ÇUKUROVA UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

GRADUATION THESIS

SENTIMENT CLASSIFICATION ON SONG PLAYLISTS' DATASET USING SPOTIFY API

By

FATMA AKIN

Advisor

Lect. PhD HAVVA ESİN ÜNAL

January 2021

ADANA

ÇUKUROVA UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

GRADUATION THESIS

SENTIMENT CLASSIFICATION ON SONG PLAYLISTS' DATASET USING SPOTIFY API

By

FATMA AKIN

Advisor

Lect. PhD HAVVA ESİN ÜNAL

January 2021

ADANA

ABSTRACT

SENTIMENT CLASSIFICATION ON PLAYLISTS'

DATASET USING SPOTIFY API

It is very difficult to come across a person who does not use music applications in today's world. We can also say that there are many music applications. Among the others, the Spotify application surpasses all of them with the number of active users and has gone into statistics as the most used application. There were 320 million monthly active users of Spotify according to the company's Q3 2020 report [1]. This makes Spotify a great environment for researchers to work in this field. Also, As Spotify has over 50 Million songs, the possibilities to create large datasets are endless. Based on this, everyone can create their own playlist according to their mood and there is no fixed number of playlists. In this respect, Spotify turns out to be a great dataset for machine learning, data mining and natural language processing.

This study aims to determine sentiment states of a playlist that was created in the Spotify Music App using sentiment classification methods. The process of detection whether these data have any sentiment and whether these sentiments positive, negative, or neutral is defined as sentiment analysis. In this study, sentiment analysis has been performed by using classification algorithms depends on the context of the song's title in the playlist, such as Random Forest Classifier, Logistic Regression, Decision Tree Classifier, Naive Bayes, XGBoost Classifier, and Support Vector Machines (SVM). According to experiments, it is seen that neural networks and support vector machines outperforms with both training and test sets.

Index Terms– Sentiment analysis, Sentiment Classification, Neural network, Support vector machine, Naive Bayes, Random Forest Classifier, Logistic Regression, Decision Tree Classifier, Natural language processing, Data Mining.

CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	:	Artificial Intelligence
POS	:	Part of Speech
SVM	:	Support Vector Machines
API	:	Application Programming Interface
JSON	:	JavaScript Object Notation
VADER	:	Valence Aware Dictionary and sEntiment Reasoner
NLTK	:	Natural Language Toolkit
TF-IDF	:	Term Frequency – Inverse Document Frequency
SVC	:	Support Vector Classifier

TABLE OF CONTENTS

1. INTRODUCTION

So numerous distinctive music apps are applying modern data science and machine learning strategies to our music information and are getting cool outputs. Spotify, for illustration, is one of the foremost well-known apps and is powered by the music we tune in to, the melodies we like, and the playlists we make and take after to deliver a customized item for its users and community. Spotify can generate playlists for us with more upbeat, positive songs, or more mellow and negative songs. So, can sentiment analysis be done according to the song titles in certain playlist? The purpose of this study is to find the answer to this question. And there is also a web API where Spotify shares a certain amount of data to support developers [2].

Sentiment Analysis: Automatically identifying opinions or sentiment expressed in text to determine whether the writer's attitude is positive, negative, or neutral. In this paper, The Spotify Songs Dataset is labelled using sentiment analysis libraries in python for use in sentiment classification applications.

2. MATERIALS AND METHODS

2.1.Materials

Within this project, data analysis was carried out using Jupyter Notebook web application and Python environment.

- The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. Uses include data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more [3].
- Python is an interpreted, high-level, and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects [4].

  Python was created in the late 1980s, and first released in 1991, by Guido van Rossum as a successor to the ABC programming language. Python 2.0, released in 2000, introduced new features, such as list comprehensions, and a garbage collection system with reference counting, and was discontinued with version 2.7 in 2020. Python 3.0, released in 2008, was a major revision of the language that is not completely backward compatible, and much Python 2 code does not run unmodified on Python 3. With Python 2's end-of-life, only Python 3.6.x and later are supported, with older versions still supporting e.g., Windows 7 (and old installers not restricted to 64-bit Windows).

2.2.Sentiment Analysis

Sentiment analysis (also known as opinion mining or emotion AI) refers to the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information. Basically, this involves taking a piece of text, whether it’s a sentence, a comment or an entire document and returning a “score” that measures how positive or negative the text is. While conducting sentiment analysis, a training data set is created using textual data whose opinion side (positive / negative / neutral) was previously determined. Later, these data are made suitable for classification by being subjected to various pre-processes with text mining methods and cleaning. These preprocesses include clearing symbols and punctuation marks in the message, separating the text into words and creating

term lists by finding the roots of each word, removing the stop words consisting of prepositions, conjunctions, and pronouns in the text, creating a vector space model with the help of term frequencies and reverse document frequencies. Various methods such as binary term frequency, inverse document frequency, n grams are used in vector space models. These data, which created a vector space model, are trained with the help of classification algorithms such as artificial neural networks, support vector machines and decision trees, which are frequently used in machine learning. New incoming feelings and opinions are classified with the help of models trained by various classifiers.

In general, sentiment and thought analysis is used to fulfill the following functions.

- Sentiment-opinion classification: It is based on the idea of expressing an opinion about a phenomenon in a document or text and measuring the emotion of the viewer towards the phenomenon.
- Subjectivity Classification: It can generally be expressed as determining whether a sentence is subjective or not. Successful subjectivity classification enables better emotion classification. This process is seen as a more difficult process than separating positive, negative, and neutral sentences.
- Summarizing opinion: It is the concept about removing the main subject from the document and the emotion in its content.
- Obtaining opinion: It is the removal of documents expressing opinion with the query made. In such systems, two types of score calculations should be made; these are the relevance score against the query and the opinion score on the query, both of which are often used to determine the level of documents.
- Emotion Detection: It is mainly detecting song emotions whether negative, positive, or neutral using lyrics. The companies use this emotion detection measure the user's music taste and suggest songs they will likely listen to.

Studies carried out in sentiment analysis until today are generally discussed in two classes on English texts, positive and negative. In some studies, considering the neutrality of thought, examinations were carried out over three classes [5].

When the literature on the field of Sentiment Analysis is examined, it is seen that the first study was done by Pang, Lee and Vaithyanatham, and the movie comments in the Internet Movie Database archive were used as a data set. In their work, they created the

vector space models required for classification by using feature extraction methods such as unigram, bigram, Part of Speech (POS) on the relevant data set. They performed the classification process using machine learning algorithms such as Naive Bayes, Maximum Entropy and SVM on the data set obtained because of vector space models. As a result of the findings obtained, the best result in the classification made for sentiment analysis was obtained with SVM machine learning method with 82.9% accuracy on the unigram dataset [6].

Information in social settings can be used for social, business purposes and many issues. However, extracting this information one by one would not be a reasonable strategy in terms of both time and cost. Therefore, these operations are carried out by developing automatic calculation systems.

In this study, it was aimed to determine whether the song titles in the playlists created in Spotify contain a negative or positive mood and accordingly to calculate the average mood of the playlist. For this purpose, the results were compared using artificial neural networks, support vector machines, Naive Bayes and Random Forest Classifier, Logistic Regression, Decision Tree classifier methods.

2.3. Methods

2.3.1 Dataset

For experiments, songs from public playlists on Spotify were preferred. Spotify web API was used to create the dataset. Based on simple REST principles, the Spotify Web API endpoints return JSON metadata about music artists, albums, and tracks, directly from the Spotify Data Catalogue. Web API also provides access to user related data, like playlists and music that the user saves in the Your Music library. Such access is enabled through selective authorization, by the user. Spotipy library in python was used to use the data in Spotify more efficiently and effectively. Spotipy is a lightweight Python library for the Spotify Web API. With Spotipy you get full access to all the music data provided by the Spotify platform [16]. In order to create the dataset and access the data, it is first necessary to create a Spotify developer account. After that, we create an application that we use the data for. [Figure – 1.1]

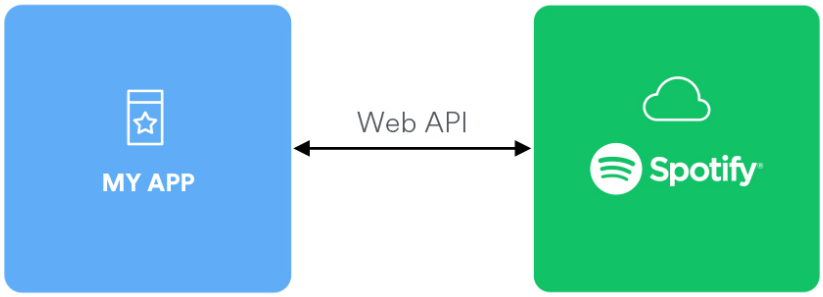

Figure – 1.1 (Web API)

The data set contains 4429 songs, their artist and album name. The dataset concentrated on distinguishing positive and negative emotions.

2.3.2. Data Attributes

In this section, Vader and TextBlob libraries in python environment are used to identify and tag the positive and negative emotions of this dataset. VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media [7]. It is fully open-sourced. And It can be accessible using NLTK library in python. The other one is TextBlob, this library's aims to provide access to common text-processing operations through a familiar interface. You can treat TextBlob objects as if they were Python strings that learned how to do Natural Language Processing [8]. In the first step of this project, it was checked which library is more efficient in sentiment labeling.

The symbols and punctuation marks in all song titles have been cleared in order to prepare the data sets, 1000 of which are divided into positive classes and 1000 into negative classes, for training and classification processes. For this, nltk and str libraries prepared in python environment were used.

2.3.3. Classification Algorithms Used in Experiments

All experiments were carried out using the python software language and in a Jupyter notebook environment. Random Forest Classifier, Logistic Regression, Decision Tree Classifier, XGBoost Classifier, Naive Bayes and Support Vector Machines (SVM), which are classification algorithms, were used on the data set.

In machine learning and statistics, classification is a supervised learning approach in which the computer program learns from the input data and then uses this learning to classify new observations. This data set may simply be bi-class (like identifying whether the person is male or female or that the mail is spam or non-spam) or it may be multi-class. Some practical examples of classification problems are speech recognition, handwriting recognition, bio metric identification, document classification etc.

These operations are carried out by some methods.

For example:

- Count Vectorizer: Text documents are converted into numerical matrices.
- Term frequency - Inverse document frequency (TF-IDF): It shows the importance of the word in the document. The TF-IDF value increases with the increasing frequency of a word in the document.

2.3.3.1. Naive Bayes Classifier (Generative Learning Model)

It is a classification technique based on Bayes' Theorem with the assumption of independence among predictors. In other words, a Naive Bayes classifier assume that the presence of a particular feature in a class is unrelated to the presence of any other feature or that all of these properties have independent contribution to the probability. This family of classifiers is relatively easy to build and particularly useful for very large data sets as it is highly scalable. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods [9].

Bayes' Theorem:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Figure – 2.1 (Bayes' Theorem)

Equation B; feature vector, A is the hypothesis expressing the probability of an attribute vector belonging to a class such as C. P (A | B) represents the successor probability. Considering the Bayes theorem, the algorithm of the Naive Bayes classifier is as follows:

- The above equation was for a single predictor variable, however, in real-world applications, there are more than one predictor variables and for a classification problem, there is more than one output class. The classes can be represented as, C1, C2, … , Ck and the predictor variables can be represented as a vector, x1,x2,…,xn.

- The objective of a Naive Bayes algorithm is to measure the conditional probability of an event with a feature vector x1, x2,…,xn belonging to a particular class Ci,

$$P(C_i|\, x_1, x_2 \ldots, x_n) = \frac{P(x_1, x_2 \ldots, x_n|C_i).P(C_i)}{P(x_1, x_2 \ldots, x_n)} \; for\; 1 < i < k$$

Figure – 2.2 (Naïve Bayes Eq)

- On computing the above equation, we get:

$$\begin{aligned} P(x_1, x_2 \ldots, x_n|C_i).P(C_i) &= P(x_1, x_2 \ldots, x_n, C_i) \\ P(x_1 x_2 \ldots x_n C_i) &= P(x_1 \mid x_2, \ldots, x_n, C_i).P(x_2, \ldots, x_n, C_i) \\ &= P(x_1 \mid x_2, \ldots, x_n, C_i).P(x_2|x_3, \ldots, x_n, C_i)\, P(x_3, \ldots, x_n, C_i) \\ &= \cdots \\ &= P(x_1 \mid x_2, \ldots, x_n, C_i).P(x_2|x_3, \ldots, x_n, C_i) \ldots P(x_{n-1}|x_n, C_i).P(x_n|C_i).P(C_i) \end{aligned}$$

Figure – 2.3 (Equation)

- However, the conditional probability, i.e., P(xj|xj+1,…,xn,Ci) sums down to P(xj|Ci) since each predictor variable is independent in Naive Bayes. The final equation comes down to:

$$P(C_i|\, x_1, x_2 \ldots, x_n) = \left(\prod_{j=1}^{j=n} P(x_j|\, C_i)\right) \cdot \frac{P(C_i)}{P(x_1, x_2 \ldots, x_n)} \; for\; 1 < i < k$$

Figure – 2.4 (Final Naïve Bayes Eq)

- Ultimately, the classifier chooses the class Ci, which has the largest expression P (xj | Ci) P (Ci), as the class of the xj vector.

2.3.3.2. Logistic Regression (Predictive Learning Model)

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression) [10]. Logistic Regression process will be like:

- Given a data (X, Y), X being a matrix of values with m examples and n features and Y being a vector with m examples. Primarily, we create a weight matrix with random initialization. Then we multiply it by features.

$$a = w_0 + w_1x_1 + w_2x_2 + ...w_nx_n$$

Figure- 2.5 (Eq 1)

- Then pass the output obtained from Eq 1. to a link function.

$$\hat{yi} = 1/(1 + e^{-a})$$

Figure – 2.6 (Link Function)

- This is followed by calculating the cost for that iteration whose formula is

$$cost(w) = (-1/m)\sum_{i=1}^{i=m} y_i log(\hat{y}_i) + (1 - y_i)log(1 - \hat{y}_i))$$

Figure – 2.7 (Cost Function)

- The derivative of this cost is calculated following which the weights are updated.

$$dw_j = \sum_{i=1}^{i=n} (\hat{y} - y)x_j^i$$

Figure – 2.8 (Gradient)

$$w_i = w_j - (\alpha * dw_j)$$

Figure – 2.9 (Update)

2.3.3.3. Random Forest Classifier

Random Forest is a flexible, easy-to-use machine learning algorithm that often produces a great result, even without hyperparameter tuning. It is also one of the most used algorithms because it can be used for both simplicity and classification and regression tasks [11].

Random Forest is a supervised learning algorithm. As the name suggests, it creates a forest and somehow randomly does it. The "forest" it has established is a collection of decision trees that are often trained by the "bagging" method. The general idea of the bagging method

is that a combination of learning models increases the overall result. Below you can see how a random forest would look like two trees:

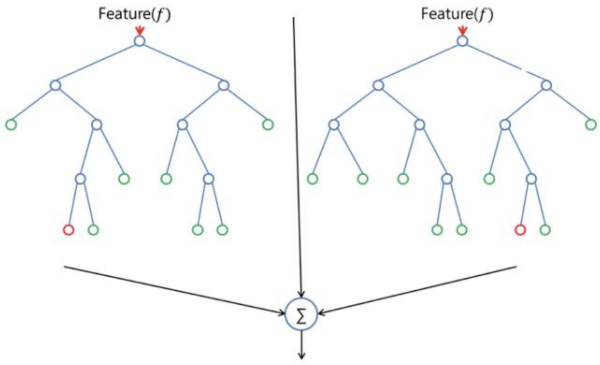

Figure – 2.10 (Random Forest Classifier)

Random Forest adds additional randomness to the model while growing trees. Instead of looking for the most important feature when breaking down a node, it looks for the best feature among a random subset of features. This results in a wide variety that usually results in a better model.

2.3.3.4. Decision Tree Classifier

A Decision Tree is a simple representation for classifying examples. It is a Supervised Machine Learning where the data is continuously split according to a certain parameter [12]. Decision Tree consists of:

- **Nodes**: Test for the value of a certain attribute.
- **Edges/ Branch**: Correspond to the outcome of a test and connect to the next node or leaf.
- **Leaf Nodes**: Terminal nodes that predict the outcome (represent class labels or class distribution).

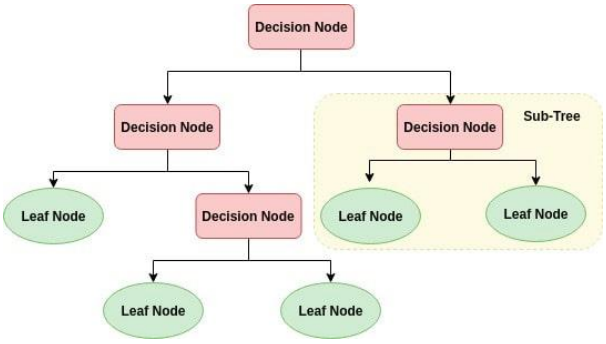

Figure – 2.11 (Decision Tree)

Classification is a two-step process, learning step and prediction step. In the learning step, the model is developed based on given training data. In the prediction step, the model is used to predict the response for given data. Decision Tree is one of the easiest and popular classification algorithms to understand and interpret.

2.3.3.5. Support Vector Machines (SVMs)

In machine learning, support-vector machines (SVMs) are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis[13] .

The support vector machine is capable of separating data into two or more classes with separation mechanisms in the form of linear in two-dimensional space, planar in three-dimensional space and hyperplane in multi-dimensional space [Figure-2.12].

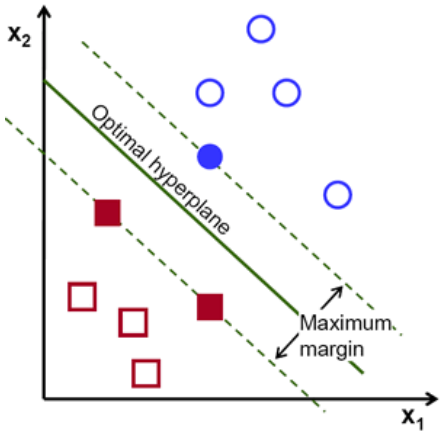

Figure – 2.12 (SVM)

- In the SVM algorithm, we are looking to maximize the margin between the data points and the hyperplane. The loss function that helps maximize the margin is hinge loss.

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$$

Figure – 2.13(Hinge loss function)

$$c(x, y, f(x)) = (1 - y * f(x))_+$$

Figure – 2.14(Represent of Hinge loss function)

- The cost is 0 if the predicted value and the actual value are of the same sign. If they are not, we then calculate the loss value. We also add a regularization parameter the cost function. The objective of the regularization parameter is to balance the margin maximization and loss. After adding the regularization parameter, the cost functions look as below.

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+$$

Figure – 2.15 (Loss function for SVM)

- Now that we have the loss function, we take partial derivatives with respect to the weights to find the gradients. Using the gradients, we can update our weights.

$$\frac{\delta}{\delta w_k} \lambda \parallel w \parallel^2 = 2\lambda w_k$$

$$\frac{\delta}{\delta w_k} \left(1 - y_i \langle x_i, w \rangle\right)_+ = \begin{cases} 0, & \text{if } y_i \langle x_i, w \rangle \geq 1 \\ -y_i x_{ik}, & \text{else} \end{cases}$$

Figure – 2.16 (Gradients)

- When there is no misclassification, i.e., our model correctly predicts the class of our data point, we only must update the gradient from the regularization parameter.

$$w = w - \alpha \cdot (2\lambda w)$$

Figure – 2.17 (Gradient Update – No Misclassification)

- When there is a misclassification, i.e., our model makes a mistake on the prediction of the class of our data point, we include the loss along with the regularization parameter to perform gradient update.

$$w = w + \alpha \cdot (y_i \cdot x_i - 2\lambda w)$$

Figure – 2.18 (Gradient Update – Misclassification)

2.3.3.6. XGBoost Classifier

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples [17].

2.3.4. Criteria Used in Comparison of Classification Algorithms

2.3.4.1. Model performance metrics

The metrics used to compare the classification performance and the related formulas are given below [14].

- Precision: Gives the degree of precision of the classifier result. It is the ratio of the number of positively labeled samples to the total samples classified as positive.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

Figure – 2.19 (Precision Eq)

- Accuracy: It is the most used measurement in the classification process. It is the ratio of correctly classified samples to the total number of samples.

$$Accuracy = \frac{TrueNegatives + TruePositive}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$

Figure – 2.20 (Accuracy Eq)

- Recall: It is the ratio of positively labeled samples to the total number of truly positive samples.
-

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

Figure – 2.21 (Recall Eq)

- F-Measure: F Measure is calculated using precision and sensitivity metrics. It is used to optimize the system towards precision or sensitivity.

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

Figure – 2.22 (F1 Eq)

3. RESULTS AND DISCUSSIONS

In this study, a data set including 2000 songs and artist names was prepared using the Spotify API. In order to use the Spotify API, first a developer account is created, then an app is created to use the API in the created developer account.

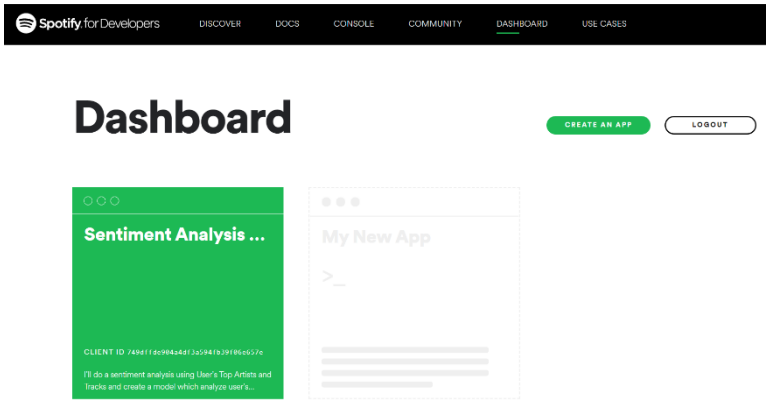

Figure – 3.1 (Spotify Dashboard)

There are two long numbers in this created application as client_id and client_secret. That number are client credentials, and it will use to get an access token. Spotify Web API has data in JSON format. For retrieve the data in python, Spotipy library will be used because it is a lightweight Python library for Spotify Web API.

```
]: import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
cid = 'Your Client ID'
secret = 'Your Secret ID'
client_credentials_manager = SpotifyClientCredentials(client_id=cid, client_secret=secret)
sp = spotipy.Spotify(client_credentials_manager = client_credentials_manager)
```

Figure – 3.2 (Spotify Authentication)

In the code below, a function compares the playlist names of the user, returning the desired playlist first and then the items of the songs in the playlist. In here, “the longest playlist ever” playlist created by user “molly_sp_22”, which is a public playlist, are drawn. Because this playlist contains more data than others. In addition, another playlist containing predominantly positive songs was created for test data.

```
def get_spotify_playlist_data(username='molly_sp_22', playlist=None):

    # set a limit to total number of tracks to analyse
    track_number_limit = 5500


    # get user playlists
    p = None
    results = sp.user_playlists(username)
    playlists = results['items']

    if playlist is None: # use first of the user's playlists
        playlist = playlists[0]['name']

    for pl in playlists:
        if pl['name'] is not None and pl['name'].lower() == playlist.lower():
            p = pl
            break
    while results['next'] and p is None:
        results = sp.next(results)
        playlists = results['items']
        for pl in playlists:
            if pl['name'] is not None and pl['name'].lower() == playlist.lower():
                p = pl
                break

    if p is None:
        print('Could not find playlist')
        return

    results = sp.user_playlist(p['owner']['id'], p['id'], fields="tracks,next")['tracks']
    tracks = results['items']
    while results['next'] and len(tracks) < track_number_limit:
        results = sp.next(results)
        if results['items'][0] is not None:
            tracks.extend(results['items'])
```

Figure – 3.3 (Data Extraction)

But due to the rules of Spotify, only 100 songs can be returned per search. That is why a loop is created to pull all the songs in the playlist. Also, the deluxe and remix versions of some songs cause duplicate data. In order to prevent this, these data are converted into single data within the function before saving the data.

```
    features = []
    # loop to take advantage of spotify being able to get data for 100 tracks at once
    for i in range(len(dat)// 100 + 1):
        fs = sp.audio_features(dat.uri.iloc[i*100:min((i+1)*100, len(dat))])
        if fs[0] is not None:
            features.extend(fs)

    fs = pd.DataFrame(features)

    dat = pd.concat([dat, fs], axis=1)
    dat['track_name'] = track_names

    # ignore live, remix and deluxe album versions
    mask = [(('live' not in s.lower()) and ('deluxe' not in s.lower())
             and ('remix' not in s.lower())) for s in dat.album.values]
    dat = dat[mask]
    mask2 = [(('remix' not in s.lower()) and
              'remastered' not in s.lower()
             and 'version' not in s.lower()) for s in dat.track_name.values]
    dat = dat[mask2]

    dat.set_index('track_name', inplace=True)
    dat = dat[~dat.index.duplicated(keep='first')]
    dat = dat.T[~dat.T.index.duplicated(keep='first')].T

    return dat
```

Figure – 3.4 (Data Extraction)

The data in the train dataset are like this:

```
df = pd.DataFrame(spotify_songs, columns = ['track_name', 'album', 'artists'])
df.to_csv("csv/spotify_songs.csv", sep = ',')
```

```
df
```

	track_name	album	artists
0	Cheers to My Teenage Years	Cheers to My Teenage Years	Chase Cimala
1	Honest	Honest	Levi Bent-Lee
2	Meet The Man	Noah Schnacky EP	Noah Schnacky
3	Ooh Ahh (My Life Be Like) (feat. Toby Mac)	The Art of Translation	Grits, TobyMac
4	Colourblind	Colourblind	Peach PRC
...	...	...	...
4424	Why You Mad?	Why You Mad?	Beau Young Prince
4425	time flies	time flies	dempsey hope
4426	School Girl	Pain is Temporary	Bankrol Hayden
4427	Prom Night	Who's Watching The Kids 2	Mir Fontane
4428	Stuck Up	Stuck Up	Music With Edgar

4429 rows × 3 columns

Figure – 3.5 (Spotify Songs Dataset)

Once the dataset has been created, the data must be positively or negatively labeled for sentiment classification. Two libraries were used for the labeling process. These are the Vader and TextBlob libraries. Before the labelling process, data must be clean for sentiment analysis. In order to prepare the data, set for training and classification processes, symbols and punctuation marks in all song titles were removed and the characters were converted to lowercase letters. For this, the nltk library prepared in python environment was used.

```
# let's experiment with some sentiment analysis concepts
# first we need to clean up the stuff in the independent field of the DF we are workign with
df['track_name'] = df[['track_name']].astype(str)

df['track_name'] = df['track_name'].str.replace('\d+', '')

# get rid of special characters
df['track_name'] = df['track_name'].str.replace(r'[^\w\s]+', '')

# get rid fo double spaces
df['track_name'] = df['track_name'].str.replace(r'\^[a-zA-Z]\s+', '')

df['track_name'] = df['track_name'].str.replace('\feat+', '')

# convert all case to lower
df['track_name'] = df['track_name'].str.lower()
```

Figure – 3.6 (Cleaning Process)

After clearing the data, the sentiment polarity values of the data in two libraries were calculated and labeled as negative or positive. When the labeled data were compared, it was seen that the data contained predominantly positive values. Somehow it turned out that Vader has more negative values tagged than the TextBlob library.

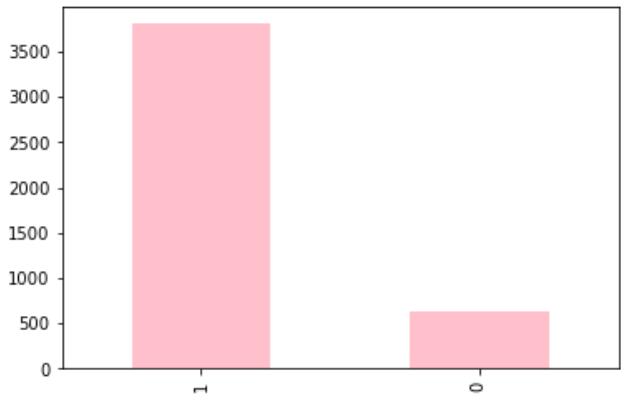

Figure – 3.7 (Vader Sentiment Analyzer)

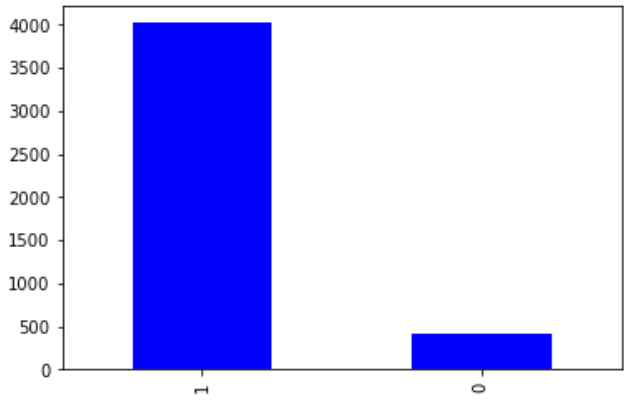

Figure – 3.8 (TextBlob Sentiment Analyzer)

As a result of this comparison, the dataset labeled with Vader will be used in the continuation of the project.

Labeled data set in the below like this:

```
df
```

	track_name	artists	sentiment	label
0	cheers to my teenage years	Chase Cimala	{'neg': 0.0, 'neu': 0.563, 'pos': 0.437, 'comp...	1
1	honest	Levi Bent-Lee	{'neg': 0.0, 'neu': 0.0, 'pos': 1.0, 'compound...	1
2	meet the man	Noah Schnacky	{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...	1
3	ooh ahh my life be like feat toby mac	Grits, TobyMac	{'neg': 0.0, 'neu': 0.762, 'pos': 0.238, 'comp...	1
4	colourblind	Peach PRC	{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...	1
...	...	...	...	...
4424	why you mad	Beau Young Prince	{'neg': 0.615, 'neu': 0.385, 'pos': 0.0, 'comp...	0
4425	time flies	dempsey hope	{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...	1
4426	school girl	Bankrol Hayden	{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...	1
4427	prom night	Mir Fontane	{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...	1
4428	stuck up	Music With Edgar	{'neg': 0.667, 'neu': 0.333, 'pos': 0.0, 'comp...	0

4429 rows × 4 columns

Figure – 3.9 (Labeled Data)

Now that the data is labeled, the necessary preparations can be made for the classification process. First, the frequency of the most used words in the dataset was calculated using the count vectorizer operation.

```
from sklearn.feature_extraction.text import CountVectorizer


cv = CountVectorizer(stop_words = 'english')
words = cv.fit_transform(train['track_name'].values.astype(str))

sum_words = words.sum(axis=0)

words_freq = [(word, sum_words[0, i]) for word, i in cv.vocabulary_.items()]
words_freq = sorted(words_freq, key = lambda x: x[1], reverse = True)

frequency = pd.DataFrame(words_freq, columns=['word', 'freq'])

frequency.head(30).plot(x='word', y='freq', kind='bar', figsize=(15, 7), color = 'blue')
plt.title("Most Frequently Occuring Words - Top 30")
```

Figure – 3.10 (Count Vectorizer)

The most common word in the dataset was the word "love". This is not surprising because the word love is used in songs in a sad sense, but also in the sense of happiness.

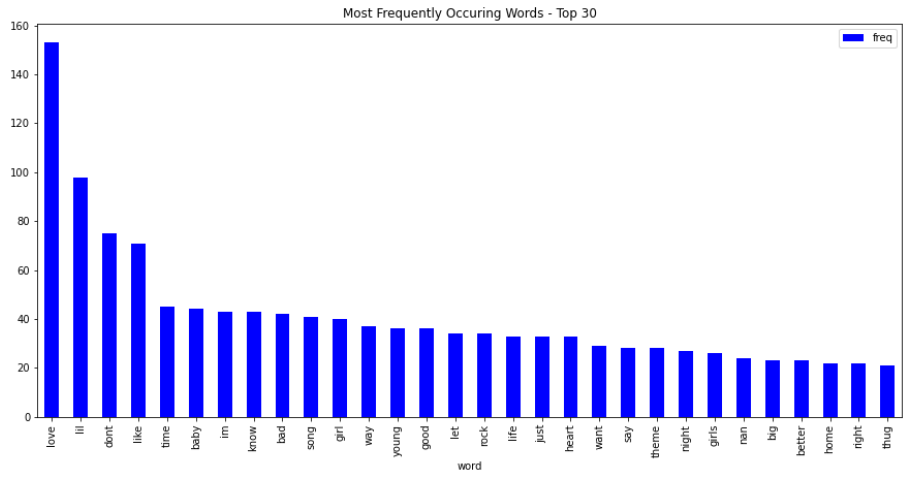

Figure – 3.11 (Word Frequencies)

After that, Which words occur in negative labeled songs can be seen. By looking at the word cloud below, it can be seen that the negative-labeled songs mainly contain words like lonely, love, heart, wrong or some abusive words.

Figure – 3.12 (Negative words)

By looking at another word cloud below, it can be seen that the positive-labeled songs mainly contain words like girl, love, heart, back or summer.

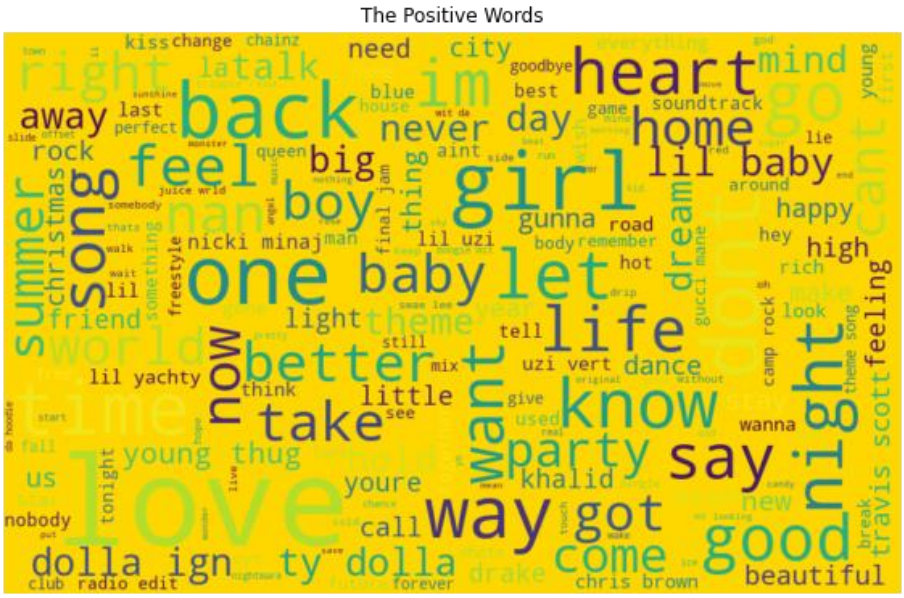

Figure – 3.13 (Positive words)

If these two clouds are compared, it can be said that the words they contain the most common words are similar. This shows that words take negative and positive values according to the sentence in which they are used.

At the end of these two comparisons, the dataset is brought into bag of words model. The bag-of-words model is a way of representing text data when modeling text with machine learning algorithms. It is simple to understand and implement and has seen great success in problems such as language modeling and document classification.

Before the creating model, the data must be clean. Therefore, a list is created for the words in the dataset and the words separated by their stems are added to this list. And this list is fitted using count vectorizer.

```
train_corpus = []

for i in range(0, 4429):
    songs = re.sub('[^a-zA-Z]', ' ', df['track_name'][i])
    songs = songs.lower()
    songs = songs.split()

    ps = PorterStemmer()

  # stemming
    songs = [ps.stem(word) for word in songs if not word in set(stopwords.words('english'))]

  # joining them back with space
    songs = ' '.join(songs)
    train_corpus.append(songs)
```

```
# creating bag of words
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(max_features = 500)
X = cv.fit_transform(train_corpus)
y = train['label'].values
```

Figure – 3.14 (Bag of Words)

The dataset created with the bag of words model is now ready to be separated into training and test data. Test size was determined as 30% of the dataset.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
```

```
print('Shape of X_train:', X_train.shape)
print('Shape of y_train:', y_train.shape)
print('Shape of X_test:', X_test.shape)
print('Shape of y_test:', y_test.shape)
```

```
Shape of X_train: (3100, 500)
Shape of y_train: (3100,)
Shape of X_test: (1329, 500)
Shape of y_test: (1329,)
```

Figure – 3.15 (Train-Test Split)

There is now a ready-made dataset for classification. Logistic regression, Random forest classifier, Decision Tree Classifier, Support Vector Machine, XGBoost Classifier and Multinomial Naive Bayes were used in this dataset for sentiment classification. Accuracy, f1_score, recall_score and precision_score was calculated separately for each model. Considering the results below, it can be said that all of the models actually give good results.

Model	Training_A	P	R	F	Validation_A
LogisticReq	0.90	0.87	0.59	0.63	0.89
RandomFor	0.95	0.79	0.71	0.74	0.90
DecisionTree	0.95	0.76	0.73	0.74	0.89
SVC	0.94	0.87	0.64	0.68	0.90
XGBClassifier	0.90	0.87	0.62	0.67	0.90
MultinomialNB	0.91	0.81	0.66	0.70	0.90

Figure – 3.16 (Score Table)

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, f1_score

#model tanımlanıyor
lr_model = LogisticRegression()

#model datayla fit ediliyor
lr_model = lr_model.fit(X_train, y_train)
#tahminler oluşturuluyor
y_pred_test = lr_model.predict(X_test)
y_pred_train = lr_model.predict(X_train)

print("Training Accuracy :", lr_model.score(X_train,y_train))
print("Validation Accuracy :", lr_model.score(X_test,y_test))

# calculating the f1 score for the validation set
print("F1 score :", f1_score(y_test, y_pred_test, average='macro'))
print("Recall score:", recall_score(y_test, y_pred_test, average='macro'))
print("Precision score:", precision_score(y_test, y_pred_test, average='macro'))
```

Figure – 3.17 (Logistic Regression)

```
from sklearn.ensemble import RandomForestClassifier
#model tanımlanıyor
rf_model = RandomForestClassifier()
#model datayla fit ediliyor
rf_model = rf_model.fit(X_train, y_train)
#tahminler oluşturuluyor
y_pred_test = rf_model.predict(X_test)
y_pred_train = rf_model.predict(X_train)

print("Training Accuracy :", rf_model.score(X_train,y_train))
print("Validation Accuracy :", rf_model.score(X_test,y_test))

# calculating the f1 score for the validation set
print("F1 score :", f1_score(y_test, y_pred_test, average='macro'))
print("Recall score:", recall_score(y_test, y_pred_test, average='macro'))
print("Precision score:", precision_score(y_test, y_pred_test, average='macro'))
```

Figure – 3.18 (Random Forest Classifier)

```python
from sklearn.tree import DecisionTreeClassifier
#model tanımlanıyor

dt_model = DecisionTreeClassifier()

#model datayla fit ediliyor
dt_model = dt_model.fit(X_train, y_train)
#tahminler oluşturuluyor
y_pred_test = dt_model.predict(X_test)
y_pred_train = dt_model.predict(X_train)

print("Training Accuracy :", dt_model.score(X_train,y_train))
print("Validation Accuracy :", dt_model.score(X_test,y_test))

# calculating the f1 score for the validation set
print("F1 score :", f1_score(y_test, y_pred_test, average='macro'))
print("Recall score:", recall_score(y_test, y_pred_test, average='macro'))
print("Precision score:", precision_score(y_test, y_pred_test, average='macro'))
```

Figure – 3.19 (Decision Tree Classifier)

```python
from sklearn.svm import SVC

svc_model = SVC()

#model datayla fit ediliyor
svc_model = svc_model.fit(X_train, y_train)
#tahminler oluşturuluyor
y_pred_test = svc_model.predict(X_test)
y_pred_train = svc_model.predict(X_train)

print("Training Accuracy :", svc_model.score(X_train,y_train))
print("Validation Accuracy :", svc_model.score(X_test,y_test))

# calculating the f1 score for the validation set
print("F1 score :", f1_score(y_test, y_pred_test, average='macro'))
print("Recall score:", recall_score(y_test, y_pred_test, average='macro'))
print("Precision score:", precision_score(y_test, y_pred_test, average='macro'))
```

Figure – 3.20 (Support Vector Classifier)

```python
from xgboost import XGBClassifier

xgb_model = XGBClassifier()

#model datayla fit ediliyor
xgb_model = xgb_model.fit(X_train, y_train)
#tahminler oluşturuluyor
y_pred_test = xgb_model.predict(X_test)
y_pred_train = xgb_model.predict(X_train)

print("Training Accuracy :", xgb_model.score(X_train,y_train))
print("Validation Accuracy :", xgb_model.score(X_test,y_test))

# calculating the f1 score for the validation set
print("F1 score :", f1_score(y_test, y_pred_test, average='macro'))
print("Recall score:", recall_score(y_test, y_pred_test, average='macro'))
print("Precision score:", precision_score(y_test, y_pred_test, average='macro'))
```

Figure – 3.21 (XGBoost Classifier)

```python
from sklearn.naive_bayes import MultinomialNB

nb_model = MultinomialNB()

#model datayla fit ediliyor
nb_model = nb_model.fit(X_train, y_train)
#tahminler oluşturuluyor
y_pred_test = nb_model.predict(X_test)
y_pred_train = nb_model.predict(X_train)

print("Training Accuracy :", nb_model.score(X_train,y_train))
print("Validation Accuracy :", nb_model.score(X_test,y_test))

# calculating the f1 score for the validation set
print("F1 score :", f1_score(y_test, y_pred_test, average='macro'))
print("Recall score:", recall_score(y_test, y_pred_test, average='macro'))
print("Precision score:", precision_score(y_test, y_pred_test, average='macro'))
```

Figure – 3.22 (Multinomial Naïve Bayes)

As a result, if the result ranges of each model are compared, it can be said that the Random Forest Classifier and Support Vector Classifier does better than other models in this data set.

Logistic Regression (LR)	0.87
Random Forest Classifier (RF)	0.88
Decision Tree Classifier (DT)	0.86
Support Vector Classifier (SVC)	0.88
XGB Classifier (XGB)	0.87
Multinomial Naïve Bayes (NB)	0.87

Figure – 3.23 (Average Results)

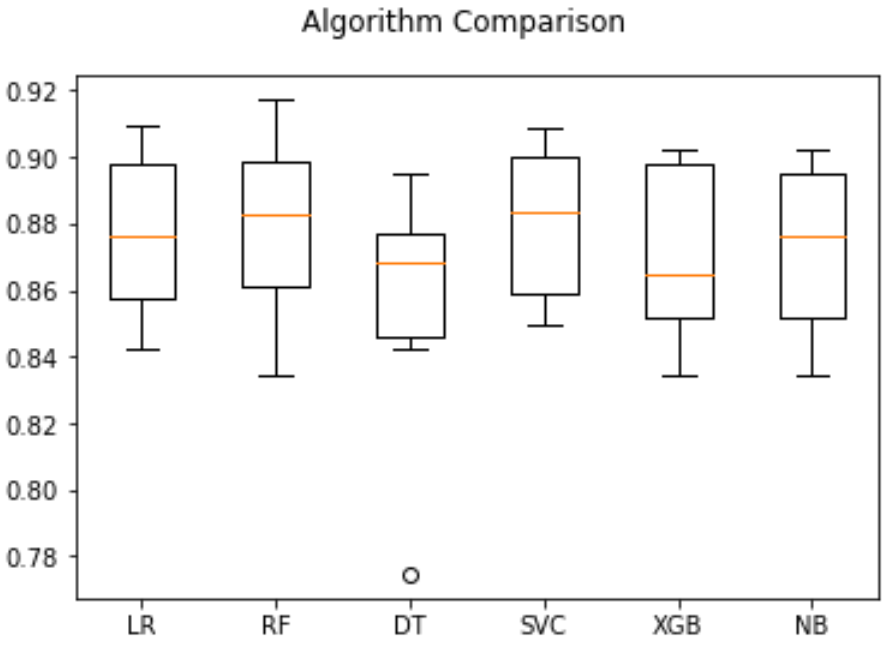

Figure – 3.24 (Algorithm Comparison)

4. CONCLUSION

In this paper, a sentiment analysis study was conducted using various machine learning techniques on a data set that includes song titles in Spotify user playlists. First, Vader Sentiment Analyzer and TextBlob Sentiment Analyzer models, which calculate sentiment polarity, were used to label data positive and negative. By comparing the labeled data, it was seen that Vader Sentiment Analyzer gave more suitable results for this data set. After labeling, the data was cleared for classification algorithms. Logistic regression, Random forest classifier, Decision Tree Classifier, Support Vector Machine, XGB Classifier and Multinomial Naive Bayes, which are frequently used classification algorithms in machine learning, were used; Performance comparison of the relevant classifiers was made according to the model performance criteria. When these performance criteria were examined, the highest values were obtained from Random Forest Classifier and Support Vector Classifier. In the training dataset, the Random Forest Classifier performed better than SVC (94%) with a 95% correct classification rate, while both classifiers performed almost the same with a correct classification rate of around 88% for the test data set.

With this project bringing results beyond expectations, we can say that, according to the song titles that the users listen to, song suggestions with similar feelings can be made. This can be used for different prediction algorithms used in music applications. For example, like the weekly playlist that the Spotify application creates by predicting the songs listened to.

5. REFERENCES

[1] Spotify Report. Retrieved from: https://investors.spotify.com/financials/

[2] Spotify Web API. Retrieved from: https://developer.spotify.com/

[3] Perkel, J. M. (2018). Why Jupyter is data scientists' computational notebook of choice. Nature, 563(7732), 145-147.

[4] Zelle, J. M. (2004). Python programming: an introduction to computer science. Franklin, Beedle & Associates, Inc..

[5] Baykara, M., & Gürtürk, U. (2017). Sosyal Medya Paylaşımlarının Duygu Analizi Yöntemiyle Sınıflandırılması, 2. International Conferance on Computer Science and Engineering, 911-916.

[6] Pang, Bo, Lillian Lee, and Shivakumar Vaithyanathan. "Thumbs up?: sentiment classification using machine learning techniques," Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10. Association for Computational Linguistics, 2002.

[7] Vader Sentiment Analysis Tool Retrieved from: https://pypi.org/project/vaderSentiment/

[8] TextBlob Sentiment Analysis Tool Retrieved from: https://textblob.readthedocs.io/en/dev/

[9] Joyce, J. (2003). Bayes' theorem.

[10] Menard, S. (2002). Applied logistic regression analysis (Vol. 106). Sage.

[11] Livingston, F. (2005). Implementation of Breiman's random forest machine learning algorithm. ECE591Q Machine Learning Journal Paper, 1-13.

[12] Brijain, M., Patel, R., Kushik, M. R., & Rana, K. (2014). A survey on decision tree algorithm for classification.

[13] Kotsiantis, S. B., Zaharakis, I., & Pintelas, P. (2007). Supervised machine learning: A review of classification techniques. Emerging artificial intelligence applications in computer engineering, 160(1), 3-24.

[14] Junker, M., Hoch, R., & Dengel, A. (1999, September). On the evaluation of document analysis components by recall, precision, and accuracy. In Proceedings of the Fifth

International Conference on Document Analysis and Recognition. ICDAR'99 (Cat. No. PR00318) (pp. 713-716). IEEE.

[15]Zhang, Y., Jin, R., & Zhou, Z. H. (2010). Understanding bag-of-words model: a statistical framework. International Journal of Machine Learning and Cybernetics, 1(1-4), 43-52.

[16] Spotipy library Retrieved from: https://spotipy.readthedocs.io/en/2.16.1/

[17] XGBoost Classifier Retrieved from: https://xgboost.readthedocs.io/en/latest/

6. CV

Fatma Akın was born on 17 January 1996 in Adana. She graduated from 75th Year Anatolian High School in 2014. She started her undergraduate education in the Computer Engineering Department of Çukurova University in 2015. The year she entered the university, she was entitled to benefit from the Snowdrops scholarship fund of the community volunteer's foundation.