

**UNIVERSITY OF HERTFORDSHIRE**

School of Computer Science

MSc Artificial Intelligence with Robotics

**Final Project Report**

7COM1040 - Computer Science Masters Project

8th June 2020

**Title**

Investigating Different Machine Learning Classifiers for Music Recommendation System

**Name:** Manan Chaudhry – 17029019

**Supervisor:** Deepak Panday

# Abstract

Machine learning is being implemented in every possible field today. This project will summarize how the recommender system works by applying machine learning techniques. Recommendation engine differs from platforms to platforms; mostly, this report will contrast how machine learning and deep learning techniques are applied to online streaming services such as Netflix, YouTube, Spotify, etc. To compete with the deep learning methods, I have used basic machine learning approaches and tuned them to get the best out of them for this particular domain. The primary basis of this project will be on Spotify and that datasets used in this project are extracted from Spotify’s Web API. This project compares works of different authors with different approaches for binary classification methods used for prediction between two classes. Classification methods such as Support Vector Machine, K-Nearest Neighbors, Random Forest are used on four datasets with different values, but they share similar feature attributes. Along with that, a simple Neural Network approach is also implemented and compared with the state-of-the-art machine learning techniques.

# MSc Final Project Declaration

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Artificial Intelligence with Robotics at the University of Hertfordshire (UH).

It is my own work except where indicated in the report.

I did not use human participants in my MSc Project.

I hereby give permission for the report to be made available on the university website provided the source is acknowledged.

Table of Contents

[Abstract 1](#_Toc42527015)

[MSc Final Project Declaration 2](#_Toc42527016)

[Chapter 1: Introduction 7](#_Toc42527017)

[1.1 Background 7](#_Toc42527018)

[1.2 Research Questions 8](#_Toc42527019)

[1.3 Motivation 8](#_Toc42527020)

[1.4 Research Aims and Objectives 8](#_Toc42527021)

[1.5 Project Plan 9](#_Toc42527022)

[1.6 Chapters Overview 9](#_Toc42527023)

[Chapter 2: Literature Review 10](#_Toc42527024)

[2.1 Recommendation Engine 11](#_Toc42527025)

[2.1.1 Collaborative Filtering 11](#_Toc42527026)

[2.1.2 Content-Based Filtering 13](#_Toc42527027)

[2.2 Modelling and Evaluation 14](#_Toc42527028)

[Chapter 3: Research Methodology 15](#_Toc42527029)

[3.1 Data collection: 16](#_Toc42527030)

[3.1.1 Authentication Process for Spotify: 17](#_Toc42527031)

[3.1.2 Audio Features Extraction 20](#_Toc42527032)

[3.2 Data Preparation 21](#_Toc42527033)

[3.2.1 Data Pre-Processing 21](#_Toc42527034)

[3.2.2 Data cleaning 22](#_Toc42527035)

[3.6 Audio Features 23](#_Toc42527036)

[3.6.1 Acousticness 23](#_Toc42527037)

[3.6.2 Danceability 24](#_Toc42527038)

[3.6.3 Duration 24](#_Toc42527039)

[3.6.4 Energy 24](#_Toc42527040)

[3.6.5 Instrumentalness 24](#_Toc42527041)

[3.6.6 Key 25](#_Toc42527042)

[3.6.7 Liveness 25](#_Toc42527043)

[3.6.8 Loudness 25](#_Toc42527044)

[3.6.9 Mode 25](#_Toc42527045)

[3.6.10 Speechiness 25](#_Toc42527046)

[3.6.11 Tempo 26](#_Toc42527047)

[3.6.12 Time Signature 26](#_Toc42527048)

[3.6.13 Valence 26](#_Toc42527049)

[3.6.14 Class 26](#_Toc42527050)

[3.7 Weka GUI – Data Mining Software 26](#_Toc42527051)

[3.8 Python and The Libraries Used 27](#_Toc42527052)

[3.9 Classification algorithms 27](#_Toc42527053)

[3.9.1 Naive Bayes 28](#_Toc42527054)

[3.9.2 Support Vector Machine 29](#_Toc42527055)

[3.9.3 Random Forest 30](#_Toc42527056)

[3.9.4 K-Nearest Neighbors 31](#_Toc42527057)

[3.9.5 Neural Networks 33](#_Toc42527058)

[Chapter 4: Results and Discussion 34](#_Toc42527059)

[4.1 Results Explanation 36](#_Toc42527060)

[4.1.1 SVM: 38](#_Toc42527061)

[4.1.2 Naive Bayes: 38](#_Toc42527062)

[4.1.3 KNN: 39](#_Toc42527063)

[4.1.4 Random Forest: 39](#_Toc42527064)

[4.1.5 NN: 40](#_Toc42527065)

[4.2 Testing: 43](#_Toc42527066)

[Chapter 5: Conclusion and Future Work 44](#_Toc42527067)

[References 46](#_Toc42527068)

[Appendices: 51](#_Toc42527069)

[Following is the code for the data retrieval. 51](#_Toc42527070)

[To remove duplicate values from our dataset: 53](#_Toc42527071)

[Code for plotting the Audio Features: 53](#_Toc42527072)

[Following are the plot: 56](#_Toc42527073)

[The following section is for the codes for all the classification methods used in the project. 62](#_Toc42527074)

[SVM 62](#_Toc42527075)

[Dataset 1: 62](#_Toc42527076)

[Dataset 2: 65](#_Toc42527077)

[Dataset 3: 65](#_Toc42527078)

[Dataset 4: 66](#_Toc42527079)

[Naïve Bayes: 67](#_Toc42527080)

[Dataset 1: 67](#_Toc42527081)

[Dataset 2: 69](#_Toc42527082)

[Dataset 3: 70](#_Toc42527083)

[Dataset 4: 72](#_Toc42527084)

[KNN: 74](#_Toc42527085)

[Dataset 1: 74](#_Toc42527086)

[Dataset 2: 75](#_Toc42527087)

[Dataset 3: 77](#_Toc42527088)

[Dataset 4: 78](#_Toc42527089)

[For Random Forest: 79](#_Toc42527090)

[Dataset 1: 79](#_Toc42527091)

[Dataset 2: 81](#_Toc42527092)

[Dataset 3: 82](#_Toc42527093)

[Dataset 4: 83](#_Toc42527094)

[Neural Network: 84](#_Toc42527095)

[Dataset 2: 87](#_Toc42527096)

[Dataset 3: 90](#_Toc42527097)

[Dataset 4: 93](#_Toc42527098)

[Screenshot from Weka with the testing data: 96](#_Toc42527099)

[For Test Data 1: 96](#_Toc42527100)

[For Test Data 2: 96](#_Toc42527101)

**List of Tables:**

[Table 1: Accuracy table 37](#_Toc42526673)

[Table 2: SVM Confusion Matrice for all the datasets 38](#_Toc42526674)

[Table 3: Naïve Bayes Confusion Matrices for all the datasets. 39](#_Toc42526675)

[Table 4: KNN Confusion Matrices for all the datasets 39](#_Toc42526676)

[Table 5: Confusion Matrices for Randon Forest 40](#_Toc42526677)

[Table 6 Confussion Matrix obtained from Test Datasets 43](#_Toc42526678)

**List of Figures:**

[Figure 1: Image Source: (Johnson, 2014) 12](#_Toc42526796)

[Figure 2: Sums up the recommender system.. 14](#_Toc42526797)

[Figure 3: Spotify’s Web API to get Audio Features 17](#_Toc42526798)

[Figure 4 Browser console for data extraction 18](#_Toc42526799)

[Figure 5 Dashboard for Spotify for Developers 19](#_Toc42526800)

[Figure 6 Example of what an app looks like in Spotify for Developers 19](#_Toc42526801)

[Figure 7 Features extraction console 20](#_Toc42526802)

[Figure 8 Track IDs Extraction 22](#_Toc42526803)

[Figure 9 CSV file format change 23](#_Toc42526804)

[Figure 10 Source: (Ray, 2017) 29](#_Toc42526805)

[Figure 11 Source: (Yiu, 2019) 31](#_Toc42526806)

[Figure 12 (Navlani, 2018) 32](#_Toc42526807)

[Figure 13 Neural Network connections 33](#_Toc42526808)

[Figure 14 Data Science Life Cycle 34](#_Toc42526809)

[Figure 15 Confusion Matrix 35](#_Toc42526810)

[Figure 16: Bar plot for accuracies. The x-axis represents the classifiers, and the y-axis represents the accuracies. Four bars in each classifier represents datasets from 1 to 4, respectively. 36](#_Toc42526811)

[Figure 17: Neural Networks Structure 42](#_Toc42526812)

# Chapter 1: Introduction

## 1.1 Background

Machine learning has been there in existence long before the internet, and even when the internet came into our lives, it didn’t take us that long to perfect it. Taking actions after perceiving the environment with the help of a program was just called being a computer, but when associating the name “machine learning” with it, it changes the whole perspective of a computer. Machine learning technologies are being developed for a very long time, and now is the time more than ever to integrate those technologies with the internet. Some would say that implementing artificial intelligence in our day to day life would make a human being lazy, which is not false, but if seen differently, it is just to make our lives better and easier than ever before.

Today, the internet provides access to mostly everything you could name and one of the main things involve shopping, watching movies, videos or listening to songs on-the-go wherever you want and whenever you want. Before the popularity of internet browsing, if you wanted to buy something, you would have to go to ten different shops and then decide on a single thing, but now you can just browse and shop all under a single website called Amazon.com. All these services are made possible by the online streaming platforms like Netflix.com, which is famous for watching movies, YouTube.com for watching a plethora of videos and Spotify.com for listening to music. In one way, you can say that these services are migrated online for our ease, and it takes a lot of technological power to make them work correctly for us.

Now everyone is moving online to the digital world, and millions of users surf on at least one of these platforms daily. For a user to stay interactive online on these platforms and spend more time on it, the user feed should show more items that they are more likely to engage with. This technology we are talking about is referred to as a  **Recommendation System.** This research will focus on how machine learning technologies have changed the recommendation system, and I’m mainly going to shed light on **music playlist generation** based on user’s liked and disliked feed.

To get a gist of it, this is how a recommendation system works: if you are buying a printer from Amazon, their recommendation algorithm would suggest you to buy the A4 size papers to go with it. Similarly, the same goes for online streaming platforms like YouTube and Spotify as they have millions of items to present, but a recommendation algorithm would recommend videos or songs that the particular user would like. These recommendations don’t only use a single method, but it involves various techniques, and they keep getting better with time as more data is fed to the system.

## 1.2 Research Questions

This report aims to centric around the below-mentioned areas:

1. How well do state-of-the-art machine learning techniques perform for a music recommendation system based on the user’s listening history?
2. If they are performing well, is there any room for improvement in the existing algorithms to make it perform better for the given domain of the dataset?

## 1.3 Motivation

What made me motivated towards working with this research topic is the Spotify’s infamous “Discover Weekly” playlist (Spotify, 2020). One of the main reasons is that this approach by Spotify is considerably new in the market, and not a lot of people have done their research and thesis on it as it was debuted in early 2015 (Ciocca, 2017). This feature generates a unique playlist for the user every week with 30 songs to which a user had never listened. The claim behind here is that the playlist is altered based on the user’s listening history; the website says, “To generate your Discover Weekly playlist - we need to get to know your tastes for a few weeks.”. From my personal experience, since I’m a frequent Spotify user, I enjoy the whole playlist. To put contrast, I sometimes end up liking 70 percentage, or more of the playlist, which is in terms of machine learning translates to have 70% accuracy of the model targeting the particular customer. The central recommendation system is generated by Echo Nest’s technology. In March 2014, Spotify acquired Echo Nest ( Brian, et al., 2014), so this is a relatively new topic for people to do their research.

## 1.4 Research Aims and Objectives

My research aim involved mainly investigating how the music playlist generation works. YouTube would recommend videos, Netflix would recommend new or old movies, and Spotify would suggest new songs for the listening session. While all these platforms are different, they all share the same agenda; improve the quality of user’s time spent on their platform. I needed to know whether they use the same technique but different approaches? My objective includes but not limited to, investigating different classifier algorithms and see which of them would perform the best for the domain of my dataset.

## 1.5 Project Plan

Project planning goes as follow:

* Deciding on the main project area on work.
* Consulting with the supervisor and finalizing the topic.
* Find relevant articles and papers on the topic area.
* Find an appropriate dataset.
* Decide on what all programming languages and software to work with.
* Did research on a new module: Data Mining to help me with my project.
* Followed numerous online articles and video playlists to learn in more depth on how to tweak the algorithms to get the most out of them in different scenarios.
* Roughly run some models to see if it is worth going forward with this database or not.
* Compare the results with other sources online.
* Extracted my own data.
* Did heavy calculations, tests and runs on the datasets to find optimal solutions.

## 1.6 Chapters Overview

The chapters in this are organized as follows:

1. Introduction: This chapter highlights what were the motivations behind taking this project and the background related to the topic.
2. Literature Review: In this chapter, I will mention the studies related to my project topic area and in what context they are similar. This section will also show some light on what are the methods of various online streaming platforms use for the recommendation process.
3. Research Methodology: This chapter will cover the extraction of the data, data pre-processing, data preparation and the machine learning classification involved in this project. Along with that, the features of the dataset are explained.
4. Results and Discussion: In this chapter, I will discuss the results obtained and explain what were my ways to bend the models to perform better.
5. Conclusion and Future Work: The last chapter would conclude what has been achieved and what more work can be included to make this research more appropriate and whether my research questions are answered or not.

# Chapter 2: Literature Review

Mostly every platform uses recommendation engines in one form or another. Examples: Facebook suggests - “People You May Know,” Netflix - “Other Movies You May Enjoy,” LinkedIn – “Jobs You May Be Interested In,” Amazon – “Customer who bought this item also bought,” YouTube – “Recommended Videos,” Google – “Search results adjusted,” Pinterest – “Recommended Images” (Minds, 2019). This report is mainly concerned with how the recommendation system works in the music industry. Recommendation technique has always been there, no matter if we talk about it being online or offline. By offline, I mean, when I’m out there in the market to buy a vinyl record, the sales guy would recommend what I can go with based on my taste. This is just a hypothetical situation, but imagine millions of users come and ask for a recommendation at the same time; it would be impossible for the single salesperson to guide. Around the early 2000s, teams of music experts used to curate playlists by hand, and the songs in the playlists would be the music they thought sounded good (Giacaglia, 2019). After some time, Pandora (Pandora, 2020) started a different approach where the music experts would analyze every song and tag them with descriptions like ‘folk,’ ‘slow,’ ‘rap,’ or ‘love’; that would create playlists of songs with similar tags. But one of the major drawbacks in these approaches was they were never centric towards individual user’s taste in music.

Fast-forwarding to today, now we have machine learning making music for us. An example of that is Google’s NSynth Super (NSynthSuper, 2020), which is an ongoing experiment developed by Google, which will help artists create new music. Apple Music and Amazon Prime, which are one of the most used music streaming platforms out there, started to use machine learning techniques for curating playlists, but the users would typically get playlists based on the artists and albums they play (Popper, 2015). Machine learning technology has come a long way; a post from 2014 says, “Why we crave human-curated playlists.” (Fowler, 2014). This article talks about that simple algorithms are not providing satisfying curated playlists, and algorithms can’t alone predict what a human would want. Natural Language Processing (NLP) has changed the way how we look at playlist generation now. NLP used by Spotify determines which articles and languages are used alongside which artists and songs are being discussed. Spotify saw an increase in subscribers from 52 million to 113 million in just under the last three years (Sharma, 2019), which shows if you are providing a good service, people will pay for the service. Well, if you think your music could miss out on an audience, (Sidekick, 2018) is a website where you can submit your music to playlist curators for free who will listen and see if your track is worth adding to their playlist or not. These are the playlists that have hundreds and thousands of followers, and the playlist gets updated every now and then.

## 2.1 Recommendation Engine

Recommendation system algorithms are usually divided into two categories, Collaborative Filtering and Content-Based Filtering. They both differ in the type of dataset and the domain of the project. Collaborative Filtering approach is based on feedback from users to items, and Content-Based Filtering is entirely based on characteristic features of the items. In this report, we will be computing the predictions constructed using Content-Based Filtering.

### 2.1.1 Collaborative Filtering

As the name suggests, Collaborative Filtering is a recommender system based on the combination of what a user did with other users. Collaborative Filtering (WIki, 2020) uses a similarity function between two objects. The mathematical formulas could be found on (Minds, 2019). They are built on datasets that are collected from feedback from users and items. In our case, items are songs. Examples for feedback could be star ratings for movies in Netflix, liked or disliked or skipped songs in Spotify, etc. Collaborative Filtering is further classified into different models, which are:

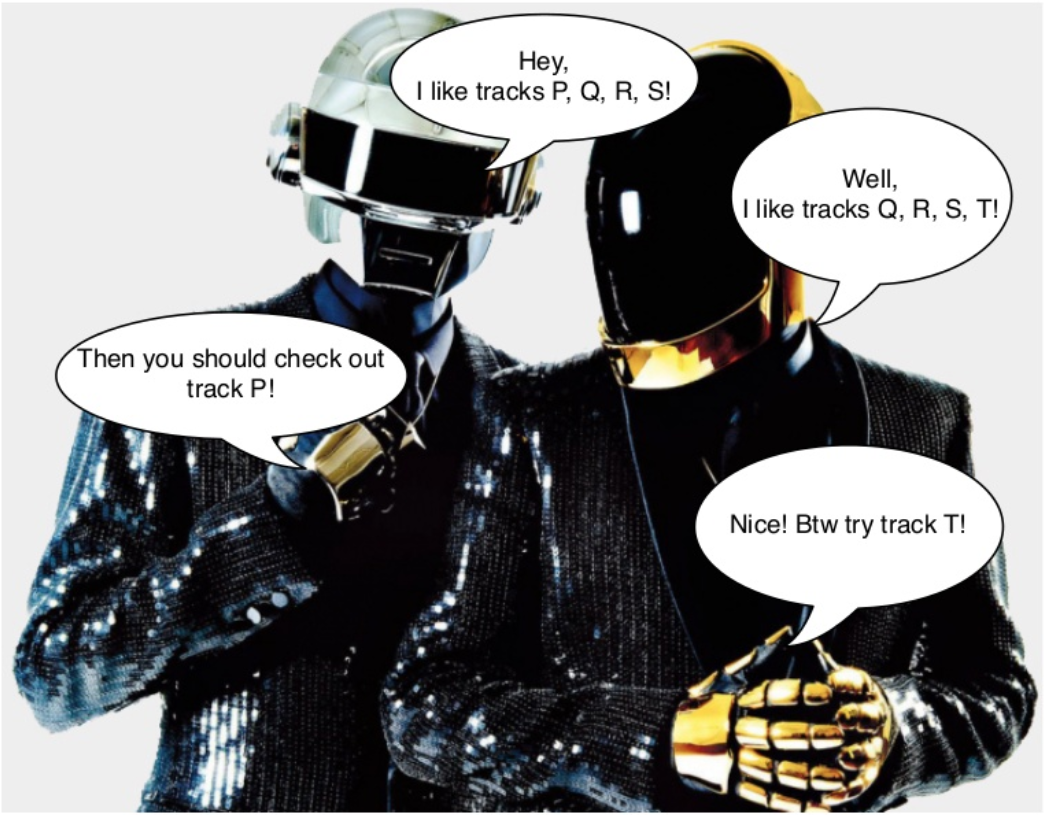
1. User-User Collaborative Filtering: This method determines how other users are similar to a user and then predict if the other user might like the item (song) or not. One drawback of this method could be that generally, over some time, a user’s taste might change. An example of how this works; if a user A likes songs A, B & C and user B like songs B, C & D, so the method would suggest User A that they should listen to song D and the same goes for User B. Illustrated in the figure below.  
     
   

Figure 1: Image Source: (Johnson, 2014)

1. Item-Item Collaborative Filtering: Same as the first one, this one is self-explanatory. This method could be defined as if a user likes an item; the method would suggest a song using the similarity function. For example, if a user likes a song by an artist A, the algorithm would suggest the user a similar song from the same artist A.
2. User-Item Collaborative Filtering: This method is a cost-insensitive approach which combines both the above processes and generates recommendations. This method generally gathers users and items which share the same characteristics and group them under a much smaller matrix. One of the major drawbacks of this method is it need to re-learn everything every time a new user or a new item or new rating enters the game, which makes it cost-insensitive. This is, by far, one of the most used and favourable approaches out there for most of the online streaming platforms.

### 2.1.2 Content-Based Filtering

One major point which I haven’t talked about yet is the Cold-Start problem. All the methods mentioned above uses techniques to form a recommendation engine from either the existing users/customers or items or their combination. But what happens when a new user subscribes to a service? How does a recommender system suggest items to the user when there’s nothing to compare with? This is called a Cold-Start problem, and this is where Content-Based Filtering comes to mitigate the issue, not entirely, of course. When a new user signs up to the service, the platform would ask questions prior to going straight to the home page. In an example for an online movie based streaming platform, a user might be asked which genres of movies they are into and further which movies they would prefer to watch. An example of a music streaming platform, a user might be asked which artists they would prefer to listen to. Upon receiving these details, the information is then fed to the algorithms, and they produce recommendations.

Even though my practice would be accounted under Item-Item Collaborative Filtering as I’m proposing a recommendation engine based on user’s likes and dislikes history, but it would require metadata of several users. Whereas in this report, classifications would be done on a single user’s data, which sits perfectly under the Content-Based Filtering. As explained in (One, 2016), for each item in the dataset, there is an item profile, and the profile is a set of features. In my data, the item is a song file from Spotify, and the elements in the item profile are the Audio Features (explained in the Results section).

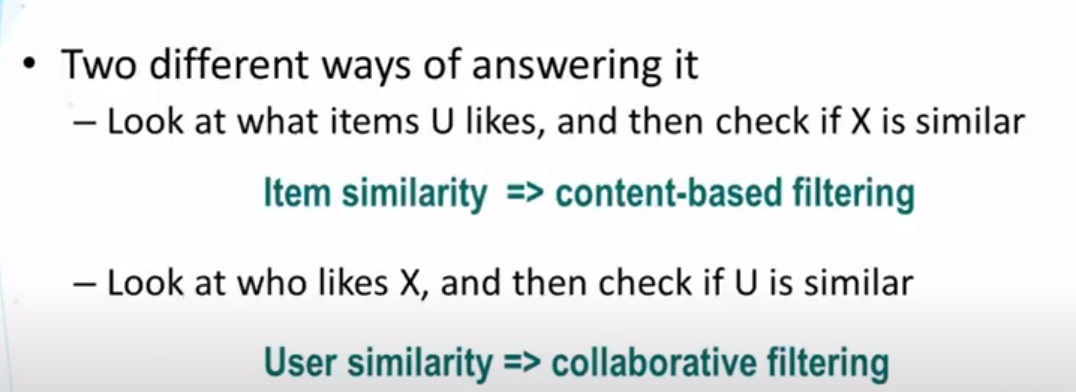


Figure 2: Sums up the recommender system..

## 2.2 Modelling and Evaluation

The recommendation system has been a popular topic for developers and researchers to work and find new and different ways of achieving higher accuracy. But accuracy doesn’t always imply the likeness of the predicted item. (Pauws, et al., 2008) have applied heuristics, (Hariri, et al., 2012) have used latent topics, (Knees, et al., 2006) could be the earliest researches on automatic music playlist generation combining audio-based similarity with web-based data. (Jannach, et al., 2015) have used another interesting approach, which includes music playlist continuation that’s coherent with the most recently played tracks.

As discussed in section 2.1.2 about the cold start problem, most of the Collaborative Filtering approaches fail to solve the cold start problem and (Oord, et al., 2013) have shed light on eliminating this problem with deep convolutional neural networks using the Million Song Dataset (Bertin-Mahnieux, et al., 2011). However, a Cold-Start problem hasn’t been totally eliminated, but Content-Based Filtering still handles this problem better than the Collaborative Filtering method. Either way, the amount of seed input for the Contend-Based process is comparatively minor than the other approaches. All the researches are based on either only liked songs or skips or cold start, but this report would primarily base the work on a user’s liked and disliked songs in Collaborative Filtering styles with a Content-Based Filtering approach.

Most of the researches are either done on huge metadata with some seed songs as an input for the recommendation engine or done by conducting surveys that involved software development as well. My aim for this project is to feed data to my model with the items that I like and the items I dislike, which in turn would generate a model that will outperform the models which explicitly only take liked items as an input. Research by (Lad, 2017) has practised a similar approach to what I have applied in this project. (Lad, 2017) has labelled their class as 1 for good, 0 for bad and applied classification algorithms like Naïve Bayes, Decision Tree, Random Forest and Artificial Neural Network with accuracy scores 0.58, 0.71, 0.73 and 0.76 respectively. Another interesting approach I was very inspired by is shown in this article (McIntire, 2017), who has implemented supervised classifiers like Logistic Regression with 60% accuracy, K-Nearest Neighbor with upper 50’s, by Decision Tree around 69%, Random Forest and with the best performing model for them was Gradient Boosting Classifier with a learning rate of 0.1and max depth of 7 yielding 0.693 score on cross-validation. Not really a classification is mainly done to support a recommendation system, but (Santos, 2017) has published an article on Towards Data Science (Santos, 2017) where he has applied a machine learning approach (using Logistic Regression and achieving 82% accuracy) to distinguish whether a belongs to his playlist or his friend’s playlist which would help us to implement our proposed algorithms to compare how well they are working. (Georgieva, et al., 2018) have computed the prediction of hit songs labelling 1 or 0; indicating the songs featured in the Billboard Hot 100 (between 1991-2010) and o indicating otherwise. In this paper, they used Logistic Regression, GDA (Gaussian Discriminant Analysis), SVM (w/ Bagging), Decision Tree and Neural Network, getting the accuracies of 73%, 73%, 72%, 100% and 77% respectively. Here, Decision Tree seems to be topping the chart, but it had overfitted the data and scored 69% on the validation set.

The following chapter would discuss about the dataset I have used, including the datasets I have obtained from different sources.

# Chapter 3: Research Methodology

In this section, I will be covering the details about the data such as data collection, data pre-processing, data cleaning, data preparation and feature extraction. Along with that, this section will also cover the details about the machine learning classifiers used in this report, which include **Naïve Bayes, SVM (Support Vector Machine), Random Forest, KNN (K-Nearest Neighbors) and Neural Networks.**

For this project, I’m using Spotify’s Web API (Spotify, 2020) to extract the data, and my dataset would consist of Audio Features (Spotify, 2020) on which my classifier evaluation is done. Along with this dataset, I have collected two test datasets which the model had never seen. As mentioned in the Literature Review section, I have also procured two more datasets from people with a similar type of research like mine. The classification would be done on all three training sets as a whole, and the two test sets would be evaluated against the training dataset that I have extracted.

## 3.1 Data collection:

The data collection process has been divided into different sections. My main motive is to extract Audio Features for the tracks that I like and the tracks that I don’t like. The initial data on which my models are computed are the songs that I already have on my different liked playlists on Spotify. The data would be comprised of around north to 600 songs Audio Features for the liked songs and another around 600 songs that I dislike. The second dataset, which is actually the test dataset consists of 90 items in total for the liked and disliked songs. This test data was collected over a period of three weeks from three consecutive ‘Discover Weekly’ playlists. During my evaluation, I found out that my models weren’t achieving a good accuracy score on this test data, so I decided to extract another test data of 36 songs in total. This second test dataset is from my Release Radar (Spotify, 2016), which is a playlist made by Spotify with all the new releases from the artists you follow and more discoveries based on the user’s listening habits. I chose this playlist as a test set because this would give a broader range of music selection to choose from, and again, this playlist has the same labelled class variable with yes and no for the items. To expand my horizon and removing bias from my personal curated dataset, I have procured three more datasets (as discussed in section 2.2), one of them was available on Kaggle (McIntire, 2017), which has 2017 data points with 16 attribute features (same as mine). Whereas the second dataset from (Santos, 2017) wasn’t available online to download, so I found his contact details and asked him to provide me with his data. I have referenced his work, but I won’t be able to make the dataset available to anyone without his permission. The third dataset was taken from (Georgieva, et al., 2018), link is available in the paper. I will be using all dataset just for the comparison purpose, and I will not manipulate or misuse the data in any way.

Moving on the extracting my own data, I have used Spotify’s Web API (Spotify, 2020), and before we go and retrieve any data, Spotify needs us to authenticate against their API, which is just a standard protocol to monitor the amount of data coming and going. The following steps will walk you through the process involved in getting the data.

### 3.1.1 Authentication Process for Spotify:

Now, previously Spotify had the API to only get the Audio Features for a single track at a time, but luckily, Spotify’s API lets us collect Audio Features for Several Tracks now.

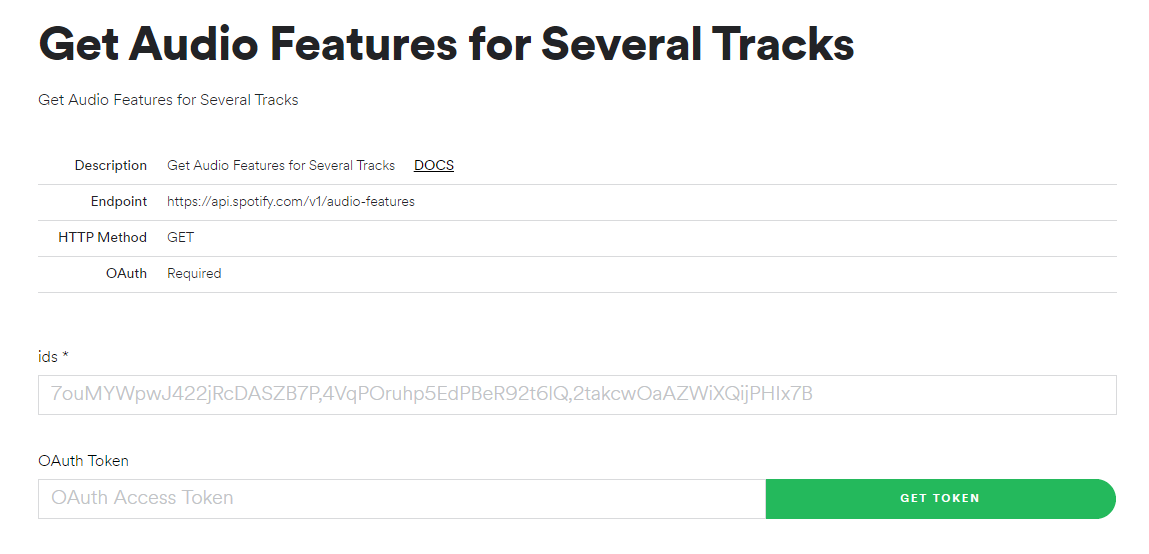


Figure 3: Spotify’s Web API to get Audio Features

The inputs in the above figure are ids and OAuth Token, which are for Track IDs and Authentication code, respectively. Every song on Spotify has a unique Track ID, and the OAuth Access token will be unique for every customer. The ids parameter is only limited to 100 Track IDs at a time, inputted as a comma-separated list of Track IDs. Exportify (Watson, 2020) is an application made available to everyone to export Spotify playlists. First, the user has to log in and authenticate the token needed by Exportify to get access to the user’s playlists. A user can download the playlists’ data they have on Spotify, and a CSV file with Track IDs for all the songs in the respective playlist and other information (which we don’t need for this project) is downloaded on the target device. The process will get very long if done manually on the API because the result will be displayed on the console of the browser (as shown in Figure 3.2), and it’s not really user friendly to extract the data into an excel file. In this project, I have used Spotipy (Lamere, 2014) is a lightweight Python library for the Spotify Web API, which made it easy with a minimal amount of coding to get collect the data from the API.

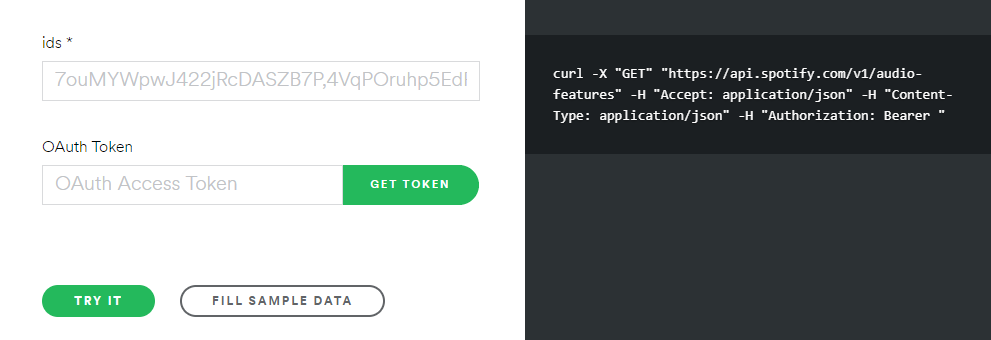


Figure 4 Browser console for data extraction

For the extraction process of Audio Features form Spotify’s Web API, (Gelo, 2018) has explained in his Jupyter Notebook with Python on how to retrieve the Audio Features using Spotipy (Spotipy, 2014) and getting any further with it, the process requires a Client ID and a Client Secret and (Nykowska, 2019) shows the steps to get the necessary. To get a client ID and the Client Secret, you need to log in to Spotify for Developers (Spotify, 2020) and click on ‘Create an App’ as shown in figure (5).

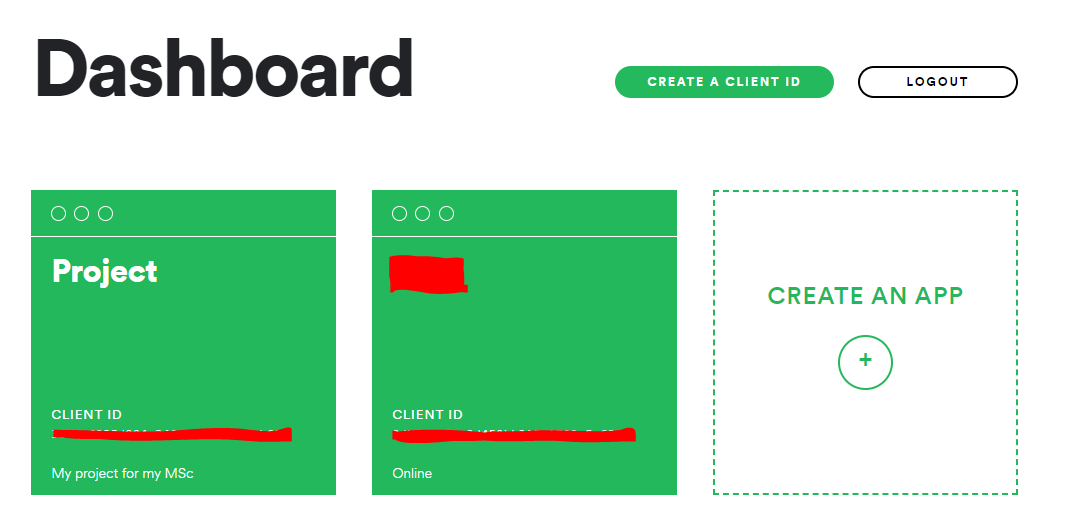


Figure 5 Dashboard for Spotify for Developers

After creating the app as (Nykowska, 2019) has explained, you should be able to see the Client ID and Client Secret, without which this project could not move forward. Make a note of the Client ID and the Client Secret, advised to copy and paste on a notepad as it would a hectic job to type both the 32-digit long codes.

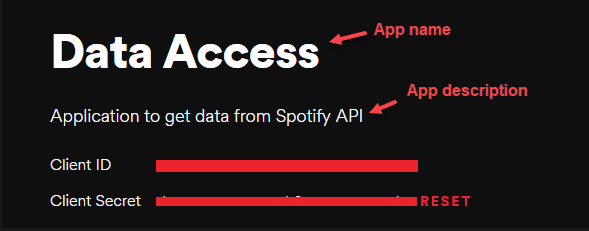


Figure 6 Example of what an app looks like in Spotify for Developers

### 3.1.2 Audio Features Extraction

Moving forward with the Audio Features extraction, Spotify provides audio features for every song in their database, and since not every song is similar, the feature values would be different for each individual song that would give this project a sense of fairness. Some of the features are Danceability, energy, loudness, tempo, etc. as shown in the figure (7).

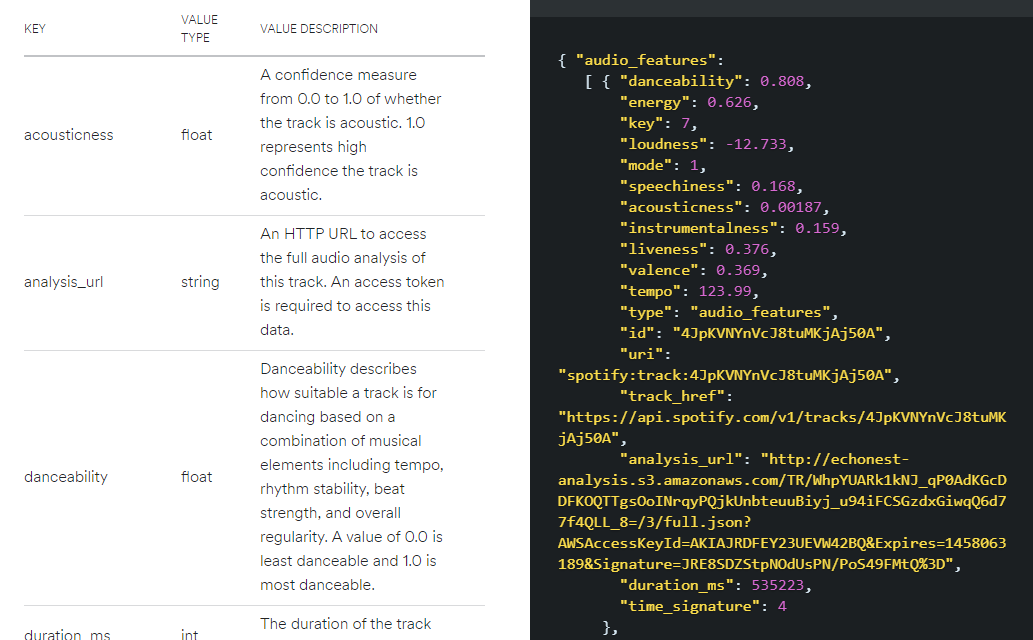


Figure 7 Features extraction console

The following python libraries are used to extract the Audio Features:

* **Spotipy**: A light python library used for data extraction from Spotify’s Web API.
* **Pandas**: Pandas is also an open-source library used for data structures and data analysis integrated with the python programming language.

The code for the Audio Features collection is provided in the Appendix. The input to the Jupyter Notebook is a comma-separated file that will have the initial data columns as Artists Name, Track Name, Track ID and Track Duration in ms. The output file will also be a CSV file with all the columns same as the input, including the audio features, as mentioned above. You can either initialize the Client ID, Client secret at the beginning of the program, or you can modify the OAuth2 script in Spotipy’s library to keep your credentials private.

## 3.2 Data Preparation

To keep it fair, the data points in extracted datasets are kept in equal numbers of liked songs and disliked songs to give it more cost-sensitive analysis. Data preparation can comprise of data pre-processing and data cleaning. The final attributes that I have used in my dataset are Artist Name, Track Name, Accousticness, Danceability, Track Duration, Energy, Instrumentalness, Key, Liveness, Loudness, Mode, Speechiness, Tempo, Time Signature, Valence and the last column field ‘Class’ is added manually after the extraction of the data depending on the domain of the research. A more detailed explanation is done in section 3.6.

### 3.2.1 Data Pre-Processing

This step is necessary to avoid the data producing misleading results. Since I have extracted the data from my playlists, there are chances of lots of similar songs to be in other playlists as well. This step is done after all the Audio Features extraction has been done for both the liked and disliked songs. Once the data is extracted, I then combined both the CSV files and removed the duplicate data points using the Pandas library for Python and the code for this step is provided in the Appendix.

After downloading the Track IDs from Exportify, the Track IDs are given in the format, as shown in the figure (8). The ID which Spotipy takes as an input is marked in the figure. A simple trick in excel will do the job, that is, ‘find and replace’ method. Finding all the entries with “spotify:track:” and leaving the ‘replace’ area blank would give a list of Track IDs we need for the project.

****

Figure 8 Track IDs Extraction

### 3.2.2 Data cleaning

The downloaded file from Exportify comes with some field columns that we don’t need for this project, such as “Added by” (added by which user) and “Added at” (the time when the song was added in the playlist). When the final CSV file is ready, we then remove the Track ID column, since they just correspond to their identity in Spotify’s database, it doesn’t reflect any meaning to the real data.

There were several problems encountered while importing the CSV to the Arff file. Arff is the file extension format for data files used in Weka, the software used for classification in this project (explained further in section 3.7). One of the problems I encountered was for the data fields having single apostrophes in them. According to Weka, if the data field has spaces in it, they need to be enclosed by single quotes. If your data has an apostrophe and then followed by more characters, it would produce an error. To resolve that issue, I just ultimately got rid of the single quotes and double quotes as they only represented the name of the song, which would still imply the same meaning to the program if quotes are removed.

I encountered another issue while loading the CSV data file to the Weka Explorer, which threw a “java.io.IOE.exception” exception. The problem occurred because of the CSV file format. There are two types of CSV file formats you can save (as shown in the figure (**9**)) as one is the standard comma-delimited, which everyone usually saves in, and the other one is CSV UTF-8 format. The Weka IDE I use accepts UTF-8 encoded data, so changing the format did the trick.

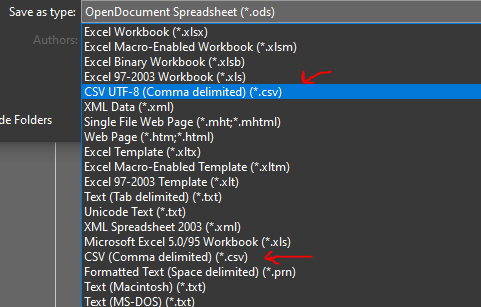


Figure 9 CSV file format change

## 3.6 Audio Features

Here I’m going to discuss the features I have used in my dataset. Keep in mind that there are some features that I haven’t used in my report (mentioned in section 3.2), which I am not going to cover here. All the generic details about the Audio Features could be found on Spotify’s website (Spotify, 2020). A bit more insight between the liked and disliked songs is represented through bar plots, which can be found in the Appendix. The mean value for only the liked items is explained under all the respective features.

### 3.6.1 Acousticness

**“**A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 being the highest acoustic measure.”

Mean: 0.159 explains that my music taste is the least towards acoustic; less guitar, piano or orchestras sounds.

### 3.6.2 Danceability

**“**Danceability describes how suitable a track is for dancing based on a combination of musical elements, including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable, and 1.0 is most danceable.”

Mean: 0.672 tells that a good part of my music selection has a danceable quality to it.

### 3.6.3 Duration

“The duration of the track in milliseconds.”

Mean: 202549.38 is an interesting insight which I didn’t know. This tells that the average music length I like is around 3.37 minutes.

### 3.6.4 Energy

“Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.”

Mean: 0.677; it looks like my music taste revolves around a good energetic vibe. If seen in the Appendix for the bar plot, the majority of the data points lie around the top end of the spectrum.

### 3.6.5 Instrumentalness

“Predicts whether a track contains no vocals. “Ooh” and “aah” sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly “vocal.” The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.”

Mean: 0.063 is a deficient measure, which is correct as most of my music is rap or pop-rap.

### 3.6.6 Key

“The estimated overall key of the track. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on. If no key was detected, the value is -1.”

Mean: 5.48. The key value ranges from 0 to 11, and little research online tells the higher the key value, the more the bassline is involved.

### 3.6.7 Liveness

“Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides a strong likelihood that the track is live.”

Mean: 0.19 says that it is highly unlikely for me to listen to a live-recorded performance.

### 3.6.8 Loudness

“The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing the relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typically range between -60 and 0 dB.”

Mean: -6.538. Less to my surprise, my music would be counted under loud music.

### 3.6.9 Mode

“Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1, and minor is 0.”

Mean: could either be 1 or 0, and my taste is equally divided, which interprets both western music collection and melody type songs.

### 3.6.10 Speechiness

“Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audiobook, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.”

Mean: 0.134. I do love trap music (which is less talking and more music) on certain occasions; the score justifies the result.

### 3.6.11 Tempo

“The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, the tempo is the speed or pace of a given piece and derives directly from the average beat duration.”

Mean: 126.11 would be considered as a medium-high tempo rate for an average listener.

### 3.6.12 Time Signature

“An estimated overall time signature of a track. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).”

Mean: 4. As the description suggests, I have got a relatively high number of values here.

### 3.6.13 Valence

“A measure from 0.0 to 1.0 is describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).”

Mean: 0.4. To justify this number, I would say my songs are a bit on an aggressive side.

### 3.6.14 Class

Class is a manually created attribute that denotes the songs that I like as ‘yes’ and ‘no’ for the songs I dislike.

## 3.7 Weka GUI – Data Mining Software

Weka is a collection of machine learning algorithms for data mining tasks (Weka, 2020). What makes Weka stand out in the crowd is it’s GUI (Graphical User Interface) which makes implementing all sorts of algorithms easy. Weka can be used for classification, regression, clustering, association rules, and visualization. I have used Weka to evaluate the models that I built in Python against the two test sets. Weka has many more functions, and anyone with no programming background could use it. I have also used Weka to visualize the ROC curves obtained from the best performing model in this project.

## 3.8 Python and The Libraries Used

For this project, I have used Jupyter Notebook as a Python IDE with Python version 3.8**.** Following are the Python libraries I have used in my programs:

* Pandas: Pandas is an open-source library for fast and powerful data analysis and manipulation.
* Numpy: Numpy is a python library that has support for large, multidimensional arrays and matrices, along with a large collection of mathematical functions to apply on these arrays.
* Seaborn: Seaborn is also a Python library for data visualization, which provides a high-level interface for drawing statistical graphics.
* Matplotlib: It provides an object-oriented API for embedding plots into applications.
* TensorFlow: Is a free software library for dataflow and differentiable programming. They are also used for math library and machine learning applications.
* Keras: Capable of running on top of TensorFlow. Designed to enable fast experimentation with deep neural networks.
* ScikitLearn: A free software machine learning library widely used for various classification and regression and clustering, etc., features.

## 3.9 Classification algorithms

This section gives a brief introduction to the state-of-the-art classification algorithms used in this project. I will only be explaining the methods in this section. Their evaluation would be done in the Results and Discuss section (Chapter 4) and further details on how the hyperparameters were manipulated to get the best results out of them for our particular domain of the dataset. Since my problem is a binary classification, all the methods are chosen by keeping that in mind.

### 3.9.1 Naive Bayes

Naïve Bayes is one of the most widely used classification technique which is based on Bayes’ Theorem. What makes this a powerful method, because it works extremely fast as compared to the other classification methods. In the domain of streaming platforms, unlike Netflix (which is an online movie streaming service), a music streaming service (like Spotify or Pandora) would have at least ten times more items to process. This classifier would assume that the presence of a particular attribute in a class is unrelated to the presence of any other attribute. Bayes theorem works on a probability function, see the figure (19):

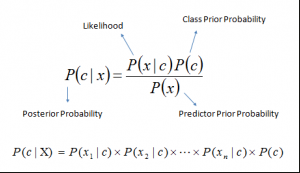


Figure (19): P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes). P(c) is the prior probability of class. P(x|c) is the likelihood, which is the probability of predictor given class. P(x) is the prior probability of predictor. (Ray, 2017)

One of the main advantages of using Naïve Bayes in my project is that when the assumption of independence holds, it performs better compared to other models like logistic regression, and it works well with a lower number of training data. But when it comes to applying in the real world, it is often referred to as a lousy estimator as it is almost impossible to get entirely independent predictors in real life.

Talking about the application of the Naïve Bayes algorithm, there have been many proposed methods used for a recommendation system, for example, (Diaz, et al., 2019) has opted to go with a Bayesian scheme for Collaborative Filtering method which promises to produce results as good as matrix factorization models, whose predictions can also be explained.

### 3.9.2 Support Vector Machine

Support Vector Machine (SVM) is a supervised machine learning algorithm that can be used for both classification and regression problems. However, it is widely used for classification problems. A Support Vector Machine is a discriminative classifier defined by a separating hyperplane (Patel, 2017). In other words, given labelled training data (supervised learning), the algorithm outputs an optimal hyperplane that categorizes new test data points.

Now, I won’t go that much in-depth to explain how it works. Our motive here is to distinguish between the classes with hyper-planes plotted on a two-dimensional space. How SVM does is by passing different lines separating the two classes and how well the hyper-plane segregates, depends on the regularization parameter of the algorithm, such as cost, kernel and the gamma parameter settings, which are the most common.

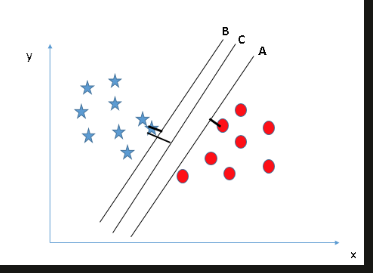


Figure 10 Source: (Ray, 2017)

A right hyper-plane will be decided by maximizing the distance (Margin) between the nearest data points (either class). In the figure (10), hyper-plane B is more inclined towards the class on the left and the hyper-plane A towards the class on the right, but hyper-plane C has the highest margin compared to both A and B. Hence, C is classified as the right hyper-plane. This is the most straightforward example for an SVM implementation, but in the real world, the data is very noisy, and it would be impossible to define a hyper-plane that would be able to separate the two classes completely, this is where we have to look at the cost parameter and adjust it according to the needs. As far as applications go, (Puspaningrum, et al., 2019) showed us an interesting approach to implement Recommendation System for Assistant Driver using Least Square Support Vector Machine and Fuzzy Logic. Since I have two labelled classes in my dataset, it would be interesting to see how well does SVM performs on our data and the test dataset.

### 3.9.3 Random Forest

Random Forest is an ensemble classifier that combines several decision trees, and the class with the most votes becomes our model’s prediction. Random Forest could be the easiest to understand and one of the finest working classifier. Before going on to Random Forest, I will provide a quick synapse on Decision Tree as they are the primary building block. In simple terms, to begin with, we will have a root node for our tree to which our full training will be provided. Now, each node will ask a True-False question about one of the features (attributes) and in response, it will be divided into two sub-nodes or subsets. The subsets then become the input to the child nodes of the tree. The goal of this method is to breakdown all the labels as we perceive down or, in other words, to produce the purest possible distribution of the label in each node.

This might look like many iterations, but a computer handles it very well. In technical terms, the predictions have variance because they will be widely spread around the right answer. What if we take predictions from hundreds of individual trees? The predictions made by decision trees might not be accurate, but combined together, the predictions will be closer to the mark on average, and that is how a Random Forest classifier works. It should get more apparent through the graphical representation in the figure (1).

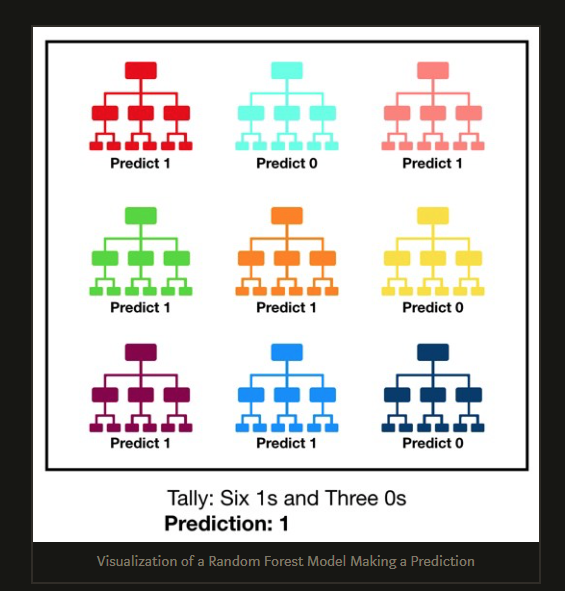


Figure 11 Source: (Yiu, 2019)

The random in Random Forest depicts that each decision tree in the forest considers a random subset of features (attributes) when forming questions and only has access to a random set of the training data points. This increases diversity in the forest leading to more robust overall predictions, hence the name ‘Random Forest.’ (Koehrsen, 2017). The best thing about the Random Forest classifier is that it never overfits the data. (Ayata, et al., 2018) have experimented in their paper recommend music based on emotions using Decision Tree, Random Forest, SVM and K-NN and have got some promising results, which tells Decision Tree and Random Forest are well applicable to this area.

### 3.9.4 K-Nearest Neighbors

The K-Nearest Neighbors algorithm is used for both classification and regression problems. In both cases, the input consists of the k closest training example in the feature space, but the output depends on whether k-NN is used for classification or regression. For classification, the output is a class membership, and for regression, the out is the property value for the object (Wikipedia, 2020). K-NN is instance-based learning where the function is only approximated locally, and all computations are deferred until function evaluation. In the classification phase, k is a user-defined constant, and an unlabeled vector is classified by assigning the label, which is most frequent among the k training samples nearest to that query point. Because K-NN is a ‘lazy learner,’ it works well on a small dataset and would not work well on noisy data.

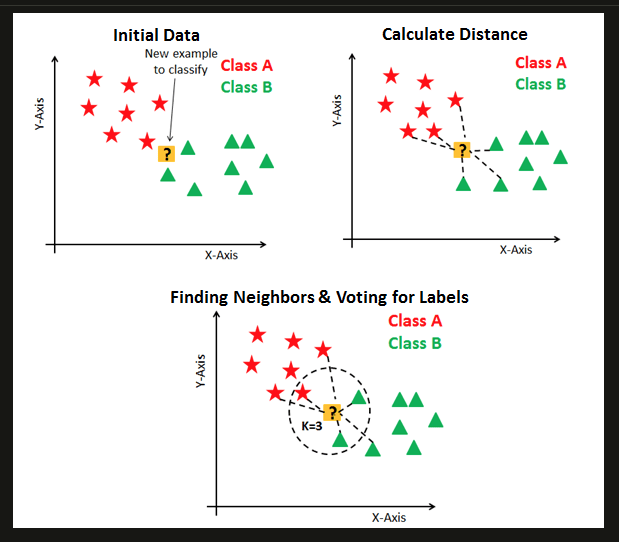


Figure 12 (Navlani, 2018)

K-NN algorithm is based on feature similarity: choosing the right value of k is a process called parameter tuning, and its import for better accuracy. For example, if you choose a low value of k, the chances are you will get a skewed answer because of the noise present, and if k value is high, there are chances of it overfitting, and the process takes up too many resources, time and computation power. One of the correct ways to choose a value of K is by taking a square root of n, where n is the total number of data points. We avoid choosing odd values of K to avoid confusion between two classes of data.

### 3.9.5 Neural Networks

The purest form of a deep learning model can be called Simple Neural Networks. A neural network is a type of machine learning which models itself after the human brain, creating an artificial neural network that, via an algorithm, allows the computer to learn by incorporating new data (DeMuro, 2019). Neural Networks are not as easy to implement as basic machine learning algorithms are. How Neural Network works is just like a child’s developing brain, this algorithm learns by adopting different techniques. In our case, we have used it as supervised learning, with the labelled data provided, the algorithm gets modified until it can process the dataset to get the desired results.

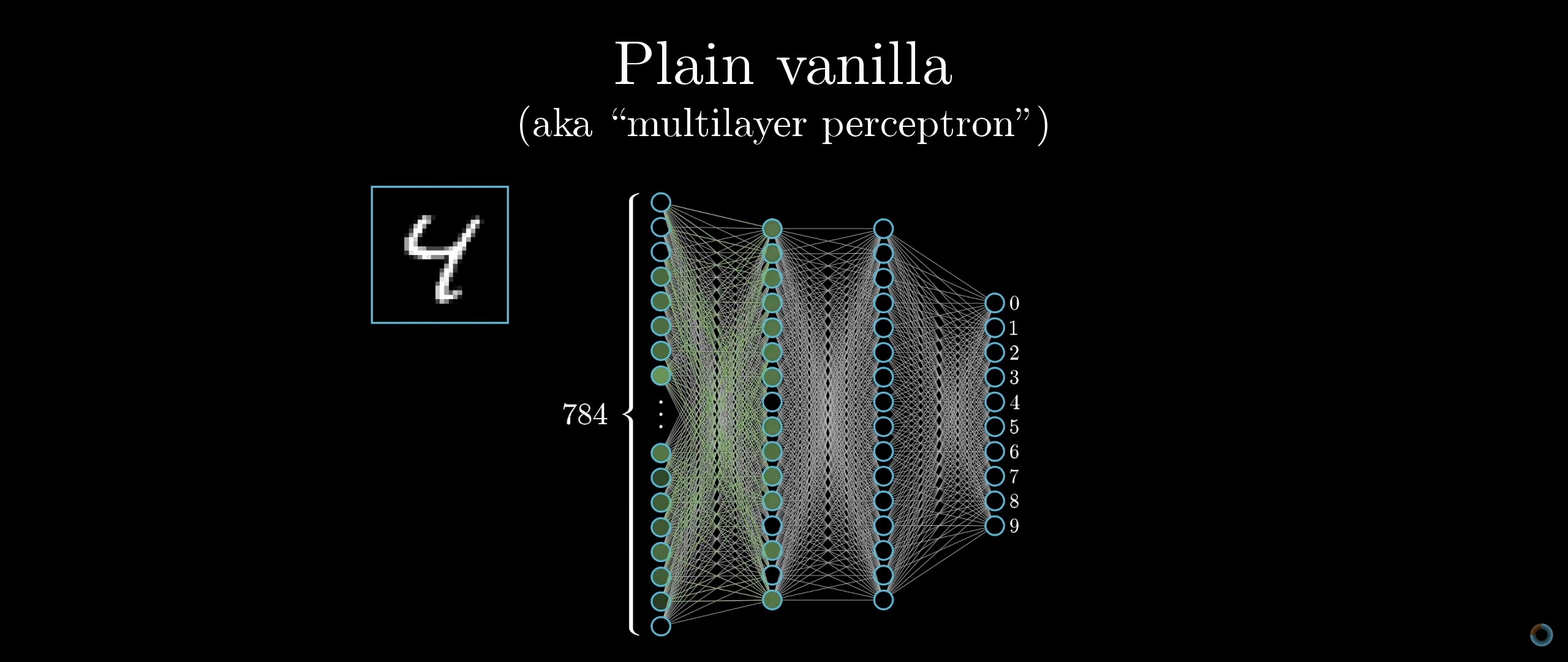


Figure 13 Neural Network connections

The figure above depicts the four layers in total. They can be classified as the input layer, first, hidden layer, second hidden layer and finally, the output layer. Each layer o nodes trains on a distinct set of features based on the previous layer’s output. Neural Networks are widely used for handwriting recognition, image classification, natural language processing, which involves text classification and so on. I wanted to see if my dataset domain could be proved a worth using neural networks or not.

# Chapter 4: Results and Discussion

The life cycle of this project starts with choosing a domain, collecting the data, exploring the data, splitting the data into training and test sets, training the models with different classifiers, evaluating the classifiers and then comparing the accuracies.

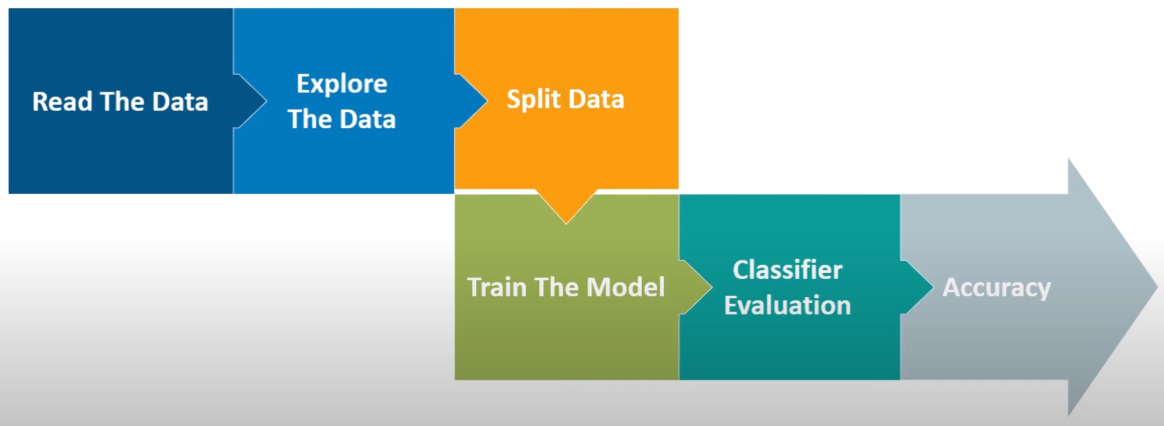


Figure 14 Data Science Life Cycle

Our motive here is to find an optimal classifier that would work the best among the four datasets I have. By optimal, I mean to not only get the best accuracy but also to get proper classification between the correctly identified True Positives and True Negatives. True Positives, True Negatives, Precision, Recall, are calculated using a Confusion Matrix that would be available to form after a model has been trained and evaluated. For example, if we think to maximize Recall, we will recommend all the items to the user according to their taste, that’s 100% Recall value. But, let’s say if we recommend 100 songs and the user only liked a single song, that would be called as a 0.1% Precision. We aim to maximize both Precision and Recall. Note: All the confusion matrices are conducted on the test data, which is set by the ‘test\_train\_split.’

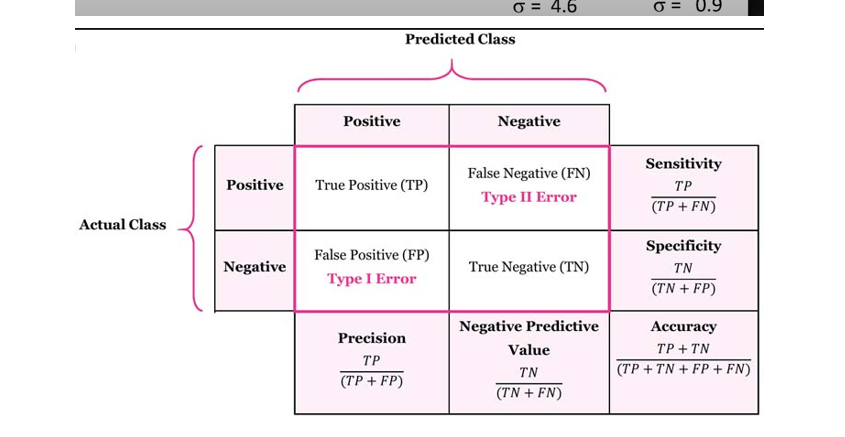


Figure 15 Confusion Matrix

The above example is for a binary classification confusion matrix. True Positive in a confusion matrix gives the information of the correctly identified instances, and True Positives are the wrongly identified instances. False Negatives are the instances that are in class A but incorrectly classified as in class B. For the data, I have used a 25% split of the dataset in which 1032 data points are for the training set, and 345 data points are for the test set. Since all the datasets have the same attribute features, it would be easier to predict how a classifier works with a very less standard deviation between each dataset. In general, there is next to no way to tell which algorithm works best for all the domains; that is why it is called an experimental science as there involves a lot of experimentation. The accuracy in this project would be entirely dependant on a user’s preference for music. A person who usually listens to the same genre of music would be natural to differentiate from others as there would not be that much noise in the dataset.

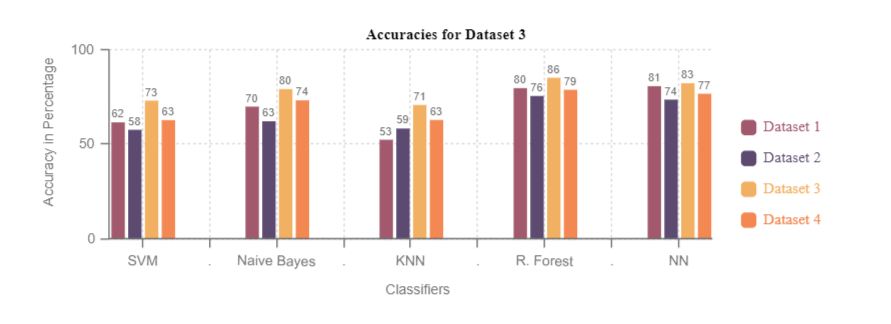


Figure 16: Bar plot for accuracies. The x-axis represents the classifiers, and the y-axis represents the accuracies.  
Four bars in each classifier represents datasets from 1 to 4, respectively.

KNN produced the worst results for us with 61.5% average accuracy, SVM produced a slightly better result with 64% average accuracy. Third came Naïve Bayes with 71.75% accuracy on average. The last two classifiers performed head to head with NN producing 78.75% accuracy on average, the model with the highest efficiency is Random Forest with 80.25% accuracy on average across the board.

## 4.1 Results Explanation

Table 1 represents the accuracy of all the algorithms implemented in this project. The first column represents all the machine learning techniques used following but the accuracies of the mentioned dataset.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Techniques** | **Dataset 1** | **Dataset 2** | **Dataset 3** | **Dataset 4** |
| **SVM (1)** | 55 | 58 | 68.3 | 59.5 |
| **SVM (2)** | 61.9 | 57.8 | 73.4 | 63.1 |
| **SVM (3)** | 52.1 | 53.1 | 61.4 | 58.5 |
| **Naive Bayes (1)** | 66.6 | 62.5 | 79.5 | 73.6 |
| **Naive Bayes (2)** | 70.2 | 57.6 | 73 | 68.6 |
| **KNN (1)** | 52.7 | 56.1 | 68.4 | 61.1 |
| **KNN (2)** | 52.4 | 58.6 | 70.6 | 60.1 |
| **KNN (3)** | 52.1 | 57.8 | 71.1 | 63.2 |
| **R. Forest (1)** | 73.7 | 74 | 84.2 | 78.1 |
| **R. Forest (2)** | 78.8 | 75.3 | 84.7 | 78.8 |
| **R. Forest (3)** | 80 | **75.9** | **85.5** | **79.1** |
| **NN (1)** | 79.3 | 70.9 | 82.1 | 72.68 |
| **NN (2)** | **81.05** | 72.8 | 82.6 | 77.03 |
| **NN (3)** | 80.03 | 73.9 | 82.1 | 76.3 |

Table 1: Accuracy table

Here, dataset 1 is the one extracted by me, dataset 2 is available at (McIntire, 2017), dataset 3 is obtained by emailing (Santos, 2017) and dataset 4 is available in (Georgieva, et al., 2018). All the datasets share the common characteristics of Audio Features, apart from dataset 4, that has three extra features such as Year, Month and ArtistScore. It is worth noticing that these three parameters play a massive part in the increased accuracy obtained out of the dataset. I was curious to see whether removing these features from the dataset would reflect any changes in the accuracy or not, and much to my surprise, the accuracy reduced by an average of 3% by applying the Neural Network algorithm on it (those results are not shared in this report).

The initial idea in this project was to investigate how effective are basic machine learning methods for a Content-Based recommender system and if there is any possibility to get better accuracies from them by tweaking their hyperparameters. Taking that into consideration, the first **(1)** settings of each mentioned techniques (excluding for Naive Bayes and Neural Network) have the baseline hyperparameter settings. Rest of the results are explained as follow:

4.1.1 SVM: **(2)** In this instance, the hyperparameter C is changed from 1 (default) to 10. The C parameter is called the Cost parameter, which optimizes SVM on how much you want to avoid the misclassifying each training example. Larger the value of C, the better the hyperplane does its job to separate the classes. The con is overfitting. (**3)** Along with the C parameter, the kernel parameter was also changed to ‘poly.’ The default value of the kernel is RBF, which is Radial Basis Function. This parameter decides the curving degree of the hyperplane.

The second parameter tuning turned out to be worse than the first one, which got the highest accuracy among all the SVMs. So we are using the confusion matrix for SVM(2).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SVM** | **Dataset 1** | | **Dataset 2** | | **Dataset 3** | | **Dataset 4** | |
| **Positive** | 40 | 91 | 156 | 23 | 103 | 9 | 383 | 0 |
| **Negative** | 14 | 131 | 138 | 71 | 48 | 55 | 262 | 0 |

Table 2: SVM Confusion Matrice for all the datasets

The highest accuracy is achieved by dataset 3 (Note: No two datasets share the same data, but features), which is 73%, and the second-best is performed by dataset 4 with a difference of 10%. Even if dataset 4 has all the True Positives instances correctly identified, all the False Positive cases are classified as True Negative. This tells even if the accuracy is higher, it doesn’t really mean the model has performed better.

4.1.2 Naive Bayes: **(1)** Naive Bayes doesn’t have that many tunable parameters. So instead, I decided to go with Bernoulli Naive Bayes. One of the main reasons to include BernaulliNB in the classification because Bernoulli is more designed for binary features. Since our problem is a binary classification problem, this algorithm fits our description very well.

**(2)** The second one I went for is Gaussian Naive Bayes, imported as GaussianNB. Naive Bayes classifier considers that big data is generated through a Gaussian process (Pulipaka, 2017). Naïve Bayes is famous for binary classification. As you can see in table 1, GaussianNB outperformed BernoulliNB for a good margin. So the confusion matrix here is taken from NB(2).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **NB** | **Dataset 1** | | **Dataset 2** | | **Dataset 3** | | **Dataset 4** | |
| **Positive** | 64 | 67 | 139 | 55 | 109 | 3 | 342 | 41 |
| **Negative** | 15 | 130 | 116 | 93 | 55 | 48 | 129 | 133 |

Table 3: Naïve Bayes Confusion Matrices for all the datasets.

All the accuracies for Naïve Bayes are somewhat close enough for all the datasets, except for dataset 2. The True Positives for dataset 2 are looking good, but the accuracy decreased due to a high number of False Negative.

4.1.3 KNN: **(2)** The settings for a baseline KNN has 5 number of neighbours as default. In KNN (2), I tried changing the value to different numbers and train the model to see if the accuracy changes. The value of 12 neighbours was the optimal selection for our environment. For **KNN(3)**, Along with the neighbour parameter, I changed the ‘p’ value, which is a power parameter for the Minkowski metric. By default, it stays at 2, but I changed it to value 1 to see a slight change in the accuracy from the baseline performance.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **KNN** | **Dataset 1** | | **Dataset 2** | | **Dataset 3** | | **Dataset 4** | |
| **Positive** | 108 | 0 | 154 | 90 | 105 | 36 | 378 | 90 |
| **Negative** | 105 | 72 | 122 | 137 | 42 | 6 | 184 | 154 |

Table 4: KNN Confusion Matrices for all the datasets

Dataset 3 performed much better than any other dataset provided with a 71.1% accuracy. And if we look at the confusion matrix as well, a 71% accuracy will get you a fair amount of correctly classified True Positives, but again, the False Negatives are the values that play a large hand in decreasing the accuracies.

4.1.4 Random Forest: **(2)** Random Forest is an ensemble machine learning algorithm that incorporates several Decision Trees, hence Forest. The baseline model of Random Forest uses 100 trees as a default parameter in Python’s Scikit Learn library. That means, it will have 100 trees made in Decision Tree fashion, and the class with the maximum mean of the votes will be chosen. In R. Forest (2), the number of trees value is tuned, and our model works well with 200 trees but doesn’t work well with either 50, 300, or 400. There’s another highly tunable parameter called max\_features that lets you choose how many feature attributes should be taken into account while building the trees. The range goes from 0.1 to 0.9 for this hyperparameter, 0.1 will accept the least number of features to be used while 0.9 is the most. This parameter, combined with the number of trees, gave an excellent accuracy score for us. **(3)** Initially, I kept the number of trees as 200 and then tweaked the max\_features parameter, eventually, including a setting called min\_samples\_leaf that takes numbers as an input that indicates the amount of minimum number of samples required to be at a leaf node. The final configuration ended up being like this: “RandomForestClassifier(n\_estimators=300, max\_features=0.7, min\_samples\_leaf=2),” which gave us the best performance out of Random Forest overall for all the datasets that I have used.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Random Forest** | **Dataset 1** | | **Dataset 2** | | **Dataset 3** | | **Dataset 4** | |
| **Positive** | 137 | 31 | 187 | 57 | 127 | 14 | 385 | 83 |
| **Negative** | 38 | 139 | 64 | 195 | 25 | 103 | 85 | 253 |

Table 5: Confusion Matrices for Randon Forest

Even though my Neural Network performs slightly better for dataset 1, there is only a difference of 1.05% between the accuracies. If we look at the confusion matrix here, the overall performance of the whole classifier has promising results, with True Negatives and False Negatives coming out to be minimal. If you remember that the True Negatives and False Positive for dataset 4 were nill in the SVM classifier but still produces a 58.5% accuracy, comparing the True Positives for both the classifiers for dataset 4 has nearly identical values but what makes Random Forest a better classifier is that it predicts the False Positive values correctly than any other classifier in our project.

4.1.5 NN:There are no baseline results as such because a Neural Network is entirely dependent on the programmer and the dimensions of the dataset. NN has a general notation on how to begin, but there is no specific starting point from where to begin. NN is all about how many hidden layers are there in the model and what is the number of Neurons you should go with; all these values would come by experimenting over and over again and keep tuning weights to minimize the total error rate. Since NN adjusts the weights by itself, our goal here is to find an optimal set of hidden layers (or even a single hidden layer) by passing different values of Neurons. Each layer needs to be specified with an activation function, and an activation function controls how the Neurons should behave upon getting the input from the previous layer. In this project, I have applied three different settings for the NN algorithm. As shown in figure (17), the model name is sequential, which means all the layers will be executed in order as programmed. The layer type describes which type of layer is being used in the Neural Network. Since this is a simple NN, we have used the Dense layer. Examples for layers could be the Flatten layer, convolutional layer, etc., that depends on what type of algorithm problem you are working with. The output shape column describes the shape of the output result. For example, the last layer is a Dense layer with one as the output shape, which means that the Dense output layer will produce a single result. Since our problem is a binary classification problem, it will spit out either the ‘yes’ class or the ‘no’ class. The activation function we have used is ‘reLu,’ which is a Rectified Linear Unit that will only be activated when an input above 0 is given. The activation function used in the output layer is ‘sigmoid’ which has an activation area between 0 and 1. Finally, the model is compiled with the parameters: loss=’binary\_crossentropy’ (since our project is based on a binary classification), optimizer=’adam’ (this field uses adam as an optimizer because it gives out optimal results overall), metrics=’accuracy’(we specified that our model should compile the results taking the accuracy measure as a guide to be improved))

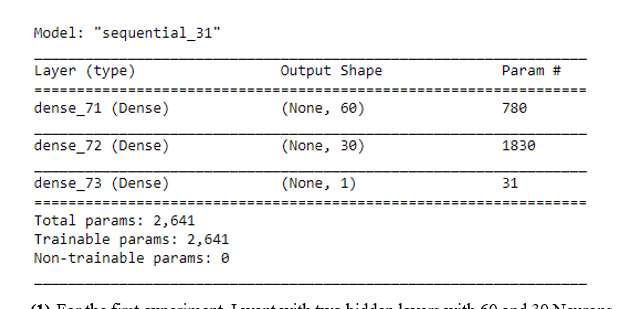


Figure 17: Neural Networks Structure

1. For the first experiment, I went with two hidden layers with 60 and 30 Neurons, respectively. Note that all the tests for this project are with different activation functions, and ‘reLu’ has produced the best results among all the activation functions.
2. The second experiment was done with a single hidden layer with 30 Neurons. This setting has produced the best result among all the different parameters of our Neural Network.
3. The final experiment was done with again a single hidden but with 60 Neurons this time.

It is not said that the above changes to the hyperparameters changes are optimal and certainly not limited to these. There were thousands of different parameter tweaks done before publishing these results, and these are the ideal changes I could find. A classifier can be categorized into two types, a good classifier and a bad classifier. An algorithm can achieve very high accuracy and still not be a good classifier. This can be explained in two scenarios; first, a Decision Tree is reputed for it being a classifier that overfits data. Overfitting can be described as a model that predicts all the instances correctly on the training set but performs very poorly when an unseen test set is provided to the model. In this case, the model can achieve a 100% accuracy, but it will not be worth deploying in the real world. The second scenario can be explained by an example; suppose a classifier output a prediction accuracy as high as 80%, but when you look at the confusion matrix, the picture is different. If the model predicts all the True Positives correctly, but it predicts all the True Negatives as False Negative, the accuracy would be biased based on the correctly identified actual positive class, but in general, it a bad model.

## 4.2 Testing:

To check how the top classifier works have been implemented on two external test sets. For this purpose, I have used Weka. I have only applied the test sets on Random Forest as it has produced the best result overall. The reason for me to go with Random Forest rather than Neural Network is because they both provide nearly the equivalent results, but the time taken by NN is far more than what Random Forest takes. I think it is a viable option to go with Random Forest because the computation time our NN took was around 17 minutes on an average and Random Forest produced the same or better results within seconds.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test with R. Forest(3)** | Test Set 1 | | Test Set 2 | |
| **Positive** | 18 | 4 | 19 | 1 |
| **Negative** | 11 | 3 | 10 | 6 |
| **Precision** | 0.54 |  | 0.74 |  |
| **Recall** | 0.58 |  | 0.69 |  |

Table 6 Confussion Matrix obtained from Test Datasets

Test Set 1 has 90 data points, and Test Set 2 has 36 data points. The first result is not that impressive with near sub-par results with 58% accuracy. That is because the test set is taken from Spotify’s Discover Weekly playlist, which uses different approaches and algorithms to come up with this playlist, and our algorithm think all the songs are likeable. On the other hand, Test Set 2 provided some decent results with 69.5% accuracy. I believe that is because this dataset contains the newly released songs from the artists I follow on Spotify, and the chances of every song to be likeable are very less. Remember, before beginning with the Spotify’s Discover Weekly to initialize, Spotify says it would take some weeks for their algorithm to learn how a user interacts with the music, for example, skipping songs after 5 secs, 20 seconds or halfway through of listening which determines the taste of the user and the algorithm to learn.

# Chapter 5: Conclusion and Future Work

In this project, I extracted the data from my personal Spotify’s playlists and made one training dataset with nearly 700 data points from the music I like and 700 more with the music I don’t like. I further extracted two test sets from two different playlists, which are produced by Spotify’s machine learning algorithms and deep learning. For me, Random Forest has turned out to be the best classifier with the highest accuracy overall and a promising result on the confusion matrix as well. Worked best for me out of the box, but tuning the parameters took us to the top. One of the significant reasons to trust Random Forest’s result is because it corrects decision trees’ habit of overfitting to their training set (Wikipedia, 2020).

Coming to our research questions:

1. How well do state-of-the-art machine learning techniques perform for a music recommendation system based on the user’s listening history? The answer is that some of the underlying machine learning techniques are not quite that powerful when it comes to predicting good results in our domain of the dataset. But on the contrary, the Random Forest classifier performed better than a Neural Networks approach. Again, a Neural Network is built differently as per the personal preferences and experiments of an individual programmer.
2. If they are performing well, is there any room for improvement in the existing algorithms to make it perform better for the given domain of the dataset? Yes, there is plenty of room for improvement even after all the tuning performed on all these machine learning techniques. The results I have obtained are promising but not limited to this experiment.

With that being said, my Random Forest method with tuned parameters has produced better results on the datasets used in (Georgieva, et al., 2018), (McIntire, 2017) and (Santos, 2017). (Santos, 2017) has applied Logistic Regression getting an accuracy of 82%, whereas my Random Forest produced an accuracy result of 85.5%. In (McIntire, 2017), he has used Logistic Regression, KNN, Decision Tree, and Random Forest, with the accuracies 60%, north to ’50s, approx. 69% and 69.3%, respectively. My Random Forest produced an efficiency of 75.9% on an average for all the different Random Forest parameter settings. Lastly, (Georgieva, et al., 2018) in their paper have used various machine learning algorithms such as Logistic Regression (73%), GDA (73%), SVM (72% w/ Bagging), Decision Tree (100%, resulted in overfitting) and Neural Network achieving 77% accuracy, my Neural Network produced the similar accuracy rates as their; however, my Random Forest algorithm has performed slightly better than their result which is 79.1% on an average.

One thing to notice here is that our datasets are not too big to be provided as seed values to train a model, but gathering a massive dataset in this manner would take a long time taking disliked content in observation as well. The limitation of this project is less research done on music playlist generation based on likes and dislikes and particularly on Spotify’s Audio Features data. Is accuracy measure the only thing to be considered while training the model? Accuracy does matter, but the precision and recall measures are to be considered equally. Would you consider to be recommended just a collection of few songs which you will definitely like, or would you like to expand your horizon with more items that you are likely to interact with, including the items which you will probably end up disliking. With the advancement in machine learning and deep learning techniques, it is concluded that the more appropriate data fed to the model, the better the chances to get results through higher accuracies with Precision and Recall.

I would like to implement further my research using Convolutional Neural Networks and try using more deep learning techniques with access to a high dimensional dataset and include surveys actually to get a meaningful result. Just like Spotify, I would like to extend my research by using Natural Language Processing.

# References

Ayata, D., Yaslan, Y. & Kamasak, M. E., 2018. *Emotion Based Music Recommendation System Using Wearable Physiological Sensors,* Istanbul: IEEE.

Bertin-Mahnieux, T., Ellis, D. P., Whitman, B. & Lamere, P., 2011. *Million Song Dataset,* s.l.: In Proceedings of the 12th ISMIR.

. B., T. & J., 2014. *Echo Nest.* [Online]   
Available at: https://blog.echonest.com/post/78749300941/the-echo-nest-joins-spotify  
[Accessed 05 2020].

Ciocca, S., 2017. *Medium.* [Online]   
Available at: https://medium.com/s/story/spotifys-discover-weekly-how-machine-learning-finds-your-new-music-19a41ab76efe  
[Accessed 02 2020].

DeMuro, J., 2019. *Techradar.* [Online]   
Available at: https://www.techradar.com/uk/news/what-is-a-neural-network  
[Accessed 06 2020].

Diaz, P. V., Ortega, F., Cobos, E. & Cabrera, R. L., 2019. *A Collaborative Filtering Approach Based,* Madrid: IEEE.

Fowler, J., 2014. *The Next Web.* [Online]   
Available at: https://thenextweb.com/dd/2014/07/12/crave-human-curated-playlists/  
[Accessed 21 02 2020].

Gelo, T., 2018. *GitHub.* [Online]   
Available at: https://github.com/tgel0/spotify-data/blob/master/notebooks/SpotifyDataRetrieval.ipynb  
[Accessed 02 2020].

Georgieva, E., Suta, M. & Burton, N., 2018. *HITPREDICT: PREDICTING HIT SONGS USING SPOTIFY DATA,* Standford: STANFORD COMPUTER SCIENCE 229: MACHINE LEARNING.

Giacaglia, G., 2019. *Medium.* [Online]   
Available at: https://medium.com/datadriveninvestor/behind-spotify-recommendation-engine-a9b5a27a935  
[Accessed 12 02 2020].

Hariri, N., Mobasher, B. & Burke, R., 2012. *Context-aware music recommendation based on latenttopic sequential patterns,* Chicago: RecSys '12: Proceedings of the sixth ACM conference on Recommender systems.

Jannach, D., Lerche, L. & Kamehkhosh, I., 2015. *Beyond "Hitting the Hits": Generating Coherent Music Playlist Continuations with the Right Tracks,* Germany: RecSys '15.

Johnson, C., 2014. *SlideShare.* [Online]   
Available at: https://www.slideshare.net/MrChrisJohnson/algorithmic-music-recommendations-at-spotify/10-Collaborative\_Filtering10HeyI\_like\_tracks\_P  
[Accessed 02 2020].

Knees, P., Pohle, T., Schedl, M. & Widmer, G., 2006. *Combining audio-based similarity with web-based data to accelerate automatic music playlist generation,* Austria: MIR '06 Proceeding.

Koehrsen, W., 2017. *Medium.* [Online]   
Available at: https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d  
[Accessed 06 2020].

Lad, R., 2017. *SPOTIFY MUSIC ANALYSIS USING MACHINE LEARNING,* Boston: Northeastern University.

Lamere, P., 2014. *Spotipy.* [Online]   
Available at: https://github.com/plamere/spotipy/blob/master/LICENSE.md  
[Accessed 04 2020].

McIntire, G., 2017. *A Machine Learning Deep Dive into My Spotify Data.* [Online]   
Available at: https://towardsdatascience.com/is-my-spotify-music-boring-an-analysis-involving-music-data-and-machine-learning-47550ae931de  
[Accessed 22 04 2020].

McIntire, G., 2017. *Kaggle.* [Online]   
Available at: https://www.kaggle.com/geomack/spotifyclassification  
[Accessed 03 2020].

Minds, C., 2019. *Slide Share.* [Online]   
Available at: https://www.slideshare.net/CrossingMinds/recommendation-system-explained/6  
[Accessed 05 04 2020].

Navlani, A., 2018. *Data Camp.* [Online]   
Available at: https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn  
[Accessed 06 2020].

NSynthSuper, 2020. *NSynth Super with Google.* [Online]   
Available at: https://nsynthsuper.withgoogle.com/  
[Accessed 06 05 2020].

Nykowska, H., 2019. *The Information Lab.* [Online]   
Available at: https://www.theinformationlab.co.uk/2019/08/07/connect-to-spotify-api-with-alteryx/  
[Accessed 02 2020].

One, A. I. -. A. i., 2016. *YouTube.* [Online]   
Available at: https://www.youtube.com/watch?v=2uxXPzm-7FY  
[Accessed 04 2020].

Oord, A. V. D., Dieleman, S. & Schrauwen, B., 2013. *Deep content-based music recommendation,* s.l.: NIPS conference.

Pandora, 2020. *Pandora.* [Online]   
Available at: https://www.pandora.com/  
[Accessed 01 05 2020].

Patel, S., 2017. *Medium.* [Online]   
Available at: https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72#:~:text=A%20Support%20Vector%20Machine%20(SVM,hyperplane%20which%20categorizes%20new%20examples.  
[Accessed 06 2020].

Pauws, S., Verhaegh, W. & Vossen, M., 2008. *Music playlist generation by adapted simulated annealing,* The Netherlands: Philips Research.

Popper, B., 2015. *The Verge.* [Online]   
Available at: https://www.theverge.com/2015/9/30/9416579/spotify-discover-weekly-online-music-curation-interview  
[Accessed 17 04 2020].

Pulipaka, D. G., 2017. *Medium.* [Online]   
Available at: https://medium.com/@gp\_pulipaka/applying-gaussian-na%C3%AFve-bayes-classifier-in-python-part-one-9f82aa8d9ec4  
[Accessed 06 2020].

Puspaningrum, A., Suheryadi, A. & Sumarudin, A., 2019. *Pre-Collision Warning and Recommendation System for Assistant Driver using Least Square Support Vector Machine and Fuzzy Logic,* Indonesia: IEEE.

Ray, S., 2017. *Analitics Vidhya.* [Online]   
Available at: https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/  
[Accessed 06 2020].

Ray, S., 2017. *Analytics Vidhya.* [Online]   
Available at: https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/  
[Accessed 06 2020].

Santos, J. D. D., 2017. *Is my Spotify music boring? An analysis involving music, data, and machine learning.* [Online]   
Available at: https://towardsdatascience.com/is-my-spotify-music-boring-an-analysis-involving-music-data-and-machine-learning-47550ae931de  
[Accessed 17 03 2020].

Santos, J. D. D., 2017. *Towards Data Science.* [Online]   
Available at: https://towardsdatascience.com/is-my-spotify-music-boring-an-analysis-involving-music-data-and-machine-learning-47550ae931de  
[Accessed 17 03 2020].

Sharma, N., 2019. *BR Soft Tech.* [Online]   
Available at: https://www.brsoftech.com/blog/spotify-statistics-future-of-the-media-streaming-app/  
[Accessed 05 2020].

Sidekick, 2018. *Sidekick.* [Online]   
Available at: https://sidekick-music.com/2018/11/30/top-spotify-playlist-curators-2018/  
[Accessed 04 2020].

Spotify.me, 2020. *Spotify.* [Online]   
Available at: www.spotify.me  
[Accessed 21 04 2020].

Spotify, 2016. *Spotify.* [Online]   
Available at: https://artists.spotify.com/blog/say-hello-to-release-radar  
[Accessed 05 05 2020].

Spotify, 2020. *Spotify.* [Online]   
Available at: https://www.spotify.com/uk/discoverweekly/  
[Accessed 01 02 2020].

Spotify, 2020. *Spotify for Developers.* [Online]   
Available at: https://developer.spotify.com/documentation/web-api/reference/tracks/get-audio-features/  
[Accessed 02 2020].

Spotify, 2020. *Spotify For Developers.* [Online]   
Available at: https://developer.spotify.com/documentation/web-api/  
[Accessed 02 2020].

Spotipy, 2014. *Spotipy.* [Online]   
Available at: https://spotipy.readthedocs.io/en/2.12.0/  
[Accessed 05 2020].

Watson, 2020. *Exportify.* [Online]   
Available at: https://rawgit.com/watsonbox/exportify/master/exportify.html  
[Accessed 03 2020].

Weka, 2020. *Weka.* [Online]   
Available at: https://www.cs.waikato.ac.nz/ml/weka/  
[Accessed 04 06 2020].

WekaDeepLearning4j, 2020. *WekaDeepLearning4j: Deep Learning Using Weka.* [Online]   
Available at: https://deeplearning.cms.waikato.ac.nz/  
[Accessed 05 2020].

WIki, 2020. *Wikipedia.* [Online]   
Available at: https://en.wikipedia.org/wiki/Collaborative\_filtering  
[Accessed 03 2020].

Wikipedia, 2020. *Wikipedia.* [Online]   
Available at: https://en.wikipedia.org/wiki/K-nearest\_neighbors\_algorithm  
[Accessed 06 2020].

Wikipedia, 2020. *Wikipedia.* [Online]   
Available at: https://en.wikipedia.org/wiki/Random\_forest  
[Accessed 26 05 2020].

Yiu, T., 2019. *Towards Data Science.* [Online]   
Available at: https://towardsdatascience.com/understanding-random-forest-58381e0602d2  
[Accessed 06 2020].

# Appendices:

This section includes all the codes, screenshots and plots used in this project.

The following code, dataset and plots can be found at:  
https://github.com/janups3/Classification\_Models.git

# Following is the code for the data retrieval.

*#Initialization for the Spotify Credentials. It is required that you already have Spotipy and Pandas libraries installed.*

**import** **spotipy**

**from** **spotipy.oauth2** **import** SpotifyClientCredentials

*#I have changed the original OAuth2 file with my Client ID and Client Secret. But instead, you can initialize them here as well.*

*#cid = "Client ID"*

*#csecret = "CLient Secret"*

client\_credentials\_manager = SpotifyClientCredentials()

sp = spotipy.Spotify(client\_credentials\_manager=client\_credentials\_manager)

*#This next step would take a CSV file with Track IDs as an input. The process to get the Track IDs is explained in the report.*

**import** **pandas** **as** **pd**

*#Loading the data file in a dataframe.*

data = pd.read\_csv('Input\_File\_Name.csv', delimiter=',' ) *#This takes CSV file with Track IDs as an input*

*#Extracting the useful data columns from the original file.*

track\_id = data['URI']

track\_name = data['Track Name']

artist\_name = data ['Artist Name']

df=pd.DataFrame({'artist\_name':artist\_name,'track\_name':track\_name,'track\_id':track\_id})

print(df.shape) *#To check the dataframe matrix*

df.head() *#df.head is used to show only the initial 5 rows instead of outputting all the rows.*

df.info()

*#This step is done to check if there are null items or not.*

*#Since the Spotify's Web API lets you put only 100 Track IDs as an input at once (discussed in detail in the report), the following code will iterate in batchsize of 100.*

rows = []

batchsize = 100

NoneC = 0

**for** i **in** range(0,len(df['track\_id']),batchsize):

batch = df['track\_id'][i:i+batchsize]

feature\_results = sp.audio\_features(batch)

**for** i, t **in** enumerate(feature\_results):

**if** t == **None**:

NoneC = NoneC + 1

**else**:

rows.append(t)

print('Number of tracks where no audio features were available:',NoneC)

print('number of elements in the Track IDs list:', len(rows))

df\_audio\_features = pd.DataFrame.from\_dict(rows,orient='columns')

print("Shape of the dataset:", df\_audio\_features.shape)

df\_audio\_features.head()

df\_audio\_features.info()

*#Removing the unnecessary columns from the Audio Features list.*

columns\_to\_drop = ['analysis\_url','track\_href','type','uri']

df\_audio\_features.drop(columns\_to\_drop, axis=1,inplace=**True**)

df\_audio\_features.rename(columns={'id': 'track\_id'}, inplace=**True**)

df\_audio\_features.shape

df\_audio\_features.head()

*# Merging the Track IDs list with Audio Features list in a single dataframe.*

Data\_Frame = pd.merge(df,df\_audio\_features,on='track\_id',how='inner')

print("Shape of the dataset:", df\_audio\_features.shape)

Data\_Frame.head()

Data\_Frame.info() *# Just to confirm that there are no null values.*

Data\_Frame.to\_csv('Output\_File\_Name.csv')

*# The output file will be a CSV file with the Audio Features merged with Track IDs list.*

# To remove duplicate values from our dataset:

# This notebook removes the duplicates from my initial playlists. Since I had several different playlists but there were possibilities that I have similar songs in different playlists.

**import** **pandas** **as** **pd**

data = pd.read\_csv('Output\_File\_Name.csv', delimiter=',' )

data.shape

data[~data.duplicated(subset=['track\_name','artist\_name'])].to\_csv('Dataset.csv')

*# The above line of code outputs a CSV without the duplicate values.*

*# Track name and Artist name are put together because two track names could be similar but these subsets together will be unique.*

# Code for plotting the Audio Features:

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**from** **matplotlib** **import** pyplot **as** plt

**import** **seaborn** **as** **sns**

%**matplotlib** inline

data = pd.read\_csv('Dataset1.csv')

type(data)

data.describe() *#by pandas*

data.head()

*# Custom color palette*

red\_blue = ['#19B5FE', '#EF4836']

palette = sns.color\_palette(red\_blue)

sns.set\_palette(palette)

liked\_acousticness = data[data['class'] =='yes']['acousticness']

disliked\_acousticness = data[data['class'] == 'no']['acousticness']

liked\_danceability = data[data['class'] =='yes']['danceability']

disliked\_danceability = data[data['class'] == 'no']['danceability']

liked\_duration\_ms = data[data['class'] =='yes']['duration\_ms']

disliked\_duration\_ms = data[data['class'] == 'no']['duration\_ms']

liked\_energy = data[data['class'] =='yes']['energy']

disliked\_energy = data[data['class'] == 'no']['energy']

liked\_instrumentalness = data[data['class'] =='yes']['instrumentalness']

disliked\_instrumentalness = data[data['class'] == 'no']['instrumentalness']

liked\_key = data[data['class'] =='yes']['key']

disliked\_key = data[data['class'] == 'no']['key']

liked\_liveness = data[data['class'] =='yes']['liveness']

disliked\_liveness = data[data['class'] == 'no']['liveness']

liked\_loudness = data[data['class'] =='yes']['loudness']

disliked\_loudness = data[data['class'] == 'no']['loudness']

liked\_mode = data[data['class'] =='yes']['mode']

disliked\_mode = data[data['class'] == 'no']['mode']

liked\_speechiness = data[data['class'] =='yes']['speechiness']

disliked\_speechiness = data[data['class'] == 'no']['speechiness']

liked\_tempo = data[data['class'] =='yes']['tempo']

disliked\_tempo = data[data['class'] == 'no']['tempo']

liked\_time\_signature = data[data['class'] =='yes']['time\_signature']

disliked\_time\_signature = data[data['class'] == 'no']['time\_signature']

liked\_valence = data[data['class'] =='yes']['valence']

disliked\_valence = data[data['class'] == 'no']['valence']

fig = plt.figure(figsize=(10, 8))

plt.title("Acousticness bar plot for liked/disliked songs")

liked\_acousticness.hist(alpha = 0.7, bins = 30, label = 'Liked')

disliked\_acousticness.hist(alpha = 0.7, bins = 30, label = 'Disliked')

plt.grid(b=**None**)

plt.legend(loc="upper right")

fig = plt.figure(figsize=(10, 8))

plt.title("Danceability bar plot for liked/disliked songs")

liked\_danceability.hist(alpha = 0.7, bins = 30, label = 'Liked')

disliked\_danceability.hist(alpha = 0.7, bins = 30, label = 'Disliked')

plt.grid(b=**None**)

plt.legend(loc="upper right")

fig = plt.figure(figsize=(10, 8))

plt.title("Duration (in ms) bar plot for liked/disliked songs")

liked\_duration\_ms.hist(alpha = 0.7, bins = 30, label = 'Liked')

disliked\_duration\_ms.hist(alpha = 0.7, bins = 30, label = 'Disliked')

plt.grid(b=**None**)

plt.legend(loc="upper right")

fig = plt.figure(figsize=(10, 8))

plt.title("Energy bar plot for liked/disliked songs")

liked\_energy.hist(alpha = 0.7, bins = 30, label = 'Liked')

disliked\_energy.hist(alpha = 0.7, bins = 30, label = 'Disliked')

plt.grid(b=**None**)

plt.legend(loc="upper right")

fig = plt.figure(figsize=(10, 8))

plt.title("Instrumentalness bar plot for liked/disliked songs")

liked\_instrumentalness.hist(alpha = 0.7, bins = 30, label = 'Liked')

disliked\_instrumentalness.hist(alpha = 0.7, bins = 30, label = 'Disliked')

plt.grid(b=**None**)

plt.legend(loc="upper right")

fig = plt.figure(figsize=(10, 8))

plt.title("Key bar plot for liked/disliked songs")

liked\_key.hist(alpha = 0.7, bins = 30, label = 'Liked')

disliked\_key.hist(alpha = 0.7, bins = 30, label = 'Disliked')

plt.grid(b=**None**)

plt.legend(loc="upper right")

fig = plt.figure(figsize=(10, 8))

plt.title("Liveness bar plot for liked/disliked songs")

liked\_liveness.hist(alpha = 0.7, bins = 30, label = 'Liked')

disliked\_liveness.hist(alpha = 0.7, bins = 30, label = 'Disliked')

plt.grid(b=**None**)

plt.legend(loc="upper right")

fig = plt.figure(figsize=(10, 8))

plt.title("Loudness bar plot for liked/disliked songs")

liked\_loudness.hist(alpha = 0.7, bins = 30, label = 'Liked')

disliked\_loudness.hist(alpha = 0.7, bins = 30, label = 'Disliked')

plt.grid(b=**None**)

plt.legend(loc="upper right")

fig = plt.figure(figsize=(10, 8))

plt.title("Mode bar plot for liked/disliked songs")

liked\_mode.hist(alpha = 0.7, bins = 30, label = 'Liked')

disliked\_mode.hist(alpha = 0.7, bins = 30, label = 'Disliked')

plt.grid(b=**None**)

plt.legend(loc="upper right")

fig = plt.figure(figsize=(10, 8))

plt.title("Speechiness bar plot for liked/disliked songs")

liked\_speechiness.hist(alpha = 0.7, bins = 30, label = 'Liked')

disliked\_speechiness.hist(alpha = 0.7, bins = 30, label = 'Disliked')

plt.grid(b=**None**)

plt.legend(loc="upper right")

fig = plt.figure(figsize=(10, 8))

plt.title("Tempo bar plot for liked/disliked songs")

liked\_tempo.hist(alpha = 0.7, bins = 30, label = 'Liked')

disliked\_tempo.hist(alpha = 0.7, bins = 30, label = 'Disliked')

plt.grid(b=**None**)

plt.legend(loc="upper right")

*#alpha to make the bar a bit translucent*

*#bins to see if there are not more than 30 different values, which might crowd the plot.*

fig = plt.figure(figsize=(10, 8))

plt.title("Time Signature bar plot for liked/disliked songs")

liked\_time\_signature.hist(alpha = 0.7, bins = 30, label = 'Liked')

disliked\_time\_signature.hist(alpha = 0.7, bins = 30, label = 'Disliked')

plt.grid(b=**None**)

plt.legend(loc="upper right")

fig = plt.figure(figsize=(10, 8))

plt.title("Valence bar plot for liked/disliked songs")

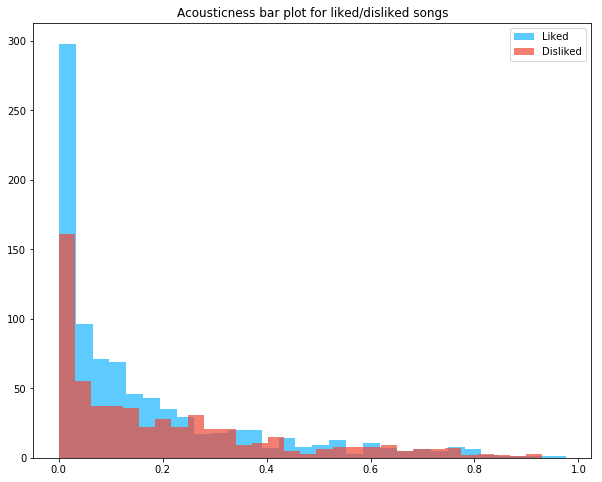
liked\_valence.hist(alpha = 0.7, bins = 30, label = 'Liked')

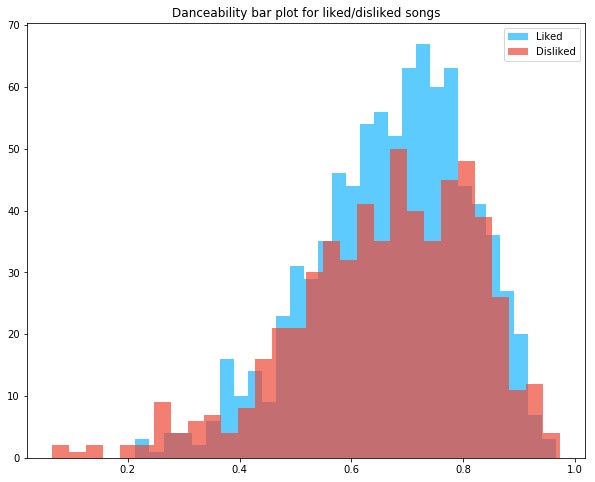
disliked\_valence.hist(alpha = 0.7, bins = 30, label = 'Disliked')

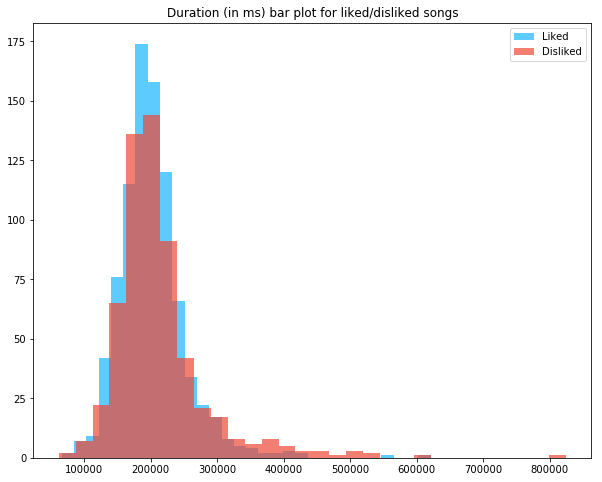
plt.grid(b=**None**)

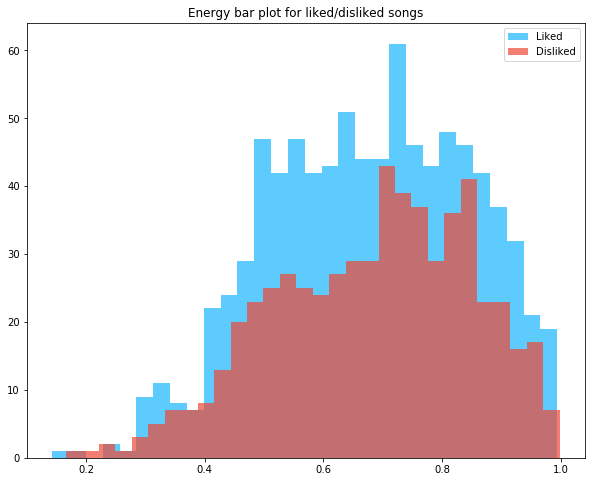
plt.legend(loc="upper right")

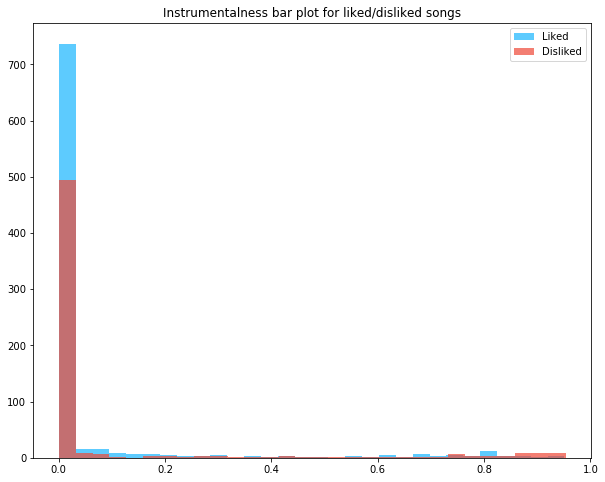
# Following are the plot:

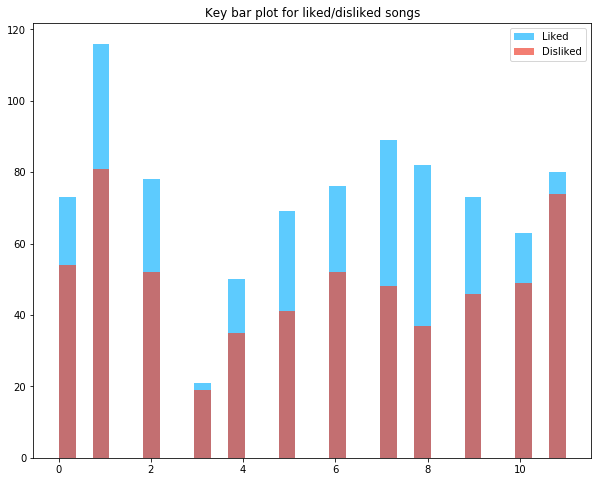


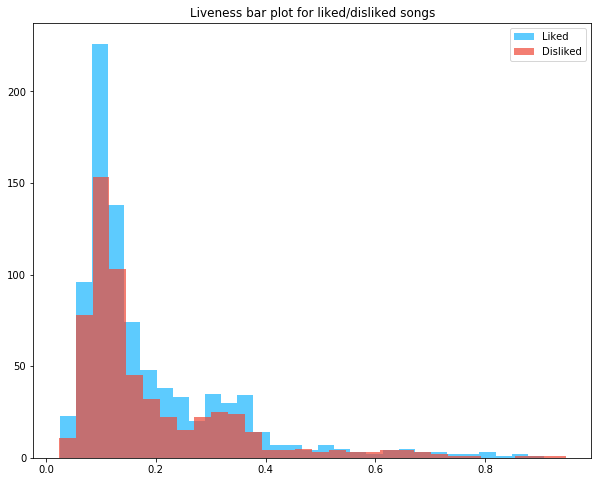


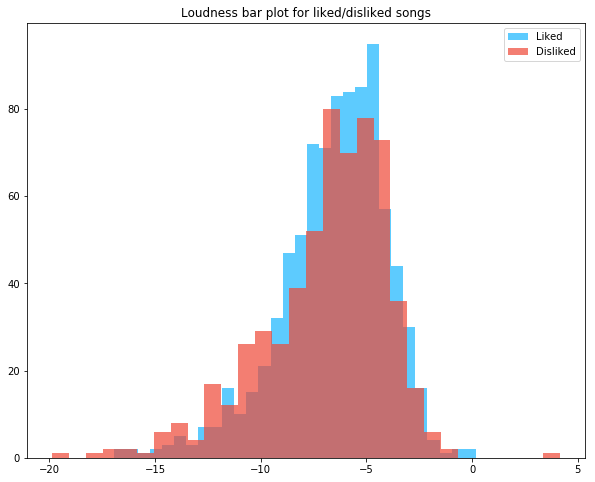


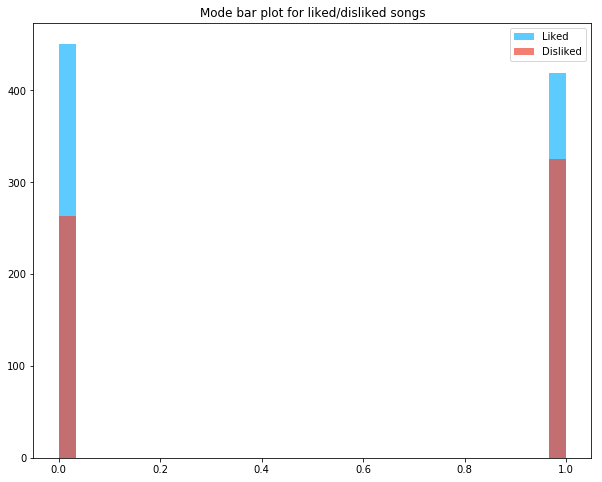


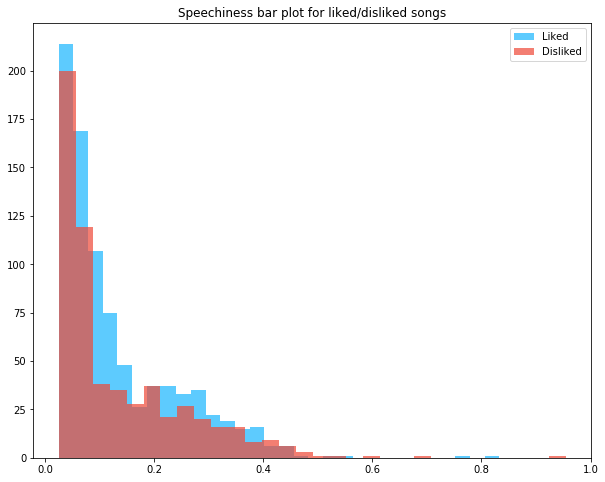


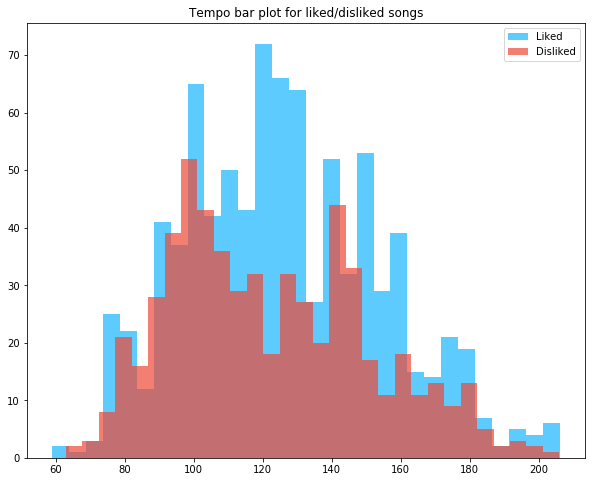


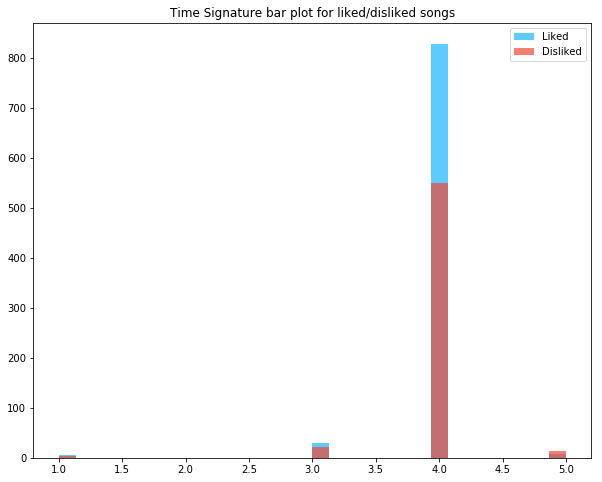


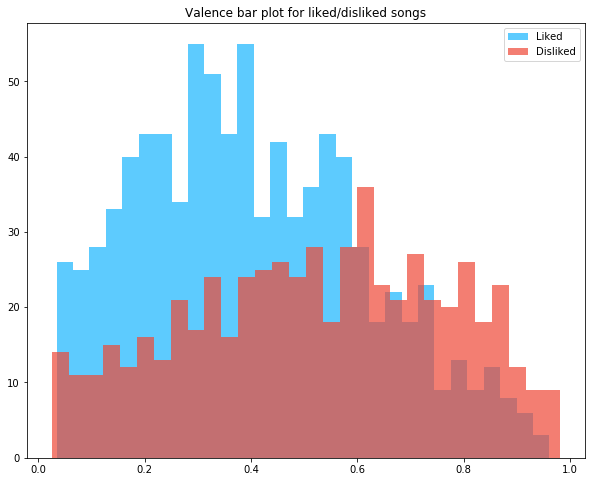












The following section is for the codes for all the classification methods used in the project.

# SVM

## Dataset 1:

*#Importing the libraries that will be used in this program.*

**from** **matplotlib** **import** pyplot **as** plt

**import** **pandas** **as** **pd**

**import** **seaborn** **as** **sn**

**from** **sklearn.svm** **import** SVC *#For the SVM classifier*

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn.metrics** **import** accuracy\_score, confusion\_matrix

*#This command is used to plot figures in our jupyter notebook*

%**matplotlib** inline

*#Importing the data into our program.*

data = pd.read\_csv('Dataset1.csv')

*#Creating a dataframe for the data using pandas library.*

df = pd.DataFrame(data)

*#Overview of the dataframe. head() command only gives the values for the first 5 rows.*

df.head()

| **acousticness** | **danceability** | **duration\_ms** | **energy** | **instrumentalness** | **key** | **liveness** | **loudness** | **mode** | **speechiness** | **tempo** | **valence** | **target** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0.0133 | 0.773 | 204014 | 0.507 | 0.26 | 1 | 0.170 | -9.584 | 1 | 0.0505 | 126.043 | 0.304 | 1 |
| **1** | 0.1750 | 0.661 | 265947 | 0.748 | 0.00 | 2 | 0.254 | -4.809 | 0 | 0.2060 | 171.983 | 0.509 | 1 |
| **2** | 0.1280 | 0.755 | 141827 | 0.772 | 0.00 | 6 | 0.157 | -5.585 | 1 | 0.4000 | 132.906 | 0.678 | 1 |
| **3** | 0.0169 | 0.852 | 216600 | 0.752 | 0.00 | 1 | 0.394 | -5.793 | 1 | 0.0597 | 89.950 | 0.389 | 1 |
| **4** | 0.1450 | 0.808 | 210800 | 0.745 | 0.00 | 10 | 0.292 | -5.260 | 0 | 0.3420 | 165.995 | 0.829 | 1 |

*#Assigning all the features to the variable X.*

X = df.drop(['target'], axis='columns')

X.head()

*#Assigning the labaelled class to variable y.*

y = df.target

*#Splitting the data into train and test set*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.2, random\_state=17)

len(X\_train)

len(X\_test)

*#The model.*

*#Following models are commented to apply a single parameter settings at a time. Uncomment to see how they work.*

*#model = SVC()*

model = SVC(C=10)

*#model = SVC(C=10, kernel='poly')*

*#To fit the model*

model.fit(X\_train, y\_train)

*#To evaluate the accuracy*

model.score(X\_test, y\_test)

**Confusion Matrix**

y\_pred = model.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm

*#To plot the confusion matrix*

plt.figure(figsize=(10,7))

sn.heatmap(cm, annot = **True**,fmt='.2f')

plt.xlabel('Predicted')

plt.ylabel('Truth')

## Dataset 2:

**from** **matplotlib** **import** pyplot **as** plt

**from** **sklearn.svm** **import** SVC

**from** **sklearn.model\_selection** **import** train\_test\_split

**import** **pandas** **as** **pd**

**import** **seaborn** **as** **sn**

**from** **sklearn.metrics** **import** accuracy\_score, confusion\_matrix

%**matplotlib** inline

data = pd.read\_csv('Dataset2.csv')

df = pd.DataFrame(data)

X = df.drop(['target'], axis='columns')

*#X.head()*

y = df.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.2, random\_state=17)

*#len(X\_train)*

*#len(X\_test)*

*#model = SVC()*

model = SVC(C=10)

*#model = SVC(C=10, kernel='poly')*

model.fit(X\_train, y\_train)

model.score(X\_test, y\_test)

y\_pred = model.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm

plt.figure(figsize=(10,7))

sn.heatmap(cm, annot = **True**,fmt='.2f')

plt.xlabel('Predicted')

plt.ylabel('Truth')

## Dataset 3:

**from** **matplotlib** **import** pyplot **as** plt

**from** **sklearn.svm** **import** SVC

**from** **sklearn.model\_selection** **import** train\_test\_split

**import** **pandas** **as** **pd**

**import** **seaborn** **as** **sn**

**from** **sklearn.metrics** **import** accuracy\_score, confusion\_matrix

%**matplotlib** inline

data = pd.read\_csv('Dataset3.csv')

df = pd.DataFrame(data)

*#df.head()*

X = df.drop(['target'], axis='columns')

*#X.head()*

y = df.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.2, random\_state=17)

*#len(X\_train)*

*#len(X\_test)*

*#model = SVC()*

model = SVC(C=10)

*#model = SVC(C=10, kernel='poly')*

model.fit(X\_train, y\_train)

model.score(X\_test, y\_test)

y\_pred = model.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm

plt.figure(figsize=(10,7))

sn.heatmap(cm, annot = **True**,fmt='.2f')

plt.xlabel('Predicted')

plt.ylabel('Truth')

## Dataset 4:

**from** **matplotlib** **import** pyplot **as** plt

**from** **sklearn.svm** **import** SVC

**from** **sklearn.model\_selection** **import** train\_test\_split

**import** **pandas** **as** **pd**

**import** **seaborn** **as** **sn**

**from** **sklearn.metrics** **import** accuracy\_score, confusion\_matrix

%**matplotlib** inline

data = pd.read\_csv('Dataset4.csv')

df = pd.DataFrame(data)

*#df.head()*

X = df.drop(['target'], axis='columns')

*#X.head()*

y = df.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.2, random\_state=17)

*#len(X\_train)*

*#len(X\_test)*

*#model = SVC()*

model = SVC(C=10)

*#model = SVC(C=10, kernel='poly')*

model.fit(X\_train, y\_train)

model.score(X\_test, y\_test)

y\_pred = model.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm

plt.figure(figsize=(10,7))

sn.heatmap(cm, annot = **True**,fmt='.2f')

plt.xlabel('Predicted')

plt.ylabel('Truth')

# Naïve Bayes:

## Dataset 1:

*#Importing the libraries that will be used in this program.*

**from** **matplotlib** **import** pyplot **as** plt

**from** **sklearn.naive\_bayes** **import** BernoulliNB *#For Bernoulli classifier*

**from** **sklearn.naive\_bayes** **import** GaussianNB *#For Gaussian classifier*

**from** **sklearn.naive\_bayes** **import** MultinomialNB

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn.metrics** **import** accuracy\_score, confusion\_matrix, classification\_report

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **seaborn** **as** **sn**

*#This command is used to plot figures in our jupyter notebook*

%**matplotlib** inline

*#Importing the data into our program.*

data = pd.read\_csv('Dataset1.csv')

*#Creating a dataframe for the data using pandas library.*

df = pd.DataFrame(data)

*#Overview of the dataframe. head() command only gives the values for the first 5 rows.*

df.head()

*#Assigning all the features to the variable X.*

X = df.drop(['target'], axis='columns')

X.head()

*#Assigning the labaelled class to variable y.*

y = df.target

*#Splitting the data into train and test set*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state = 17)

len(X\_train)

len(X\_test)

*#The model.*

model1 = BernoulliNB(binarize = 0.1) *#So that our function binarizes the data.*

*#To fit the model*

model1.fit(X\_train, y\_train)

*#To evaluate the accuracy*

model1.score(X\_test, y\_test)

*#Confusion Matrix for Bernoulli NB*

y\_pred = model1.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm

*#To plot the confusion matrix*

plt.figure(figsize=(10,7))

sn.heatmap(cm, annot = **True**,fmt='.2f')

plt.xlabel('Predicted')

plt.ylabel('Truth')

*#Model 2*

*#The model.*

model2 = GaussianNB()

*#To fit the model*

model2.fit(X\_train, y\_train)

*#To evaluate the accuracy*

model2.score(X\_test, y\_test)

*#Confusion Matrix for Gaussian NB*

y\_pred = model2.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm

*#To plot the confusion matrix*

plt.figure(figsize=(10,7))

sn.heatmap(cm, annot = **True**,fmt='.2f')

plt.xlabel('Predicted')

plt.ylabel('Truth')

## Dataset 2:

*#Importing the libraries that will be used in this program.*

**from** **matplotlib** **import** pyplot **as** plt

**from** **sklearn.naive\_bayes** **import** BernoulliNB *#For Bernoulli classifier*

**from** **sklearn.naive\_bayes** **import** GaussianNB *#For Gaussian classifier*

**from** **sklearn.naive\_bayes** **import** MultinomialNB

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn.metrics** **import** accuracy\_score, confusion\_matrix, classification\_report

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **seaborn** **as** **sn**

*#This command is used to plot figures in our jupyter notebook*

%**matplotlib** inline

*#Importing the data into our program.*

data = pd.read\_csv('Dataset2.csv')

*#Creating a dataframe for the data using pandas library.*

df = pd.DataFrame(data)

*#Overview of the dataframe. head() command only gives the values for the first 5 rows.*

df.head()

*#Assigning all the features to the variable X.*

X = df.drop(['target'], axis='columns')

X.head()

*#Assigning the labaelled class to variable y.*

y = df.target

*#Splitting the data into train and test set*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state = 17)

len(X\_train)

len(X\_test)

*#The model.*

model1 = BernoulliNB(binarize = 0.1) *#So that our function binarizes the data.*

*#To fit the model*

model1.fit(X\_train, y\_train)

*#To evaluate the accuracy*

model1.score(X\_test, y\_test)

*#Confusion Matrix for Bernoulli NB*

y\_pred = model1.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm

*#To plot the confusion matrix*

plt.figure(figsize=(10,7))

sn.heatmap(cm, annot = **True**,fmt='.2f')

plt.xlabel('Predicted')

plt.ylabel('Truth')

*#Model 2*

*#The model.*

model2 = GaussianNB()

*#To fit the model*

model2.fit(X\_train, y\_train)

*#To evaluate the accuracy*

model2.score(X\_test, y\_test)

*#Confusion Matrix for Gaussian NB*

y\_pred = model2.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm

*#To plot the confusion matrix*

plt.figure(figsize=(10,7))

sn.heatmap(cm, annot = **True**,fmt='.2f')

plt.xlabel('Predicted')

plt.ylabel('Truth')

## Dataset 3:

*#Importing the libraries that will be used in this program.*

**from** **matplotlib** **import** pyplot **as** plt

**from** **sklearn.naive\_bayes** **import** BernoulliNB *#For Bernoulli classifier*

**from** **sklearn.naive\_bayes** **import** GaussianNB *#For Gaussian classifier*

**from** **sklearn.naive\_bayes** **import** MultinomialNB

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn.metrics** **import** accuracy\_score, confusion\_matrix, classification\_report

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **seaborn** **as** **sn**

*#This command is used to plot figures in our jupyter notebook*

%**matplotlib** inline

*#Importing the data into our program.*

data = pd.read\_csv('Dataset3.csv')

*#Creating a dataframe for the data using pandas library.*

df = pd.DataFrame(data)

*#Overview of the dataframe. head() command only gives the values for the first 5 rows.*

df.head()

*#Assigning all the features to the variable X.*

X = df.drop(['target'], axis='columns')

X.head()

*#Assigning the labaelled class to variable y.*

y = df.target

*#Splitting the data into train and test set*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state = 17)

len(X\_train)

len(X\_test)

*#The model.*

model1 = BernoulliNB(binarize = 0.1) *#So that our function binarizes the data.*

*#To fit the model*

model1.fit(X\_train, y\_train)

*#To evaluate the accuracy*

model1.score(X\_test, y\_test)

*#Confusion Matrix for Bernoulli NB*

y\_pred = model1.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm

*#To plot the confusion matrix*

plt.figure(figsize=(10,7))

sn.heatmap(cm, annot = **True**,fmt='.2f')

plt.xlabel('Predicted')

plt.ylabel('Truth')

*#Model 2*

*#The model.*

model2 = GaussianNB()

*#To fit the model*

model2.fit(X\_train, y\_train)

*#To evaluate the accuracy*

model2.score(X\_test, y\_test)

*#Confusion Matrix for Gaussian NB*

y\_pred = model2.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm

*#To plot the confusion matrix*

plt.figure(figsize=(10,7))

sn.heatmap(cm, annot = **True**,fmt='.2f')

plt.xlabel('Predicted')

plt.ylabel('Truth')

## Dataset 4:

*#Importing the libraries that will be used in this program.*

**from** **matplotlib** **import** pyplot **as** plt

**from** **sklearn.naive\_bayes** **import** BernoulliNB *#For Bernoulli classifier*

**from** **sklearn.naive\_bayes** **import** GaussianNB *#For Gaussian classifier*

**from** **sklearn.naive\_bayes** **import** MultinomialNB

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn.metrics** **import** accuracy\_score, confusion\_matrix, classification\_report

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **seaborn** **as** **sn**

*#This command is used to plot figures in our jupyter notebook*

%**matplotlib** inline

*#Importing the data into our program.*

data = pd.read\_csv('Dataset4.csv')

*#Creating a dataframe for the data using pandas library.*

df = pd.DataFrame(data)

*#Overview of the dataframe. head() command only gives the values for the first 5 rows.*

df.head()

*#Assigning all the features to the variable X.*

X = df.drop(['target'], axis='columns')

X.head()

*#Assigning the labaelled class to variable y.*

y = df.target

*#Splitting the data into train and test set*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state = 17)

len(X\_train)

len(X\_test)

*#The model.*

model1 = BernoulliNB(binarize = 0.1) *#So that our function binarizes the data.*

*#To fit the model*

model1.fit(X\_train, y\_train)

*#To evaluate the accuracy*

model1.score(X\_test, y\_test)

*#Confusion Matrix for Bernoulli NB*

y\_pred = model1.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm

*#To plot the confusion matrix*

plt.figure(figsize=(10,7))

sn.heatmap(cm, annot = **True**,fmt='.2f')

plt.xlabel('Predicted')

plt.ylabel('Truth')

*#Model 2*

*#The model.*

model2 = GaussianNB()

*#To fit the model*

model2.fit(X\_train, y\_train)

*#To evaluate the accuracy*

model2.score(X\_test, y\_test)

*#Confusion Matrix for Gaussian NB*

y\_pred = model2.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm

*#To plot the confusion matrix*

plt.figure(figsize=(10,7))

sn.heatmap(cm, annot = **True**,fmt='.2f')

plt.xlabel('Predicted')

plt.ylabel('Truth')

# KNN:

## Dataset 1:

*#Importing the libraries that will be used in this program.*

**import** **matplotlib.pyplot** **as** **plt**

**from** **sklearn.neighbors** **import** KNeighborsClassifier *#For KNN classifier*

**from** **sklearn** **import** neighbors, preprocessing

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn** **import** metrics

**from** **sklearn.metrics** **import** accuracy\_score, confusion\_matrix, classification\_report

**import** **seaborn** **as** **sn**

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

*#This command is used to plot figures in our jupyter notebook*

%**matplotlib** inline

*#Importing the data into our program.*

data = pd.read\_csv('Dataset1.csv')

*#Creating a dataframe for the data using pandas library.*

df = pd.DataFrame(data)

*#Overview of the dataframe. head() command only gives the values for the first 5 rows.*

df.head()

*#Assigning all the features to the variable X.*

X = df.drop(['target'], axis='columns')

X.head()

*#Assigning the labaelled class to variable y.*

y = df.target

*#Splitting the data into train and test set*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state = 17)

print(X\_train.shape,X\_test.shape)

*#The model*

model = neighbors.KNeighborsClassifier(n\_neighbors=12, p=1)

*#Fitting the model*

model.fit(X\_train, y\_train)

*#To get the classification report such as precision and recall, which are useful for us in this project.*

y\_expect = y\_test

y\_pred = model.predict(X\_test)

print(metrics.classification\_report(y\_expect, y\_pred))

*#To check the accuracy*

model.score(X\_test, y\_test)

*#Confusion Matrix*

y\_pred = model.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm

*#To plot the confusion matrix*

plt.figure(figsize=(10,7))

sn.heatmap(cm, annot = **True**,fmt='.2f')

plt.xlabel('Predicted')

plt.ylabel('Truth')

## Dataset 2:

*#Importing the libraries that will be used in this program.*

**import** **matplotlib.pyplot** **as** **plt**

**from** **sklearn.neighbors** **import** KNeighborsClassifier *#For KNN classifier*

**from** **sklearn** **import** neighbors, preprocessing

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn** **import** metrics

**from** **sklearn.metrics** **import** accuracy\_score, confusion\_matrix, classification\_report

**import** **seaborn** **as** **sn**

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

*#This command is used to plot figures in our jupyter notebook*

%**matplotlib** inline

*#Importing the data into our program.*

data = pd.read\_csv('Dataset2.csv')

*#Creating a dataframe for the data using pandas library.*

df = pd.DataFrame(data)

*#Overview of the dataframe. head() command only gives the values for the first 5 rows.*

df.head()

*#Assigning all the features to the variable X.*

X = df.drop(['target'], axis='columns')

X.head()

*#Assigning the labaelled class to variable y.*

y = df.target

*#Splitting the data into train and test set*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state = 17)

print(X\_train.shape,X\_test.shape)

*#The model*

model = neighbors.KNeighborsClassifier(n\_neighbors=12, p=1)

*#Fitting the model*

model.fit(X\_train, y\_train)

*#To get the classification report such as precision and recall, which are useful for us in this project.*

y\_expect = y\_test

y\_pred = model.predict(X\_test)

print(metrics.classification\_report(y\_expect, y\_pred))

*#To check the accuracy*

model.score(X\_test, y\_test)

*#Confusion Matrix*

y\_pred = model.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm

*#To plot the confusion matrix*

plt.figure(figsize=(10,7))

sn.heatmap(cm, annot = **True**,fmt='.2f')

plt.xlabel('Predicted')

plt.ylabel('Truth')

## Dataset 3:

*#Importing the libraries that will be used in this program.*

**import** **matplotlib.pyplot** **as** **plt**

**from** **sklearn.neighbors** **import** KNeighborsClassifier *#For KNN classifier*

**from** **sklearn** **import** neighbors, preprocessing

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn** **import** metrics

**from** **sklearn.metrics** **import** accuracy\_score, confusion\_matrix, classification\_report

**import** **seaborn** **as** **sn**

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

*#This command is used to plot figures in our jupyter notebook*

%**matplotlib** inline

*#Importing the data into our program.*

data = pd.read\_csv('Dataset3.csv')

*#Creating a dataframe for the data using pandas library.*

df = pd.DataFrame(data)

*#Overview of the dataframe. head() command only gives the values for the first 5 rows.*

df.head()

*#Assigning all the features to the variable X.*

X = df.drop(['target'], axis='columns')

X.head()

*#Assigning the labaelled class to variable y.*

y = df.target

*#Splitting the data into train and test set*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state = 17)

print(X\_train.shape,X\_test.shape)

*#The model*

model = neighbors.KNeighborsClassifier(n\_neighbors=12, p=1)

*#Fitting the model*

model.fit(X\_train, y\_train)

*#To get the classification report such as precision and recall, which are useful for us in this project.*

y\_expect = y\_test

y\_pred = model.predict(X\_test)

print(metrics.classification\_report(y\_expect, y\_pred))

*#To check the accuracy*

model.score(X\_test, y\_test)

*#Confusion Matrix*

y\_pred = model.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm

*#To plot the confusion matrix*

plt.figure(figsize=(10,7))

sn.heatmap(cm, annot = **True**,fmt='.2f')

plt.xlabel('Predicted')

plt.ylabel('Truth')

## Dataset 4:

*#Importing the libraries that will be used in this program.*

**import** **matplotlib.pyplot** **as** **plt**

**from** **sklearn.neighbors** **import** KNeighborsClassifier *#For KNN classifier*

**from** **sklearn** **import** neighbors, preprocessing

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn** **import** metrics

**from** **sklearn.metrics** **import** accuracy\_score, confusion\_matrix, classification\_report

**import** **seaborn** **as** **sn**

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

*#This command is used to plot figures in our jupyter notebook*

%**matplotlib** inline

*#Importing the data into our program.*

data = pd.read\_csv('Dataset4.csv')

*#Creating a dataframe for the data using pandas library.*

df = pd.DataFrame(data)

*#Overview of the dataframe. head() command only gives the values for the first 5 rows.*

df.head()

*#Assigning all the features to the variable X.*

X = df.drop(['target'], axis='columns')

X.head()

*#Assigning the labaelled class to variable y.*

y = df.target

*#Splitting the data into train and test set*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state = 17)

print(X\_train.shape,X\_test.shape)

*#The model*

model = neighbors.KNeighborsClassifier(n\_neighbors=12, p=1)

*#Fitting the model*

model.fit(X\_train, y\_train)

*#To get the classification report such as precision and recall, which are useful for us in this project.*

y\_expect = y\_test

y\_pred = model.predict(X\_test)

print(metrics.classification\_report(y\_expect, y\_pred))

*#To check the accuracy*

model.score(X\_test, y\_test)

*#Confusion Matrix*

y\_pred = model.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm

*#To plot the confusion matrix*

plt.figure(figsize=(10,7))

sn.heatmap(cm, annot = **True**,fmt='.2f')

plt.xlabel('Predicted')

plt.ylabel('Truth')

# For Random Forest:

## Dataset 1:

*#Importing the libraries that will be used in this program.*

**import** **matplotlib.pyplot** **as** **plt**

**from** **sklearn.ensemble** **import** RandomForestClassifier *#For Random Forest Classifier*

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn.metrics** **import** accuracy\_score

**from** **sklearn.metrics** **import** accuracy\_score, confusion\_matrix, classification\_report

**import** **seaborn** **as** **sn**

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

*#This command is used to plot figures in our jupyter notebook*

%**matplotlib** inline

*#Importing the data into our program.*

data = pd.read\_csv('Dataset1.csv')

*#Creating a dataframe for the data using pandas library.*

df = pd.DataFrame(data)

*#Overview of the dataframe. head() command only gives the values for the first 5 rows.*

df.head()

*#Assigning all the features to the variable X.*

X = df.drop(['target'], axis='columns')

X.head()

*#Assigning the labaelled class to variable y.*

y = df.target

*#Splitting the data into train and test set*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state = 17)

len(X\_train)

len(X\_test)

*#The model*

model = RandomForestClassifier(n\_estimators=300, max\_features=0.7, min\_samples\_leaf=2)

*#Fitting the model*

model.fit(X\_train, y\_train)

*#TO check the accuracy*

model.score(X\_test, y\_test)

*# Confusion Matrix*

y\_pred = model.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm

*#To plot the confusion matrix*

plt.figure(figsize=(10,7))

sn.heatmap(cm, annot = **True**,fmt='.2f')

plt.xlabel('Predicted')

plt.ylabel('Truth')

## Dataset 2:

*#Importing the libraries that will be used in this program.*

**import** **matplotlib.pyplot** **as** **plt**

**from** **sklearn.ensemble** **import** RandomForestClassifier *#For Random Forest Classifier*

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn.metrics** **import** accuracy\_score

**from** **sklearn.metrics** **import** accuracy\_score, confusion\_matrix, classification\_report

**import** **seaborn** **as** **sn**

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

*#This command is used to plot figures in our jupyter notebook*

%**matplotlib** inline

*#Importing the data into our program.*

data = pd.read\_csv('Dataset2.csv')

*#Creating a dataframe for the data using pandas library.*

df = pd.DataFrame(data)

*#Overview of the dataframe. head() command only gives the values for the first 5 rows.*

df.head()

*#Assigning all the features to the variable X.*

X = df.drop(['target'], axis='columns')

X.head()

*#Assigning the labaelled class to variable y.*

y = df.target

*#Splitting the data into train and test set*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state = 17)

len(X\_train)

len(X\_test)

*#The model*

model = RandomForestClassifier(n\_estimators=300, max\_features=0.7, min\_samples\_leaf=2)

*#Fitting the model*

model.fit(X\_train, y\_train)

*#TO check the accuracy*

model.score(X\_test, y\_test)

*# Confusion Matrix*

y\_pred = model.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm

*#To plot the confusion matrix*

plt.figure(figsize=(10,7))

sn.heatmap(cm, annot = **True**,fmt='.2f')

plt.xlabel('Predicted')

plt.ylabel('Truth')

## Dataset 3:

*#Importing the libraries that will be used in this program.*

**import** **matplotlib.pyplot** **as** **plt**

**from** **sklearn.ensemble** **import** RandomForestClassifier *#For Random Forest Classifier*

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn.metrics** **import** accuracy\_score

**from** **sklearn.metrics** **import** accuracy\_score, confusion\_matrix, classification\_report

**import** **seaborn** **as** **sn**

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

*#This command is used to plot figures in our jupyter notebook*

%**matplotlib** inline

*#Importing the data into our program.*

data = pd.read\_csv('Dataset3.csv')

*#Creating a dataframe for the data using pandas library.*

df = pd.DataFrame(data)

*#Overview of the dataframe. head() command only gives the values for the first 5 rows.*

df.head()

*#Assigning all the features to the variable X.*

X = df.drop(['target'], axis='columns')

X.head()

*#Assigning the labaelled class to variable y.*

y = df.target

*#Splitting the data into train and test set*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state = 17)

len(X\_train)

len(X\_test)

*#The model*

model = RandomForestClassifier(n\_estimators=300, max\_features=0.7, min\_samples\_leaf=2)

*#Fitting the model*

model.fit(X\_train, y\_train)

*#TO check the accuracy*

model.score(X\_test, y\_test)

*# Confusion Matrix*

y\_pred = model.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm

*#To plot the confusion matrix*

plt.figure(figsize=(10,7))

sn.heatmap(cm, annot = **True**,fmt='.2f')

plt.xlabel('Predicted')

plt.ylabel('Truth')

## Dataset 4:

*#Importing the libraries that will be used in this program.*

**import** **matplotlib.pyplot** **as** **plt**

**from** **sklearn.ensemble** **import** RandomForestClassifier *#For Random Forest Classifier*

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn.metrics** **import** accuracy\_score

**from** **sklearn.metrics** **import** accuracy\_score, confusion\_matrix, classification\_report

**import** **seaborn** **as** **sn**

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

*#This command is used to plot figures in our jupyter notebook*

%**matplotlib** inline

*#Importing the data into our program.*

data = pd.read\_csv('Dataset4.csv')

*#Creating a dataframe for the data using pandas library.*

df = pd.DataFrame(data)

*#Overview of the dataframe. head() command only gives the values for the first 5 rows.*

df.head()

*#Assigning all the features to the variable X.*

X = df.drop(['target'], axis='columns')

X.head()

*#Assigning the labaelled class to variable y.*

y = df.target

*#Splitting the data into train and test set*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state = 17)

len(X\_train)

len(X\_test)

*#The model*

model = RandomForestClassifier(n\_estimators=300, max\_features=0.7, min\_samples\_leaf=2)

*#Fitting the model*

model.fit(X\_train, y\_train)

*#TO check the accuracy*

model.score(X\_test, y\_test)

*# Confusion Matrix*

y\_pred = model.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

cm

*#To plot the confusion matrix*

plt.figure(figsize=(10,7))

sn.heatmap(cm, annot = **True**,fmt='.2f')

plt.xlabel('Predicted')

plt.ylabel('Truth')

Neural Network:  
Dataset 1:

*#Importing the libraries that will be used in this program.*

**import** **pandas**

**from** **keras.models** **import** Sequential

**from** **keras.layers** **import** Dense *#Importing Dense layer which we will mostly use.*

**from** **keras.wrappers.scikit\_learn** **import** KerasClassifier

**from** **sklearn.model\_selection** **import** cross\_val\_score

**from** **sklearn.preprocessing** **import** LabelEncoder

**from** **sklearn.model\_selection** **import** StratifiedKFold

**from** **sklearn.preprocessing** **import** StandardScaler

**from** **sklearn.pipeline** **import** Pipeline

*# load dataset*

dataframe = pandas.read\_csv("Dataset1.csv")

*#Printing the values in the dataframe*

dataset = dataframe.values

*# split into input (X) and output (Y) variables*

X = dataset[:,:12].astype(float) *#All the features apart from the label class is assigned to the variable X*

Y = dataset[:,12] *#Label values assignment to variable Y*

*# Neural Network with two hidden layers*

*# encode class values as integers*

encoder = LabelEncoder()

encoder.fit(Y)

encoded\_Y = encoder.transform(Y)

*#Creating model*

**def** create\_baseline():

*# create model*

model = Sequential()

model.add(Dense(60, input\_dim=12, activation='relu')) *#First Dense layer with 60 neurons*

model.add(Dense(30, activation='relu')) *#Second Dense layer with 30 neurons*

model.add(Dense(1, activation='sigmoid')) *#Output Layer*

*# Compile model*

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

**return** model

*# evaluate baseline model with standardized dataset*

estimators = []

estimators.append(('standardize', StandardScaler()))

estimators.append(('mlp', KerasClassifier(build\_fn=create\_baseline, epochs=100, batch\_size=5, verbose=0)))

pipeline = Pipeline(estimators)

*#Using 10 split K-Fold validation*

kfold = StratifiedKFold(n\_splits=10, shuffle=**True**)

results = cross\_val\_score(pipeline, X, encoded\_Y, cv=kfold)

print("Standardized: **%.2f%%** (**%.2f%%**)" % (results.mean()\*100, results.std()\*100))

*# Neural Network with 1 hidden layer of 30 neurons*

*# encode class values as integers*

encoder = LabelEncoder()

encoder.fit(Y)

encoded\_Y = encoder.transform(Y)

*# baseline model*

**def** create\_baseline():

*# create model*

model = Sequential()

model.add(Dense(30, input\_dim=12, activation='relu')) *#First Dense layer with 30 neurons*

model.add(Dense(1, activation='sigmoid')) *#Output layer*

*# Compile model*

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

**return** model

*# evaluate baseline model with standardized dataset*

estimators = []

estimators.append(('standardize', StandardScaler()))

estimators.append(('mlp', KerasClassifier(build\_fn=create\_baseline, epochs=100, batch\_size=5, verbose=0)))

pipeline = Pipeline(estimators)

*#Using 10 split K-Fold validation*

kfold = StratifiedKFold(n\_splits=10, shuffle=**True**)

results = cross\_val\_score(pipeline, X, encoded\_Y, cv=kfold)

print("Standardized: **%.2f%%** (**%.2f%%**)" % (results.mean()\*100, results.std()\*100))

*# Neural Network with one hidden layer of 60 neurons*

*# encode class values as integers*

encoder = LabelEncoder()

encoder.fit(Y)

encoded\_Y = encoder.transform(Y)

*# baseline model*

**def** create\_baseline():

*# create model*

model = Sequential()

model.add(Dense(60, input\_dim=12, activation='relu')) *#First Dense layer with 60 neurons*

model.add(Dense(1, activation='sigmoid')) *#Output layer*

*# Compile model*

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

**return** model

*# evaluate baseline model with standardized dataset*

estimators = []

estimators.append(('standardize', StandardScaler()))

estimators.append(('mlp', KerasClassifier(build\_fn=create\_baseline, epochs=100, batch\_size=5, verbose=0)))

pipeline = Pipeline(estimators)

*#Using 10 split K-Fold validation*

kfold = StratifiedKFold(n\_splits=10, shuffle=**True**)

results = cross\_val\_score(pipeline, X, encoded\_Y, cv=kfold)

print("Standardized: **%.2f%%** (**%.2f%%**)" % (results.mean()\*100, results.std()\*100))

## Dataset 2:

*#Importing the libraries that will be used in this program.*

**import** **pandas**

**from** **keras.models** **import** Sequential

**from** **keras.layers** **import** Dense *#Importing Dense layer which we will mostly use.*

**from** **keras.wrappers.scikit\_learn** **import** KerasClassifier

**from** **sklearn.model\_selection** **import** cross\_val\_score

**from** **sklearn.preprocessing** **import** LabelEncoder

**from** **sklearn.model\_selection** **import** StratifiedKFold

**from** **sklearn.preprocessing** **import** StandardScaler

**from** **sklearn.pipeline** **import** Pipeline

*# load dataset*

dataframe = pandas.read\_csv("Dataset2.csv")

*#Printing the values in the dataframe*

dataset = dataframe.values

*# split into input (X) and output (Y) variables*

X = dataset[:,:13].astype(float) *#All the features apart from the label class is assigned to the variable X*

Y = dataset[:,13] *#Label values assignment to variable Y*

*# Neural Network with two hidden layers*

*# encode class values as integers*

encoder = LabelEncoder()

encoder.fit(Y)

encoded\_Y = encoder.transform(Y)

*#Creating model*

**def** create\_baseline():

*# create model*

model = Sequential()

model.add(Dense(60, input\_dim=13, activation='relu')) *#First Dense layer with 60 neurons*

model.add(Dense(30, activation='relu')) *#Second Dense layer with 30 neurons*

model.add(Dense(1, activation='sigmoid')) *#Output Layer*

*# Compile model*

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

**return** model

*# evaluate baseline model with standardized dataset*

estimators = []

estimators.append(('standardize', StandardScaler()))

estimators.append(('mlp', KerasClassifier(build\_fn=create\_baseline, epochs=100, batch\_size=5, verbose=0)))

pipeline = Pipeline(estimators)

*#Using 10 split K-Fold validation*

kfold = StratifiedKFold(n\_splits=10, shuffle=**True**)

results = cross\_val\_score(pipeline, X, encoded\_Y, cv=kfold)

print("Standardized: **%.2f%%** (**%.2f%%**)" % (results.mean()\*100, results.std()\*100))

*# Neural Network with 1 hidden layer of 30 neurons*

*# encode class values as integers*

encoder = LabelEncoder()

encoder.fit(Y)

encoded\_Y = encoder.transform(Y)

*# baseline model*

**def** create\_baseline():

*# create model*

model = Sequential()

model.add(Dense(30, input\_dim=13, activation='relu')) *#First Dense layer with 30 neurons*

model.add(Dense(1, activation='sigmoid')) *#Output layer*

*# Compile model*

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

**return** model

*# evaluate baseline model with standardized dataset*

estimators = []

estimators.append(('standardize', StandardScaler()))

estimators.append(('mlp', KerasClassifier(build\_fn=create\_baseline, epochs=100, batch\_size=5, verbose=0)))

pipeline = Pipeline(estimators)

*#Using 10 split K-Fold validation*

kfold = StratifiedKFold(n\_splits=10, shuffle=**True**)

results = cross\_val\_score(pipeline, X, encoded\_Y, cv=kfold)

print("Standardized: **%.2f%%** (**%.2f%%**)" % (results.mean()\*100, results.std()\*100))

*# Neural Network with one hidden layer of 60 neurons*

*# encode class values as integers*

encoder = LabelEncoder()

encoder.fit(Y)

encoded\_Y = encoder.transform(Y)

*# baseline model*

**def** create\_baseline():

*# create model*

model = Sequential()

model.add(Dense(60, input\_dim=13, activation='relu')) *#First Dense layer with 60 neurons*

model.add(Dense(1, activation='sigmoid')) *#Output layer*

*# Compile model*

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

**return** model

*# evaluate baseline model with standardized dataset*

estimators = []

estimators.append(('standardize', StandardScaler()))

estimators.append(('mlp', KerasClassifier(build\_fn=create\_baseline, epochs=100, batch\_size=5, verbose=0)))

pipeline = Pipeline(estimators)

*#Using 10 split K-Fold validation*

kfold = StratifiedKFold(n\_splits=10, shuffle=**True**)

results = cross\_val\_score(pipeline, X, encoded\_Y, cv=kfold)

print("Standardized: **%.2f%%** (**%.2f%%**)" % (results.mean()\*100, results.std()\*100))

## Dataset 3:

*#Importing the libraries that will be used in this program.*

**import** **pandas**

**from** **keras.models** **import** Sequential

**from** **keras.layers** **import** Dense *#Importing Dense layer which we will mostly use.*

**from** **keras.wrappers.scikit\_learn** **import** KerasClassifier

**from** **sklearn.model\_selection** **import** cross\_val\_score

**from** **sklearn.preprocessing** **import** LabelEncoder

**from** **sklearn.model\_selection** **import** StratifiedKFold

**from** **sklearn.preprocessing** **import** StandardScaler

**from** **sklearn.pipeline** **import** Pipeline

*# load dataset*

dataframe = pandas.read\_csv("Dataset3.csv")

*#Printing the values in the dataframe*

dataset = dataframe.values

*# split into input (X) and output (Y) variables*

X = dataset[:,:13].astype(float) *#All the features apart from the label class is assigned to the variable X*

Y = dataset[:,13] *#Label values assignment to variable Y*

*# Neural Network with two hidden layers*

*# encode class values as integers*

encoder = LabelEncoder()

encoder.fit(Y)

encoded\_Y = encoder.transform(Y)

*#Creating model*

**def** create\_baseline():

*# create model*

model = Sequential()

model.add(Dense(60, input\_dim=13, activation='relu')) *#First Dense layer with 60 neurons*

model.add(Dense(30, activation='relu')) *#Second Dense layer with 30 neurons*

model.add(Dense(1, activation='sigmoid')) *#Output Layer*

*# Compile model*

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

**return** model

*# evaluate baseline model with standardized dataset*

estimators = []

estimators.append(('standardize', StandardScaler()))

estimators.append(('mlp', KerasClassifier(build\_fn=create\_baseline, epochs=100, batch\_size=5, verbose=0)))

pipeline = Pipeline(estimators)

*#Using 10 split K-Fold validation*

kfold = StratifiedKFold(n\_splits=10, shuffle=**True**)

results = cross\_val\_score(pipeline, X, encoded\_Y, cv=kfold)

print("Standardized: **%.2f%%** (**%.2f%%**)" % (results.mean()\*100, results.std()\*100))

*# Neural Network with 1 hidden layer of 30 neurons*

*# encode class values as integers*

encoder = LabelEncoder()

encoder.fit(Y)

encoded\_Y = encoder.transform(Y)

*# baseline model*

**def** create\_baseline():

*# create model*

model = Sequential()

model.add(Dense(30, input\_dim=13, activation='relu')) *#First Dense layer with 30 neurons*

model.add(Dense(1, activation='sigmoid')) *#Output layer*

*# Compile model*

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

**return** model

*# evaluate baseline model with standardized dataset*

estimators = []

estimators.append(('standardize', StandardScaler()))

estimators.append(('mlp', KerasClassifier(build\_fn=create\_baseline, epochs=100, batch\_size=5, verbose=0)))

pipeline = Pipeline(estimators)

*#Using 10 split K-Fold validation*

kfold = StratifiedKFold(n\_splits=10, shuffle=**True**)

results = cross\_val\_score(pipeline, X, encoded\_Y, cv=kfold)

print("Standardized: **%.2f%%** (**%.2f%%**)" % (results.mean()\*100, results.std()\*100))

*# Neural Network with one hidden layer of 60 neurons*

*# encode class values as integers*

encoder = LabelEncoder()

encoder.fit(Y)

encoded\_Y = encoder.transform(Y)

*# baseline model*

**def** create\_baseline():

*# create model*

model = Sequential()

model.add(Dense(60, input\_dim=13, activation='relu')) *#First Dense layer with 60 neurons*

model.add(Dense(1, activation='sigmoid')) *#Output layer*

*# Compile model*

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

**return** model

*# evaluate baseline model with standardized dataset*

estimators = []

estimators.append(('standardize', StandardScaler()))

estimators.append(('mlp', KerasClassifier(build\_fn=create\_baseline, epochs=100, batch\_size=5, verbose=0)))

pipeline = Pipeline(estimators)

*#Using 10 split K-Fold validation*

kfold = StratifiedKFold(n\_splits=10, shuffle=**True**)

results = cross\_val\_score(pipeline, X, encoded\_Y, cv=kfold)

print("Standardized: **%.2f%%** (**%.2f%%**)" % (results.mean()\*100, results.std()\*100))

## Dataset 4:

*#Importing the libraries that will be used in this program.*

**import** **pandas**

**from** **keras.models** **import** Sequential

**from** **keras.layers** **import** Dense *#Importing Dense layer which we will mostly use.*

**from** **keras.wrappers.scikit\_learn** **import** KerasClassifier

**from** **sklearn.model\_selection** **import** cross\_val\_score

**from** **sklearn.preprocessing** **import** LabelEncoder

**from** **sklearn.model\_selection** **import** StratifiedKFold

**from** **sklearn.preprocessing** **import** StandardScaler

**from** **sklearn.pipeline** **import** Pipeline

*# load dataset*

dataframe = pandas.read\_csv("Dataset4.csv")

*#Printing the values in the dataframe*

dataset = dataframe.values

*# split into input (X) and output (Y) variables*

X = dataset[:,:14].astype(float) *#All the features apart from the label class is assigned to the variable X*

Y = dataset[:,14] *#Label values assignment to variable Y*

*# Neural Network with two hidden layers*

*# encode class values as integers*

encoder = LabelEncoder()

encoder.fit(Y)

encoded\_Y = encoder.transform(Y)

*#Creating model*

**def** create\_baseline():

*# create model*

model = Sequential()

model.add(Dense(60, input\_dim=14, activation='relu')) *#First Dense layer with 60 neurons*

model.add(Dense(30, activation='relu')) *#Second Dense layer with 30 neurons*

model.add(Dense(1, activation='sigmoid')) *#Output Layer*

*# Compile model*

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

**return** model

*# evaluate baseline model with standardized dataset*

estimators = []

estimators.append(('standardize', StandardScaler()))

estimators.append(('mlp', KerasClassifier(build\_fn=create\_baseline, epochs=100, batch\_size=5, verbose=0)))

pipeline = Pipeline(estimators)

*#Using 10 split K-Fold validation*

kfold = StratifiedKFold(n\_splits=10, shuffle=**True**)

results = cross\_val\_score(pipeline, X, encoded\_Y, cv=kfold)

print("Standardized: **%.2f%%** (**%.2f%%**)" % (results.mean()\*100, results.std()\*100))

*# Neural Network with 1 hidden layer of 30 neurons*

*# encode class values as integers*

encoder = LabelEncoder()

encoder.fit(Y)

encoded\_Y = encoder.transform(Y)

*# baseline model*

**def** create\_baseline():

*# create model*

model = Sequential()

model.add(Dense(30, input\_dim=14, activation='relu')) *#First Dense layer with 30 neurons*

model.add(Dense(1, activation='sigmoid')) *#Output layer*

*# Compile model*

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

**return** model

*# evaluate baseline model with standardized dataset*

estimators = []

estimators.append(('standardize', StandardScaler()))

estimators.append(('mlp', KerasClassifier(build\_fn=create\_baseline, epochs=100, batch\_size=5, verbose=0)))

pipeline = Pipeline(estimators)

*#Using 10 split K-Fold validation*

kfold = StratifiedKFold(n\_splits=10, shuffle=**True**)

results = cross\_val\_score(pipeline, X, encoded\_Y, cv=kfold)

print("Standardized: **%.2f%%** (**%.2f%%**)" % (results.mean()\*100, results.std()\*100))

*# Neural Network with one hidden layer of 60 neurons*

*# encode class values as integers*

encoder = LabelEncoder()

encoder.fit(Y)

encoded\_Y = encoder.transform(Y)

*# baseline model*

**def** create\_baseline():

*# create model*

model = Sequential()

model.add(Dense(60, input\_dim=14, activation='relu')) *#First Dense layer with 60 neurons*

model.add(Dense(1, activation='sigmoid')) *#Output layer*

*# Compile model*

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

**return** model

*# evaluate baseline model with standardized dataset*

estimators = []

estimators.append(('standardize', StandardScaler()))

estimators.append(('mlp', KerasClassifier(build\_fn=create\_baseline, epochs=100, batch\_size=5, verbose=0)))

pipeline = Pipeline(estimators)

*#Using 10 split K-Fold validation*

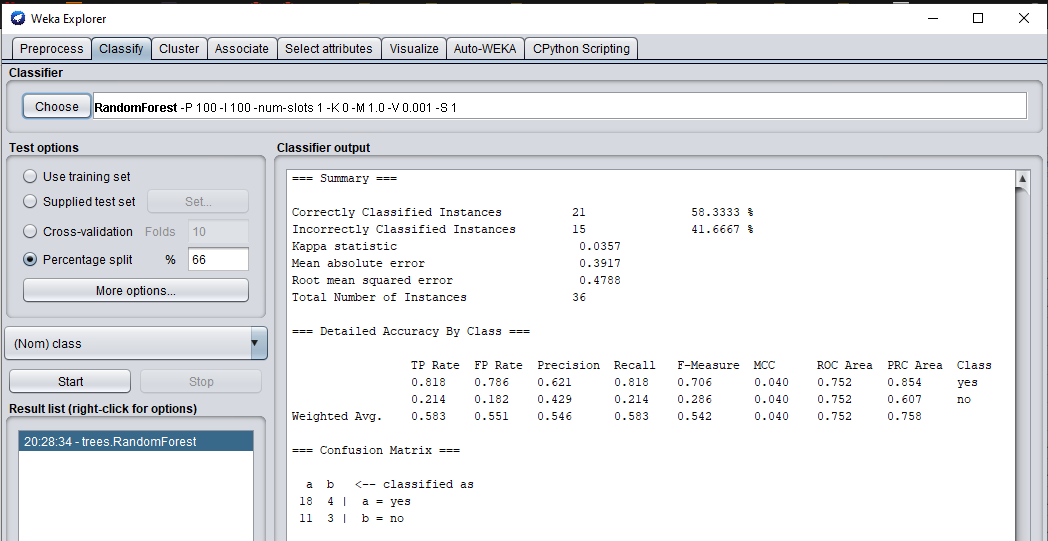
kfold = StratifiedKFold(n\_splits=10, shuffle=**True**)

results = cross\_val\_score(pipeline, X, encoded\_Y, cv=kfold)

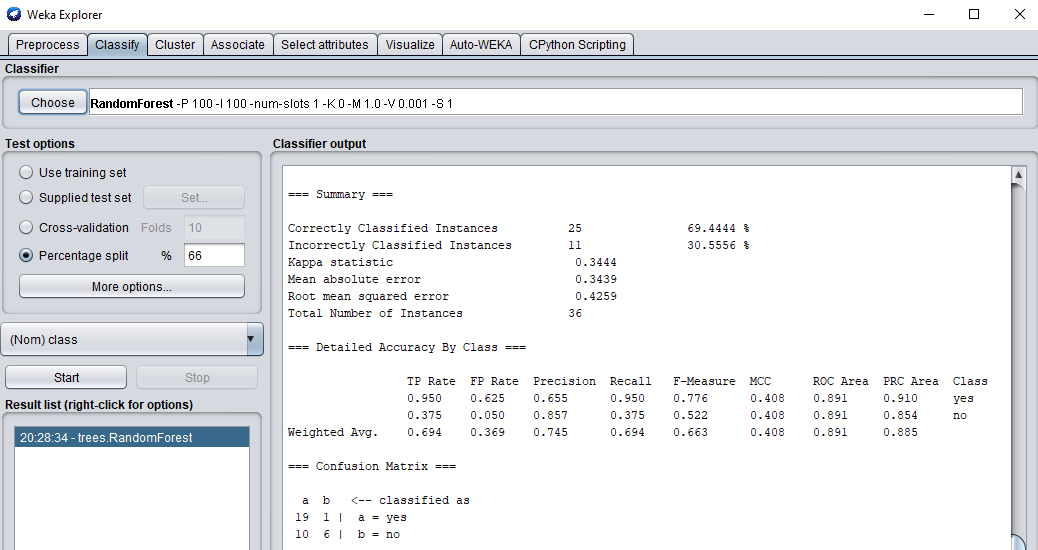
print("Standardized: **%.2f%%** (**%.2f%%**)" % (results.mean()\*100, results.std()\*100))

# Screenshot from Weka with the testing data:

## For Test Data 1:

****

## For Test Data 2:

****