CSE454 Data Mining

Project Report

Youtube Link: https://youtu.be/Xxb768_953o

Ahmet Tuğkan Ayhan
1901042692

Context

Introduction

In this report, I will describe the project I prepared for the Data Mining course. In this project, I did sentiment analysis on a data set. I downloaded the data set from Kaggle and used it on my own Jupyter Notebook file.

Contents of the Data Set

There are 1 input and 3 outputs for this dataset. Input is a review and outputs are: Positive, Negative and Neutral. At the end of the process, trained models should be able to tell if the given review is a positive, negative or neutral comment.

Here are some of the data inside the data set.

Görüş	Durum
ses kalitesi ve ergonomisi rezalet, sony olduğu için aldım ama 4'de 1 fiyatına çin replika ürün als	Olumsuz
hizli teslimat tesekkürler	Tarafsız
ses olayı süper....gece çalıştır sıkıntı yok.....kablo uzun işinizi çok rahat ettirir.....çekme olayı son	Olumlu
geldi bigün kullandık hemen bozoldu hiçtavsiye. etmem	Olumsuz
Kulaklığın sesi kaliteli falan değil. Aleti öve öve bitiremeyen yorumlar şüpheli. Tizler yok gibi ve o	Olumsuz
Giriş seviyesindeki kullanıcılar için kabul edilebilir sonuçlar veren bir lens, fokus motoru ve makr	Tarafsız
kullanisli baya	Olumlu
Dezavantajlar : Pahalı ürün Merhabalar bu lens başka yerlerde 327 tl ile 389 tl arası satılıyo	Tarafsız
ÜRÜN GÜZEL BU PARALARA BAŞKA BULAMAZSINIZ	Tarafsız
Tasarım ve kalite iyi olmasına rağmen yazma hızı oldukça düşük kalıyor.	Tarafsız
Çok değil çekim gücü az olduğu için 3 puan	Tarafsız
İki kere aldım ikisindede gelen jöle kutusu kırılmış ve içinde jöle paketin içine yayılmıştı.sorun ara	Olumsuz
Klavye tuşları basmakla basmamak arasında kalıyor. Mouse ara sıra takılma yapıyor. Başka da bir şey yok.	Tarafsız
WPA2 ile sorun yaşayanlar eğer masaüstü bilgisayarını kablosuz ağ adaptörü ile kablosuz moder	Olumsuz
Ürün pompasız geliyor. Onun dışında güzel ürün. Pompası olmadığı için 2 yıldız kırdım. Mağaza ürünü zamanında gönderdi.ürün iyi paketlenmişti.	Tarafsız
bu ürünü geçen sene almıştım çok fazla kullanmamama rağmen orta ısıtıcısı bozuldu tavsiye etm	Olumsuz
Mukemmel bir ürün kesinlikle alın	Olumlu
Ürünü genellikle oyun oynayan oğlum için aldım. SD kart takılabildiği için hafızayı artırabileceğim	Olumsuz
Telefonu bir süredir kullanıyorum. Üzerinde bir çizik bilek olmamasına rağmen, konuşurken ekra	Olumsuz
Berbat bir dizayn. Her yerinde açıklık var, aldığı havanını yarısını içeri verir. Berbat kötü.	Olumsuz
Çok hızlı bir şekilde elime geldi akabinde evet oldukça sesli ama elektrik süpürgesi kadar iyi çeki	Tarafsız

Importing the Data

To import data in Jupyter Notebook, we first use **pandas** library to read the csv file.

```
import pandas as pd
```

After importing pandas library, by using **pd.read_csv** we can read the csv file.

```
# Reading and saving the data set into data object
data = pd.read_csv("/kaggle/input/sentiment-analysis/sentiment_analysis.csv", encoding = "utf-16")
```

I uploaded the csv file to the Kaggle as **sentiment_analysis.csv** and read it from there. Since csv file has utf-16 encoding, I specified it in the read function.

To control if the data set is successfully read, we can use head function to show first 5 data in the dataset

```
# Let's see what is inside the data
data.head()
```

	Görüş	Durum
0	ses kalitesi ve ergonomisi rezalet, sony olduğ...	Olumsuz
1	hizli teslimat tesekkürler	Tarafsız
2	ses olayı süper....gece çalıştır sıkıntı yok.....	Olumlu
3	geldi bigün kullandık hemen bozoldu hiçtavsiye...	Olumsuz
4	Kulaklığın sesi kaliteli falan değil. Aleti öv...	Olumsuz

Preprocessing

In the preprocessing part, we need to clear dataset from unwanted data such as null values, outliers, unnecessary words(stopword), etc.

Before starting to preprocessing, I changed the column names from “Görüş”, “Durum” to “Review”, “Status”.

```
data = data.rename({'Görüş': 'Review'}, axis=1)
data = data.rename({'Durum': 'Status'}, axis=1)
```

Removing Null Values

First let’s see if there are any null values in the data set by using **info** function.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11429 entries, 0 to 11428
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Review  11426 non-null  object
 1   Status  11429 non-null  object
dtypes: object(2)
memory usage: 178.7+ KB
```

Here, we acn see that, there is count difference between columns. That means inside review object there should be at least 3 null values. We can remove these null values by using **dropna** function

```
data = data.dropna()
```

After using **dropna** function, we can look at the dataset again to see if the null values are gone or not.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11426 entries, 0 to 11428
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Review  11426 non-null  object
 1   Status  11426 non-null  object
dtypes: object(2)
memory usage: 267.8+ KB
```

Now there are no null values inside our dataset. So we can move to the next step.

Removing Stopwords

Stopwords are words which doesn't affect the meaning of the sentence such as "and", "or", etc. To find Turkish stopwords, we first need to import a library called **nltk.** And from nltk we need to import **stopwords** class.

```
# For Stopwords
import nltk
from nltk.corpus import stopwords
```

To use this class, we need to loop through each word in a sentence and if there is a stopword which used in Turkish, we doesn't include that word to our edited sentence. For example:

```
# Tokenizing the sentence1 after lowercasing the letters inside the sentence
sentence2 = nltk.word_tokenize(sentence2)
# Removing stopwords from sentence
sentence2 = [word for word in sentence2 if not word in set(stopwords.words("turkish"))]
```

Here we first tokenize our sentence to get words and then we loop through the words to find Turkish stopwords, after comparing we don't include the stopwords to our final sentence.

Converting Turkish Letters to English Letters

This process is not necessary but to see if it makes a difference when we build our classification models.

First, we need to convert given sentence to list of words to loop through each letter in a word.

```
# Turning sentence into list of words
listOfWords = list(sentence)
```

Then, we loop through each letter in a word to find Turkish letters. If find one, we convert it to the English equivalent.

```
# Checking for every word in the sentence
for i in range(0, len(listOfWords)):
    if listOfWords[i] == "ü":
        listOfWords[i] = "u"
    elif listOfWords[i] == "Ü":
        listOfWords[i] = "U"
    elif listOfWords[i] == "ö":
        listOfWords[i] = "o"
    elif listOfWords[i] == "Ö":
        listOfWords[i] = "O"
    elif listOfWords[i] == "İ":
        listOfWords[i] = "I"
    elif listOfWords[i] == "ç":
        listOfWords[i] = "c"
    elif listOfWords[i] == "Ç":
        listOfWords[i] = "C"
    elif listOfWords[i] == "ğ":
        listOfWords[i] = "g"
    elif listOfWords[i] == "Ğ":
        listOfWords[i] = "G"
    elif listOfWords[i] == "ş":
        listOfWords[i] = "s"
    elif listOfWords[i] == "Ş":
        listOfWords[i] = "S"
    elif listOfWords[i] == "ı":
        listOfWords[i] = "i"
```

After replacing letters, we need to remerge the words into a sentence by calling join function.

```
# Turning list of words into one string
sentence0 = ''.join(listOfWords)
```

Removing Punctuation Marks

Since we only need words for our sentiment analysis, it is important to remove unwanted values from the dataset. Punctuation marks are one of those unwanted values.

To remove punctuation marks, we need to use regular expressions. One way to use them is like that:

```
# Change every symbol except letters into space
sentence1 = re.sub("[^a-zA-Z]", " ", sentence0)
sentence2 = sentence1.lower()
```

Here we get subsets of the given sentence by using a regular expression and replace every symbol we got except a letter, we turn it into a space. After that, we lowercase all the letters inside a sentence to keep dataset consistent.

Before starting to preprocessing a review inside the dataset was looking like this:

“ürünü hiç ama hiç beğenmedim kimse tavsiye etmiyorum. Ürünün içi plastik

dolu. paslanmaz çelik yazıyor tamam da filtre plastik, vida plastik,

içindeki rezidansın başlığı da plastik kesinlikle sağlıklı bir ürün

değil arkadaşlar. bilseydim almazdım. içinde plastik parçalar olmayan
çelik ya da cam kettle'lar kullanılmalı. “

After done with preprocessing, same sentence looks like this:

“urunu hic hic begenmedim kimse tavsiye etmiyorum urunun ici plastik dolu paslanmaz celik yaziyor tamam filtre plastik vida plastik icindeki rezidansin basligi plastik kesinlikle saglikli bir urun degil arkadaslar bilseydim almazdim icinde plastik parcalar olmayan celik cam kettle lar kullanilmali”

Data Analyzing

Most Frequently Used Words

To find most occurred words in the dataset we need to use a class named **CountVectorizer**.

We can import this class from sklearn like this:

```
from sklearn.feature_extraction.text import CountVectorizer,
```

To use CountVectorizer, we need to specify a value to indicate how many words we want to find. First 10, 100, 1000 ? This parameter is called max_feature, and we can also indicate we don't want to count stopwords. This parameter is not necessary since we already removed them.

Then we use fit_transform and get_feature_names_out function of the CountVectorizer class to print out most frequently used 10 words.

```
cv = CountVectorizer(max_features = 10, stop_words = stopwords.words("turkish"))
space_matrix = cv.fit_transform(replaced_data).toarray() # x

print("Most frequently used {} words {}".format(10, cv.get_feature_names_out()))
```

```
Most frequently used 10 words ['aldim' 'bir' 'cok' 'degil' 'guzel' 'icin' 'iyi' 'tavsiye' 'urun' 'urunu']
```

Word Cloud

Since now we have calculated the most used words inside the dataset, we can use this values to show them in a word cloud view. An example of this can be implemented like this:

```
most_frequently_used_words = cv.get_feature_names_out()

plt.subplots(figsize=(12,12))
wordcloud=WordCloud(background_color="white",width=1024,height=768).generate(" ".join(list(most_frequently_used_words)))
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```

Model Building

I tested my dataset on 3 different machine learning model. In first model, I used Multinomial Naïve Bayes algorithm. I tested this model with oversampled and undersampled data.

I compared the **recall** and **precision** to see if it got better or worse. If it is worse, then probably there is overfitting problem. But this never happened. Almost in every cases, accuracy rate along with recall and precision rates, they increased at least %1-2.

At first, for x axis of my data, I choosed **space_matrix** value which I got from **fit_transform** function as a return value but after doing some research, I noticed that using **TfidfVectorizer** would be a better idea.

TfidfVectorizer is similar to **CountVectorizer**, but it takes into account the relative frequency of each word in the entire dataset instead of just in the individual text. This can help to de-emphasize common words that are unlikely to be informative.

```
vectorizer = TfidfVectorizer()
#X = space_matrix
X = vectorizer.fit_transform(replaced_data)
y = data.iloc[:,1].values
```

To test on real examples I created a function which has 3 sentence inside and first preprocesses these sentences then vectorizes it before predicting using giving model.

Results of the prediction are printed to the console. Code works like this:

```
# Will be used to test models
def testModels(model):
    reviews = [
        "Ürün güzel ancak içinde süzgeci olduğunda kapak tam olarak oturmuyor",
        "Güzel bir ürün işime yaradı.",
        "aldığınıza pişman olursunuz tavsiye etmiyorum.",
    ]

    for review in reviews:
        # Replace Turkish characters with English and remove stopwords, etc.
        review = replaceLetters(review)
        # Use vectorizer
        reviewVec = vectorizer.transform([review])
        # Predict
        prediction = model.predict(reviewVec)

        print(f'Review: {review} | Prediction: {prediction} ')
```

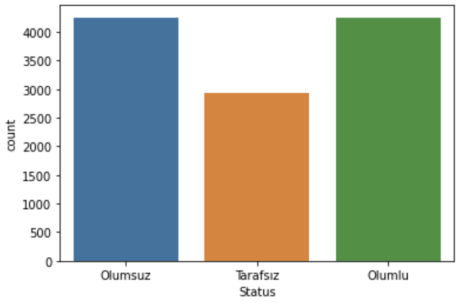
Multinomial Naïve Bayes

Multinomial Naive Bayes classifier is a specific instance of a Naive Bayes classifier which uses a multinomial distribution for each of the features.

I tested this model with normal data, oversampled data and undersampled data. The reason behind this, there is a class imbalance problem with our dataset.

```
sns.countplot(data["Status"])
```

```
<AxesSubplot:xlabel='Status', ylabel='count'>
```



As can be seen in the figure, there are less “Tarafsız” status in the dataset.

To solve this problem, I first oversampled the “Tarafsız” data using SMOTE and then undersampled majority data with RandomUnderSampler.

Normal Data

To analyze model with normal data, we first need to see how do we implement it. Models in this project are implemented using **sklearn** library. For Multinomial Naïve Bayes we need to import **MultinomialNB** class from **sklearn**

```
from sklearn.naive_bayes import MultinomialNB
```

After importing the class, we can implement our model like this:

```
mnb = MultinomialNB()

mnb.fit(x_train,y_train)

y_pred_mnb = mnb.predict(x_test)
```

Here we can see, MultinomialNB constructor returns an object. By using this object we can call **fit** function to train our model.

For train data, we split the X and y objects we got at the start of Model Building section like this:

```
x_train, x_test, y_train, y_test = train_test_split(X,y, test_size = 0.15, random_state = 0)
#x_train, x_test, y_train, y_test = train_test_split(X,y, test_size = 0.25, random_state = 0)
```

Here I tried with different text sizes to see how results change and noticed using %15 of the data as test data gives better results.

After we call predict function with our test data, we can print the results. With our example reviews.

```
print("MultinomialNB Accuracy  : ", accuracy_score(y_pred_mnb, y_test))
print("MultinomialNB Precision : ", precision_score(y_test, y_pred_mnb, average='weighted'))
print("MultinomialNB Recall    : ", recall_score(y_test, y_pred_mnb, average='weighted'))
print("MultinomialNB F1        : ", f1_score(y_test, y_pred_mnb, average='weighted'))
print("\n")
testModels(mnb)
```

```
MultinomialNB Accuracy  :  0.7018669778296382
MultinomialNB Precision :  0.7089524386586405
MultinomialNB Recall    :  0.7018669778296382
MultinomialNB F1        :  0.6509991166986239


Review: urun guzel ancak icinde suzgeci oldugunda kapak tam olarak oturmuyor | Prediction: ['Tarafsız']
Review: guzel bir urun isime yaradi | Prediction: ['Olumlu']
Review: aldiginiza pisman olursunuz tavsiye etmiyorum | Prediction: ['Olumsuz']
```

Instead of only showing accuracy rates, I also calculated precision, recall and f1 scores of the model and printed them. These values are described like this:

Recall (also called Sensitivity or True Positive Rate) is the proportion of actual positive samples that are correctly identified as positive (True Positive) by the model. It is calculated as:

Recall = (True Positives) / (True Positives + False Negatives)

Precision is the proportion of predicted positive samples that are actually positive. It is calculated as:

Precision = (True Positives) / (True Positives + False Positives)

A high recall value means that the model has a low False Negative rate, i.e. it is able to correctly identify most of the positive samples.

A high precision value means that the model has a low False Positive rate, i.e. it is able to correctly identify most of the negative samples as negative.

F1-score is the harmonic mean of precision and recall. It ranges between 0 and 1, where 1 is the best value and 0 is the worst.

Oversampled Data (SMOTE)

To analyze model with oversampled data, we first need to oversample the minority data. For oversampling, I used SMOTE class from **imblearn** library.

```
from imblearn.over_sampling import SMOTE
```

Oversampling is done by getting SMOTE object and using it's fit_resample function to get oversampled x_train and y_train data back. It is used like this:

```
# Create an instance of SMOTE
smote = SMOTE(sampling_strategy='minority',random_state=42, k_neighbors=2)
#smote = SMOTE(sampling_strategy='minority')

# Fit and transform the training data
x_train_oversampled, y_train_oversampled = smote.fit_resample(x_train, y_train)
```

Giving random_state and k_neighbors values like this, increases the success of model. So instead of initializing without parameters I used these parameters.

After getting x_train_oversampled and y_train_oversampled objects, we can apply the same process as we used in normal data set like this:

```
mnb.fit(x_train_oversampled, y_train_oversampled)

y_pred_mnb = mnb.predict(x_test)

print("MultinomialNB_SMOTE Accuracy  : ", accuracy_score(y_pred_mnb, y_test))
print("MultinomialNB_SMOTE Precision : ", precision_score(y_test, y_pred_mnb, average='weighted'))
print("MultinomialNB_SMOTE Recall    : ", recall_score(y_test, y_pred_mnb, average='weighted'))
print("MultinomialNB_SMOTE F1        : ", f1_score(y_test, y_pred_mnb, average='weighted'))
print("\n")
testModels(mnb)
```

```
MultinomialNB_SMOTE Accuracy  :  0.7170361726954493
MultinomialNB_SMOTE Precision :  0.7171052993398405
MultinomialNB_SMOTE Recall    :  0.7170361726954493
MultinomialNB_SMOTE F1        :  0.7158572473137084


Review: urun guzel ancak icinde suzgeci oldugunda kapak tam olarak oturmuyor | Prediction: ['Tarafsız']
Review: guzel bir urun isime yaradi | Prediction: ['Olumlu']
Review: aldiginiza pisman olursunuz tavsiye etmiyorum | Prediction: ['Olumsuz']
```

After oversampling, model gives slightly better results.

Undersampled Data (RandomUnderSampler)

Undersampling works same as oversampling. Here, instead of using SMOTE library, we import RandomUnderSampler class from **imblearn** library.

```
from imblearn.under_sampling import RandomUnderSampler
```

Then, we get RandomUnderSampler object and use it's fit_resample function to get x_train_undersampled and y_train_undersampled objects back.

```
# Create an instance of RandomUnderSampler
rus = RandomUnderSampler(sampling_strategy='majority',random_state=42)

# Fit and transform the training data
x_train_undersampled, y_train_undersampled = rus.fit_resample(x_train, y_train)
```

Here I indicated that under-sampling will be done on majority classes and random_state as 42 to increase success rate of model. After getting x_train_undersampled and

y_train_undersampled objects, we can apply the same process as we used in normal data set like this:

```
y_pred_mnb = mnb.predict(x_test)

print("MultinomialNB_RU Accuracy  : ", accuracy_score(y_pred_mnb, y_test))
print("MultinomialNB_RU Precision : ", precision_score(y_test, y_pred_mnb, average='weighted'))
print("MultinomialNB_RU Recall    : ", recall_score(y_test, y_pred_mnb, average='weighted'))
print("MultinomialNB_RU F1        : ", f1_score(y_test, y_pred_mnb, average='weighted'))
print("\n")
testModels(mnb)
```

```
MultinomialNB_RU Accuracy  :  0.7170361726954493
MultinomialNB_RU Precision :  0.7171052993398405
MultinomialNB_RU Recall    :  0.7170361726954493
MultinomialNB_RU F1        :  0.7158572473137084


Review: urun guzel ancak icinde suzgeci oldugunda kapak tam olarak oturmuyor | Prediction: ['Tarafsız']
Review: guzel bir urun isime yaradi | Prediction: ['Olumlu']
Review: aldiginiza pisman olursunuz tavsiye etmiyorum | Prediction: ['Olumsuz']
```

It is still improvement after normal data but gives same results with oversampled data.

Multinomial Logistic Regression

Multinomial logistic regression is practically identical to logistic regression, except that dataset have multiple possible outcomes instead of just one.

To use multinomial logistic regression, first we need to import LogisticRegression class from sklearn library.

```
from sklearn.linear_model import LogisticRegression
```

After that we call LogisticRegression constructor to return the object. We use this object to call our **fit** and **predict** functions.

```
logr = LogisticRegression(multi_class='multinomial', max_iter=15000)

logr.fit(x_train, y_train)

y_pred_logr = logr.predict(x_test)
```

I changed the iteration count to 15000 since I have more than 10k data. It's initial value is limited with 100 iteration.

Then I used fit function to train model. To test the model, I called predict function with text data. Results are like this:

```
print("Multinomial Logistic Regression Accuracy  : ", accuracy_score(y_pred_logr, y_test))
print("Multinomial Logistic Regression Precision : ", precision_score(y_test, y_pred_logr, average='weighted'))
print("Multinomial Logistic Regression Recall    : ", recall_score(y_test, y_pred_logr, average='weighted'))
print("Multinomial Logistic Regression F1        : ", f1_score(y_test, y_pred_logr, average='weighted'))
print("\n")
testModels(mnb)
```

```
Multinomial Logistic Regression Accuracy  :  0.721120186697783
Multinomial Logistic Regression Precision :  0.7097275547443258
Multinomial Logistic Regression Recall    :  0.721120186697783
Multinomial Logistic Regression F1        :  0.7104623275661603


Review: urun guzel ancak icinde suzgeci oldugunda kapak tam olarak oturmuyor | Prediction: ['Tarafsız']
Review: guzel bir urun isime yaradi | Prediction: ['Olumlu']
Review: aldiginiza pisman olursunuz tavsiye etmiyorum | Prediction: ['Olumsuz']
```

It has a lot better F1 score than Multinomial Naïve Bayes function with normal data. But it is not good than oversampled and undersampled versions of Naïve Bayes model.

Support Vector Machine

To test my dataset with another model, I used support vector machine with no additional parameters.

To be able to use support vector machine, first we need to import SVC class from sklearn library.

```
from sklearn.svm import SVC
```

After that, we use the same syntax with other models to initialize and train our model.

```
svc = SVC()
svc.fit(x_train, y_train)

y_pred_svc = svc.predict(x_test)
```

Once we get return value from predict function, we can calculate score of the model.

```
print("SVC Accuracy  : ", accuracy_score(y_pred_svc, y_test))
print("SVC Precision : ", precision_score(y_test, y_pred_svc, average='weighted'))
print("SVC Recall    : ", recall_score(y_test, y_pred_svc, average='weighted'))
print("SVC F1        : ", f1_score(y_test, y_pred_svc, average='weighted'))
print("\n")
testModels(mnb)
```

```
SVC Accuracy  :  0.7117852975495916
SVC Precision :  0.6992201566022693
SVC Recall    :  0.7117852975495916
SVC F1        :  0.6997154400915995


Review: urun guzel ancak icinde suzgeci oldugunda kapak tam olarak oturmuyor | Prediction: ['Tarafsız']
Review: guzel bir urun isime yaradi | Prediction: ['Olumlu']
Review: aldiginiza pisman olursunuz tavsiye etmiyorum | Prediction: ['Olumsuz']
```

support vector machine gives good recall score but it's F1 and precision scores are worse than every model than I tried except Multinomial Naïve Bayes with normal data model.

Conclusion

As we seen from the model scores, it is important to preprocess dataset before we start modeling. Score table for the models is like this:

Best accuracy score achieved with **Multinomial Logistic Regression (%72.1)**

Best precision score achieved with **Resampled Multinomial Naïve Bayes (%71.7)**

Best recall score achieved with **Multinomial Logistic Regression (%72.1)**

Best F1 score achieved with **Resampled Multinomial Naïve Bayes (%71.5)**

Worst accuracy score achieved with **Normal Multinomial Naïve Bayes (%70.1)**

Worst precision score achieved with **Support Vector Machine (%69.9)**

Worst recall score achieved with **Normal Multinomial Naïve Bayes (%70.1)**

Worst F1 score achieved with **Normal Multinomial Naïve Bayes (%65.0)**