Predicting Anthony Fantano Album Ratings

CA4010 Data Warehousing and Data Mining Report

Student Name(s): Joao Pereira & Conor Joyce
Student ID(s): 19354106 & 19425804
Programme: BSc in Computer Applications, Software Engineering
Lecturer: Mark Roantree
Module Code: CA4010

Table of Contents

1. Declaration

I/We declare that this material, which I/We now submit for assessment, is entirely my/our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my/our work. I/We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I/We have read and understood the Assignment Regulations. I/We have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me/us or any other person for assessment on this or any other course of study.

I/We have read and understood the referencing guidelines found at http://www.dcu.ie/info/regulations/plagiarism.shtml, https://www4.dcu.ie/students/az/plagiarism and/or recommended in the assignment guidelines.

Name(s): Joao Pereira & Conor Joyce
Date: 8/11/2022

2. Introduction

For our Data Warehousing and Data Mining project we decided to take a more fun and personalized approach. We are predicting Anthony Fantano's Album Ratings. Anthony Fantano is an American trending YouTuber who critiques a different music album every day. Each album he reviews ranges from different genres and artists. At the end of the video after an album is analyzed, Anthony shares his personal overall rating of the album from 1 to 10, 1 being the lowest rating and 10 the highest.

With this in mind, we want to predict the score an album would receive if it were to be reviewed by Anthony. We got our dataset from Kaggle but the data originates from Spotify's API. The dataset contains 3024 albums and 36623 tracks in total, so we have tons of data to examine and create an accurate prediction of the score.

The attributes that we will be discussing are all retrieved from Kaggle. Each attribute is based on a scale of 0 to 1, with 0 being low and 1 being high in its category. Although, the only exceptions to this are tempo and loudness which are not measures on the mentioned scale of 0-1 but are values measured in decibels and beats-per-minute respectively. These are not subjective values defined by Spotify like acousticness or danceability but exact and measurable real-world values. The following is a list of attributes we used for our predictions:

- Duration
- Explicit
- Key
- Mode
- Energy
- Instrumentalness
- Loudness
- Speechiness
- Tempo

3. Idea

3.1. Idea Goal

The project's entire idea is to predict what rating an album will receive out of 10 if Anthony Fantano was to upload a video on the album and critique it. Initially, our idea came from endlessly browsing through Kaggle in order to find a dataset/idea that we thought matched our personalities. After a while, we both settled on the dataset named “Anthony Fantano Album Reviews”. Coincidentally we both are fans of Anthony Fantano with shared views on his YouTube content. This dataset caught both of our interests straight away and seemed like an exciting and interesting project to implement.

3.2. Dataset Description

For this project to be feasible, we had to fetch an accurate dataset in order to carry out some form of analysis. As mentioned above, we retrieved the dataset from Kaggle. This dataset contains data spanning from 2010 to 2022. On top of this, the dataset is updated on a quarterly basis to ensure it is at the latest version it can be. The dataset itself consists of two CSV files (tracks and albums respectively) as well as 29 columns in total. Upon originally testing and studying the dataset to fetch some test values, we managed to end up with a complication over the amount of data we were utilizing, therefore, as a team, we progressed forward by cleaning and converting the data into a more appropriate and understandable dataset that fit our needs going forward.

4. Preparation of Data (Data Preprocessing)

4.1 Data Cleaning

The first step that came with Data Cleaning was removing unnecessary attributes, these include:

- Albums.csv
  - Everything except the Spotify ID and Rating
- Tracks.csv
  - Spotify ID
  - Youtube ID
  - Name
  - Preview

The reasoning behind the removal of the attributes was that the IDs and other string metadata were not needed for our prediction. We keep the album's Spotify ID for now so that we can cross-reference an album with its tracks, as each track has an Album ID attribute.

4.2 Data Transformation

```
n = 4 # n is the number of potential ratings, classes
album_ratings = albums.rating.apply(lambda x : round(x / 10 * (n-1)))

album_ratings = album_ratings.map({
    0:'poor',
    1:'fair',
    2:'good',
    3:'excellent'
})

```

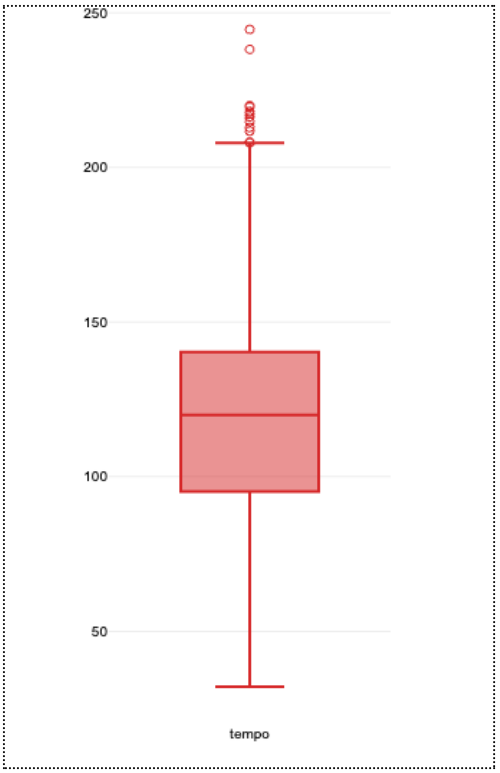
Most of our attributes were already normalised on a scale from 0-1 so there was no need to transform the data in these attributes. Our rating scores were on a scale from 0-10; meaning 11 different potential output classifications. Keeping this would likely result in very poor accuracy in our predictions, therefore we normalised the scores down to a scale of 0-3 - where 0 is poor, 1 is fair, 2 is good and 3 is excellent.

```
tracks_copy = tracks.copy()
overall_averages = pd.DataFrame()
for index, row in albums.iterrows():
    album_tracks = tracks_copy.loc[tracks["album_id"] == row["spotify_id"]]

    # remove albums with zero tracks after purging outliers
    if len(album_tracks) == 0:
        albums = albums[albums["spotify_id"] != row["spotify_id"]]

    album_tracks = album_tracks.copy().drop(["album_id"], axis=1)
    album_averages = album_tracks.mean(numeric_only=True)
    overall_averages = pd.concat([overall_averages,
        pd.DataFrame({k:[v] for k,v in album_averages.items()})],
        ignore_index=True)

```

Another transformation we performed was attribute construction. For every album reviewed, we got the metadata for every track within that album by cross-referencing the IDs. We then used the data for every track to calculate the mean for every attribute on those tracks. For example, the attribute “explicit” would be either 1 or 0 for every track, and we would calculate the mean “explicit” attribute for a whole album - a mean of 0.2 would infer that 1 in 5 songs in an album were explicit. We felt this was a good solution to simplifying the data we had while still providing a good source of data for the prediction algorithms to perform effectively.

4.2 Data Reduction of Noise (Outliers)

Throughout our dataset, we have discovered hundreds of outliers, especially within the Tempo attribute.



As displayed in the **boxplot** above, many outliers are clearly evident within the Tempo attribute. We initially anticipated that these outliers would negatively impact our prediction, however, after removing all the outliers and comparing our results before and after this, it was noticeable that these outliers only had a slight impact. This analysis is crucial as one of our main topics throughout this project was the mention of outliers and noisy data however after conducting a relevant examination, outliers were not as significant in our result as we originally thought.

```
for column, _ in tracks.items():
    if not pd.to_numeric(tracks[column], errors='coerce').notnull().all():
        continue

    Q1 = tracks[column].quantile(0.25)
    Q3 = tracks[column].quantile(0.75)
    IQR = Q3 - Q1

    tracks = tracks.loc[(tracks[column] > (Q1 - 1.5 * IQR)) &
        (tracks[column] < (Q3 + 1.5 * IQR))]

```

In our implementation, we filtered only values that were within the upper and lower fences, as demonstrated in the boxplot above. We did this for every attribute for a track, and if any attribute was outside the bounds of these fences, we deleted the track from our DataFrame - the reasoning for deleting the tracks altogether is that some albums have short or significantly differing transitionary/intro songs, these songs do not effectively represent the album itself, and these types of songs would more than likely be outliers in our data.

5. Algorithm

5.1 Algorithm Description

We originally planned to just use Naive Bayes for our predictions, but after further analysis - explained more in-depth in *Final Results, Reflection & Insights* - we made a decision to utilize four different algorithms to calculate the prediction and then ultimately narrow it down to the algorithms that performed the best for our dataset.

5.2 Implementation

We imported implementations of the following algorithms from scikit:

- Naive Bayes
- Random Forest
- Linear Discriminant Analysis
- K-Nearest Neighbor

```
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
```

We split our data into training and testing data at a ratio of 80:20 and looped through our process, swapping out which classifier was used each time. We also ran our prediction models 10 times each and stored the average score in our results map, to get a better idea of how each algorithm performed on average with different splits of training and testing data.

```
from sklearn.model_selection import train_test_split

for classifiers in [
    GaussianNB,
    RandomForestClassifier,
    LinearDiscriminantAnalysis,
    KNeighborsClassifier(n_neighbors=20),
]:
    total_score = 0
    iters = 10
    for i in range(iters):
        x_train, x_test, y_train, y_test = train_test_split(
            inputs, album_ratings, test_size=0.2)

        model = classifier()
        model.fit(x_train, y_train)
        score = model.score(x_test, y_test)
        total_score += score
    average_score = total_score / iters
    print(f"{classifier.__name__}:    {average_score}")
```

One thing to note with this is for the K-Nearest Neighbours algorithm, we found that increasing the number of neighbours improved our score by approximately 5%, while the other algorithms had little to no improvement when we attempted to tune their parameters, so we were happy to use these with their default parameters.

5.3 Results & Analysis

```
Algorithm                      Time Taken Average Score
---------                      ---------- -------------
GaussianNB                     0.0565s    80.714%
RandomForestClassifier         5.9679s    80.357%
LinearDiscriminantAnalysis     0.1169s    80.938%
KNeighborsClassifier           0.3202s    80.670%
```

The first insight we got from outputting the results was the speed of each algorithm. As displayed, Random Forest was approximately 6 times slower than the rest of the algorithms while Naive Bayes was the quickest. This makes complete sense as it is known that Random Forest's performance under large datasets such as ours, is proven slow and ineffective in real-time production. With a staggering 2239 albums after our data cleaning, we expected this algorithm to run on the slower side however we anticipated seeing the results compared to its competitors. The rest of the algorithms performed exceptionally, with all being under 0.4 seconds. From our results above, we can see all of the algorithms we used averaged **80% accuracy**.

Count of results	poor	fair	good	excellent
Real Data	13	331	1806	89
Naive Bayes	0	0	2239	0
Random Forest	12	263	1895	69
Linear Discriminant Analysis	0	3	2235	1
K-Nearest Neighbours	0	0	2239	0

When we dug deeper into our results we found something very interesting. We outputted the predictions for each algorithm on our input DataFrame and noticed that Naive Bayes was guessing "good" every time. Since that was the most common result in our dataset, it still had good accuracy!

```
model = GaussianNB()
model.fit(x_train, y_train)
prediction = model.predict(inputs)
prediction[:500]
```

```
array(['good', 'good', 'good', 'good', 'good', 'good', 'good', 'good',
       'good', 'good', 'good', 'good', 'good', 'good', 'good', 'good',
       'good', 'good', 'good', 'good', 'good', 'good', 'good', 'good',
       'good', 'good', 'good', 'good', 'good', 'good', 'good', 'good',
       'good', 'good', 'good', 'good', 'good', 'good', 'good', 'good',
       'good', 'good', 'good', 'good', 'good', 'good', 'good', 'good',
```

K-Nearest Neighbours also is completely biased towards outputting "good" but this is due to a potential misconfiguration with the number of neighbours. We discovered that increasing the number of neighbours, did increase our accuracy score (for the same reason Naive Bayes had a high accuracy), but increasing the number of neighbours is also what caused our implementation of KNN to be so biased.

6. Final Results, Reflection & Insights

Across the entire project and all the factors and tuning that came with it, we managed to gather a lot of insights and understanding of how large datasets operate. We have learned that datasets are **not perfect**. Many of the values contained in different attributes represented incorrect, missing, or even misjudged values. With this in mind, it is essential to scan through a dataset and mend the inaccurate and false figures in order to ensure a precise and accurate prediction.

Another important insight that stood out to us was noisy data. Regarding our data, it was evident from the start that Tempo contained at least a few outliers. Our first instinct was to remove all the outliers to create a more equal and balanced prediction however, after a long analysis and comparison between including and excluding outliers, it proved to us that our initial instinct was actually wrong and that outliers had little to no effect on our prediction results.

Some confusion occurred when analysing our results. As shown above, in *Results and Analysis*, many of the algorithms we used were either completely or mostly **biased** towards outputting a single result. What happened here was that we first implemented Naive Bayes and saw this strange outcome. After further analysis and testing, we decided to test out other algorithms to try and pinpoint the root cause of the "issue". Through this, we discovered that Random Forest was a much more effective algorithm for our particular dataset. Random Forest is a much **slower**

algorithm compared to Naive Bayes, so we decided that it would be a useful insight to compare the performance against the accuracy of the other named algorithms.

Implementing and experimenting with different algorithms allowed us to gain a lot of information that would be required in the progression of being able to provide improved accuracy. We initially began the algorithm phase by trying out different algorithms. With each one, we tested the accuracy in percentage as well as the time of execution.
When all the necessary data was collected, we were originally going to just use the algorithm that performed the best, however, considering a few of the algorithms demonstrated similar accuracy and timing, we settled down to use four of the algorithms within the final implementation. We feel this is a good idea as it gets to show the difference between four different algorithms. This allows us and whoever is inspecting our results to have a chance to see how various classifier algorithms performed for us.

Also, a surprising point to note is how both Naive Bayes and Random Forest resulted in the exact same accuracy score but **Naive Bayes is clearly wrong in its method of achieving that accuracy score**. Finally, it also proved to be a good learning point to study all the algorithms and ensure the implementation was returning an output within our realm of expectations.

7. Appendix

[1] Dataset - https://www.kaggle.com/datasets/josephgreen/anthony-fantano-album-review-dataset (Joseph Green)
[2] - https://towardsdatascience.com/understanding-random-forest-58381e0602d2 (Tony Yiu)
[3] - https://stackoverflow.com/questions/41844311/list-of-all-classification-algorithms
[4] - https://www.ritchieng.com/pandas-scikit-learn/