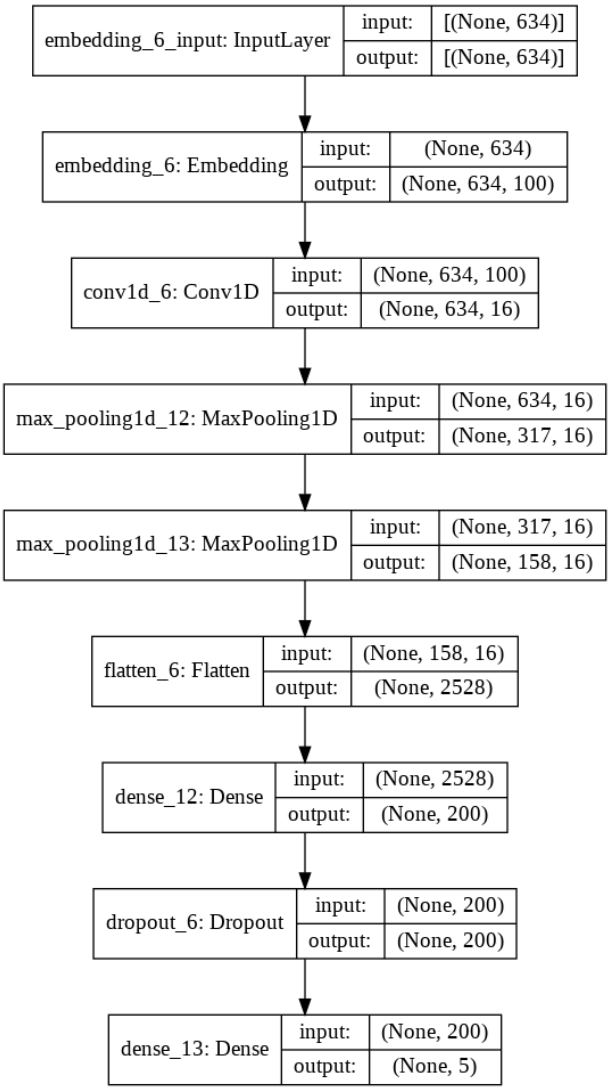
After calculating the max_token size, I padded my data and put an OOV token if I see any out of vocabulary word.

### Labels:

Created one-hot encoding vectors for each label.
‘Spor’ =[1,0,0,0,0]
‘Yazarlar’=[0,1,0,0]
‘Video’ = [0,0,1,0,0]
‘Dunya’=[0,0,0,1,0]
‘Turkiye’=[0,0,0,0,1]

# Classification:

## Final Model with Random Embeddings:

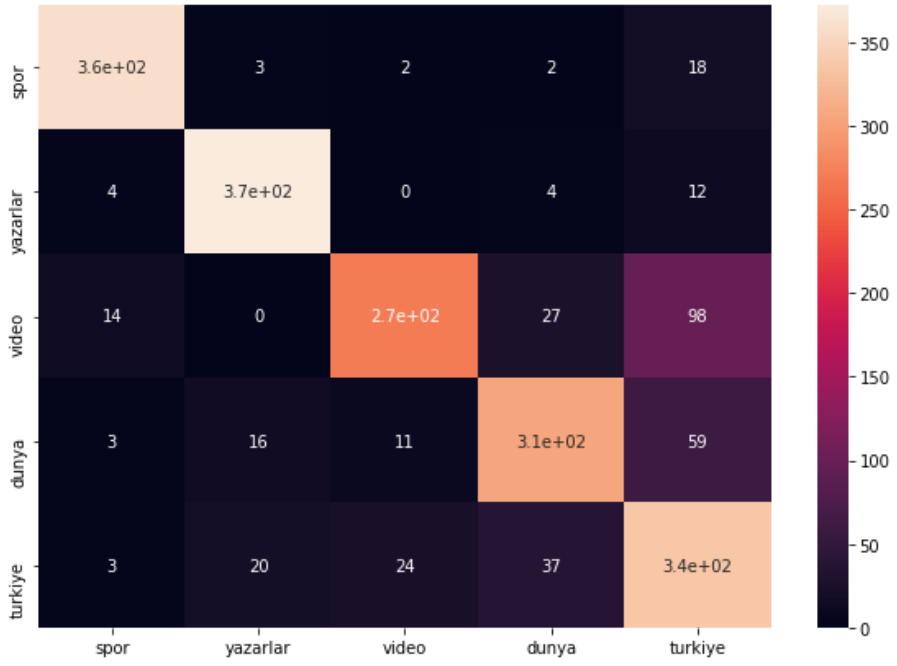



I tried various methods in order to get good results but I got a maximum 82 classification accuracy. I realized that complex models do not work good in our dataset. Because of this fact, I tried simple models and get the best results with the following model;
I created 100 sized word2vec vectors in the embedding layer then put that data into a convolutional layer in order to extract features. After that I used double maxpooling in order to reduce complexity. After these steps I put a dense layer with relu activation function. After that again in order to reduce complexity I put 0.75 dropout layer then I give output with softmax layer.

Classification Report

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| spor | 0.93 | 0.94 | 0.94 | 383 |
| yazarlar | 0.95 | 0.91 | 0.93 | 411 |
| video | 0.66 | 0.88 | 0.75 | 306 |
| dunya | 0.77 | 0.81 | 0.79 | 376 |
| turkiye | 0.80 | 0.64 | 0.71 | 524 |
| accuracy | | | 0.82 | 2000 |
| macro avg | 0.82 | 0.84 | 0.82 | 2000 |
| weighted avg | 0.83 | 0.82 | 0.82 | 2000 |

Confusion Matrix



## Model with Pre-trained Embeddings from akoksal[2]:

I used the same model that I used with Random Embeddings.

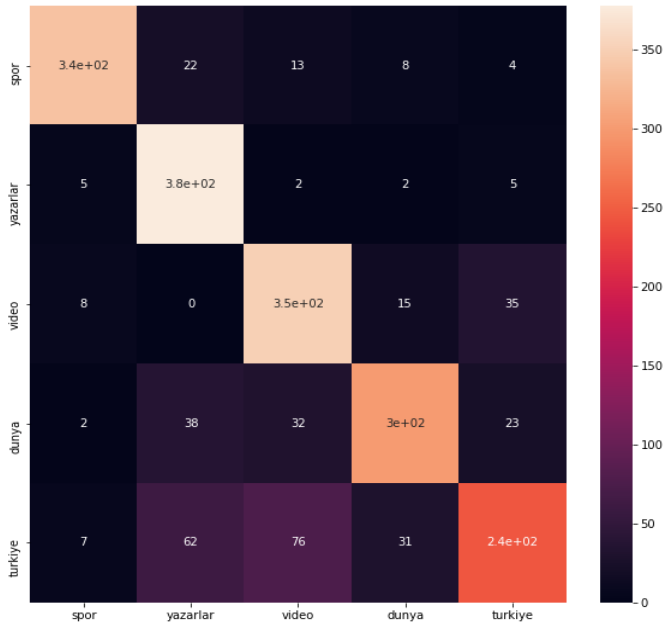
With the pre-trained word-embeddings which sized 400, I got a maximum 81 classification accuracy. I tried training the pre-trained word embeddings in the embedding layer and got lower accuracy. In order to get more accuracy, I didn't trained the embedding layer in the following results.

I think the reason for getting a lower score in akoksal's pre-trained embeddings is the domain of the akoksal's training data. He used wikipedia in order to get those weights.

### Classification Report

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| spor | 0.88 | 0.94 | 0.91 | 359 |
| yazarlar | 0.96 | 0.76 | 0.85 | 500 |
| video | 0.86 | 0.74 | 0.79 | 473 |
| dunya | 0.76 | 0.84 | 0.80 | 356 |
| turkiye | 0.58 | 0.79 | 0.67 | 312 |
| accuracy | | | 0.81 | 2000 |
| macro avg | 0.81 | 0.81 | 0.80 | 2000 |
| weighted avg | 0.83 | 0.81 | 0.81 | 2000 |

### Confusion Matrix



## Model with Pre-trained Embeddings from homework2:

I used the same model that I used with Random Embeddings.

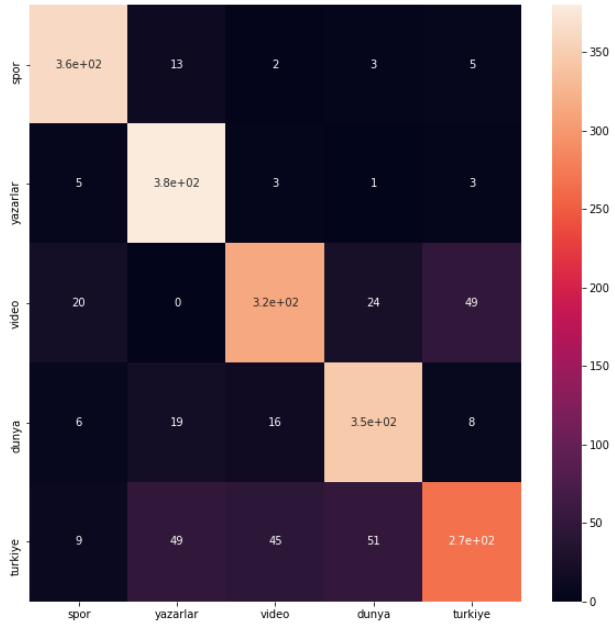
With the pre-trained word-embeddings which sized 32, I got a maximum 83 classification accuracy. I tried not to train the pre-trained word embeddings in the embedding layer and got lower accuracy. In order to get more accuracy, I trained the embedding layer in the following results.

I think the reason for getting a best score in hw2's pre-trained embeddings is the domain of our hw2 dataset is similar to our current dataset.

### Classification Report

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| spor | 0.94 | 0.90 | 0.92 | 401 |
| yazarlar | 0.97 | 0.82 | 0.89 | 461 |
| video | 0.77 | 0.83 | 0.80 | 381 |
| dunya | 0.88 | 0.81 | 0.84 | 425 |
| turkiye | 0.63 | 0.80 | 0.71 | 332 |
| accuracy | | | 0.83 | 2000 |
| macro avg | 0.84 | 0.83 | 0.83 | 2000 |
| weighted avg | 0.85 | 0.83 | 0.84 | 2000 |

### Confusion Matrix



# Summarize:

In order to compare the results;

- ➔ **Naive bayes**: 78 classification accuracy (micro f1-score)
- ➔ **Logistic Regression:** 86 classification accuracy (micro f1-score)
- ➔ **Random Embedding CNN**: 82 classification accuracy (micro f1-score)
- ➔ **Akoksal's Embedding CNN**: 81 classification accuracy (micro f1-score)
- ➔ **Hw2's Embedding CNN**: 83 classification accuracy (micro f1-score)
- ➔ **Extra "dbmdz/bert-base-turkish-cased" BERT model**: 96 classification accuracy (micro f1-score)

At first I implemented Naive Bayes and Logistic Regression models and compared the results, I thought that it's normal because of the nature of the models, what I mean by nature is naive bayes is generative and logistic regression is discriminative models. So, Naïve Bayes assumes all the features to be conditionally independent [3] which is not in our case so we expect to get a lower score in naive bayes. Also, When the training data size is small relative to the number of features, logistic regression can help reduce overfitting and result in a more generalised model [3]. So, when we look at scores, the comparison between naive bayes and logistic regression results were as expected.

After these 2 models, I implemented CNN with random embeddings with a very complex CNN architecture which consists several convolutional layers and too much dense layers and got scores in between 70-80%, at first i thought that I am doing something wrong so I tried a lot of different architectures and asked TA if i am doing something wrong. He recommended me to try more simple models because our data may not be good for complex CNN architectures. I tried really simple models and got results like 83-84% but I still didn't get satisfactory results because I expected more from CNN. After finished my homework, I wonder what score I will get if I put the same data into the "dbmdz/bert-base-turkish-cased" model from huggingface[4] and got 96% classification accuracy. That point I got satisfied but still I do not know how to increase CNN performance.

# References:


- [1] https://github.com/otuncelli/turkish-stemmer-python
- [2] https://github.com/akoksal/Turkish-Word2Vec
- [3]https://dataespresso.com/en/2017/10/24/comparison-between-naive-bayes-and-logistic-regression/
- [4] https://huggingface.co/dbmdz/bert-base-turkish-cased

Following are used for general information and datasets.

- Çiğdem İnan ACI, Adem ÇIRAK, and Berrin Yanıkoglu. "Türkçe Haber Metinlerinin Konvolüsyonel Sinir Ağları ve Word2Vec Kullanılarak Sınıflandırılması"
- https://machinelearningmastery.com/develop-n-gram-multichannel-convolutional-neural-network-sentiment-analysis/
- https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html
- https://machinelearningmastery.com/develop-word-embedding-model-predicting-movie-review-sentiment/
- https://github.com/ibrahimcelenli/cnn-word2vec-tweets-classification
- https://github.com/ibrahimcelenli/news-classification
- https://medium.com/@kocur4d/hyper-parameter-tuning-with-pipelines-5310aff069d6
- https://towardsdatascience.com/multi-class-metrics-made-simple-part-ii-the-f1-score-ebe8b2c2ca1