

CAP5771 Spring 2025 Project

Haylee Zuba

Milestone 2

Introduction

Project Objective

The objective of this project is to explore the relationship of taste in music and choice of video games. By considering song metadata and data from video game sales, we can get an idea of this relationship. My project is a recommendation engine that takes in a song or artist, and produces the video game with the highest similarity score. This score is calculated based on key features extracted from the datasets I used: year, emotional mappings, genre, and 1 more to be inserted later. To achieve this, I used multiple python libraries. I used pandas to manipulate the datasets, sklearn for normalization of data and creation/training of the machine learning models, matplotlib for creating visualizations of the data, and numpy for arithmetic manipulation of the data. For the UI, I used HTML, CSS, and Javascript to create the UI and add functionality. The data used in this project is the Audio Features from songs 1911-2011, Spotify Track Features, and video game sales trends, all available on Kaggle.

Project Timeline

Since the due dates of the project have changed, I had to adjust my timetable accordingly in order to complete the project by the new due dates. Also, as the project has developed, I have had new ideas for features and functions that will create a fuller fleshed out project. The syllabus and course schedule note that

4/14 - 4/23 are designated for presentations, so the goal is to finish the project in its entirety before the presentations begin. Here is the timeline for Milestone 3:

Task	Date to be Completed	Estimated Time
Model Evaluation/Improvement	4/10	~6 hours
Model Interpretation	4/11	~4 hours
Bias identification and analysis	4/13	~4 hours
UI Finalization	4/15	~6 hours
Presentation and Final Report	4/16	~6 hours

EDA Recap

After performing EDA on the 3 datasets, a few conclusions were apparent. Since the DEAM dataset did not contain labels for song names, I ended up finding other datasets that made it easier to match songs with video games. After processing the datasets and preparing them for merging, I created the dataset used for the feature engineering. Since I have domain knowledge - meaning I can see the columns of the datasets and have a general idea of how the data will look, I chose features I thought most important in calculating similarity scores - popularity, and emotional level. If I had chosen better datasets earlier, I would

have implemented a span feature that allowed items within 10 years of release to be considered. The Hidden Trends in Video Games dataset provides information on video games such as genre, release date, popularity globally and nationally, company, and platform of release. For purposes of this project, we don't need information on the publisher or the platform on which it is played. In order to prepare this dataset for feature engineering, I modified the dataset to only contain the columns I deemed necessary: Title, genre, Year of release, and ranking. These attributes are the most important because we can compare these to song metadata. From the Spotify track features, I created a subset containing the genre, artist_name, track_name, popularity, energy, tempo, and valence information. These features from the datasets that we explored in the EDA section are important for the feature engineering section of this project.

While the new Spotify dataset contained 18 columns of musical metadata, I wanted to round out the musical analysis in the code as much as possible. To supplement this, I also used the Audio Features of Songs from 1921 to 2011. This dataset is also a subset of the Million Song dataset. This dataset specifically focuses on the timbre of songs. Timbre refers to the tone color or quality of sounds. I included timbre in the emotional calculation because, as one of the eight pillars of music, it is key to determining the emotional coefficient of a song. For example, a user who prefers a song with a brighter timbre or sound quality might

be well suited for an adventure game of similar timbre. On the other hand, a user who prefers songs with a dark timbre, would probably prefer to play a horror game. These considerations create a more nuanced song and game match.

Feature Engineering

Feature Creation

In order to train the 3 machine learning models on the data, it is crucial to manipulate the datasets into an aggregated dataset. For this project specifically, I merged the Video Game Sales, Spotify Track Features, and Audio Features datasets into one dataset. Then, I will be able to properly split the data into training, testing, and validation sets. I want the machine learning models to predict a video game based on a song, so after performing EDA on the datasets used I can determine which features to create from the datasets. Below shows the final dataset with some original columns from the datasets and some new features I created.

```
"/Users/hayleezuba/PycharmProjects/Data Analysis/.venv/bin/python" /Users/hayleezuba/PycharmProjects/Data Analysis/src/data_manipulation.py
genre  artist_name  ...  Genre  emotion_score_y
1311976  Opera  Maria Callas  ...  Fighting  0.80
3406348  Indie  The Gaslight Anthem  ...  Platform  0.27
6636598  Opera  Gaetano Donizetti  ...  Shooter  0.63
12191220  Opera  Jules Massenet  ...  Adventure  0.85
13861321  Soundtrack  Alan Silvestri  ...  Adventure  0.85
12207086  Opera  Jules Massenet  ...  Role-Playing  0.77
1342879  Opera  Giuseppe Verdi  ...  Sports  0.70
7675727  Opera  Plácido Domingo  ...  Misc  0.50
4591241  Movie  Judy Kuhn  ...  Misc  0.50
11067211  Blues  The Blues Brothers  ...  Strategy  0.43
13858042  Soundtrack  Alan Silvestri  ...  Adventure  0.85

[11 rows x 11 columns]
```

The features I have created and am using from the datasets are engagement_score emotional mapping. Engagement is created from the ranking in the video games dataset and popularity scores of the songs. Emotional mapping uses valence, tempo, and energy values from the songs to match with genre values from video game data. I manually created a mapping that assigned values to each genre, and was able to merge the song and video game datasets on that.

```
genre_emotion_map = {  
    'Strategy': {'energy': .2, 'tempo': .4, 'valence': .7},  
    'Sports': {'energy': .9, 'tempo': .8, 'valence': .4},  
    'Simulation': {'energy': .6, 'tempo': .5, 'valence': .4},  
    'Shooter': {'energy': .6, 'tempo': .4, 'valence': .9},  
    'Role-Playing': {'energy': .7, 'tempo': .8, 'valence': .8},  
    'Racing': {'energy': .75, 'tempo': .6, 'valence': .4},  
    'Puzzle': {'energy': .1, 'tempo': .1, 'valence': .1},  
    'Platform': {'energy': .3, 'tempo': .2, 'valence': .3},  
    'Misc': {'energy': .5, 'tempo': .5, 'valence': .5},  
    'Fighting': {'energy': .8, 'tempo': .8, 'valence': .8},  
    'Adventure': {'energy': .85, 'tempo': .9, 'valence': .8},  
    'Action': {'energy': 1.0, 'tempo': 1.0, 'valence': .75}}
```

Figure representing the mapping between video game genre and music metadata.

Span refers to the year of release, but with a 10 year consideration buffer. Essentially, if the song or video game came out in 1995, the songs or video games considered for matching have a release year 1990 to 2000. These features are extracted from the old datasets and are used to calculate videogame-song matching.

Categorical Variable Encoding

When training machine learning models, non numerical data must be encoded into numerical values. In order to prepare the data for training, I had to

use label encoding on artist and track names. There were about 10 thousand entries in the song track dataset, so one hot encoding took too long. However, one hot encoding worked on the Game Title and genre. Once I encoded this information, I was able to extract a new feature from the data called Engagement_score, and use my other created feature emotional_score to prepare the data for training.

Feature Selection

Feature Importance Evaluation

When evaluating feature importance, I already had domain knowledge. I approached the data with features in mind that I wanted to extract and utilise in my similarity calculations. If I did not know what features would appear most frequently, or were most crucial to the project, I would have used either correlation analysis or a chi square test to determine feature importance. I also learned a lot about RFE feature importance, where the class recursively reduces features until only the desired number of most important features remain. However, since I had domain knowledge, I decided not to write any code calculating the importance of features. I was unsure on this approach until I read up on different approaches to Data Science problems and projects. Overall, the features I chose ended up being good fits for the task at hand.

Dimensionality Reduction

The datasets I chose were large, but not too large or over complicated to the point where it affected computation time or interpretation. I dropped columns I deemed unimportant and merged features or created new ones. Every code I ran on the datasets ran in a few seconds and produced clear results after the preprocessing. Since further dimensionality reduction was not needed for my chosen data, I did not need to implement specific dimensionality reduction, though I did research PCA and other dimensionality reduction techniques. However, after running the models, it may appear PCA will be required for fine tuning the models.

Data Modeling

Data Splitting

In order to split the data for training the machine learning models, I used a standard 80% training and 20% testing split. I used SKLearn's `train_test_split` function to split the data into the two sets. I opted for an 80/20 split because I wanted to avoid overfitting, which is why I did not use another standard split of 85/15. I also am in the process of manually creating a validation set of data, but this has not been finished by the project due date. I decided to create a validation set that is close in size in order to get experience manually creating a validation set, and to fine tune the machine learning models for milestone 3.

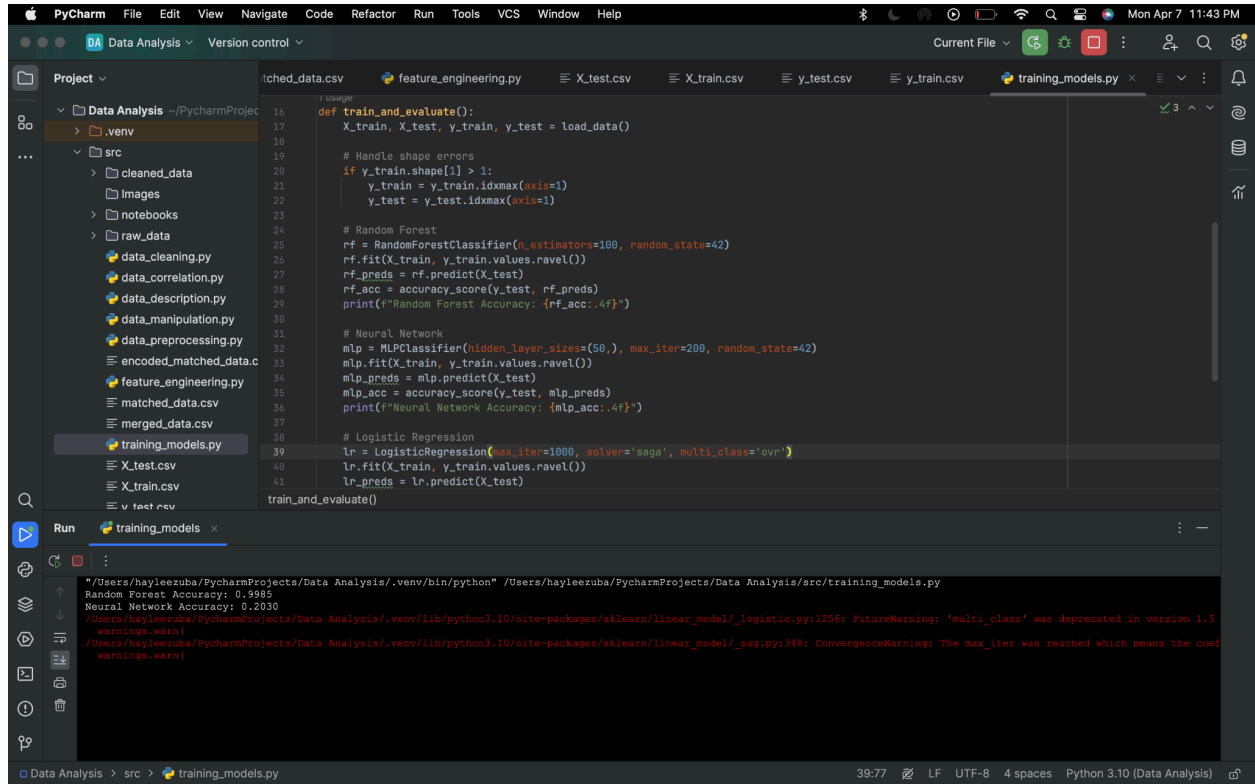
Model Training and Selection

I chose these three models to train: Random Forest, Logistic Regression, and Neural Network. I chose these methods after researching them for various reasons. Random forest is a model that combines multiple decision trees, making it a very robust model. This model is good to combat overfitting, and works well with categorical data. This model fit my video game and song datasets well because it handles complex data well and provided accurate results. Logistic regression was my next choice because I wanted to really examine if the models would overfit or not. It is fast and efficient and scales well for large datasets. Logistic regression took the longest to run, due to its linear nature. Neural networks is another example of fast, efficient, and scalable models. Neural networks are commonly implemented when nonlinear, complex data is used with little feature engineering.

Model Analysis

After training and testing each model, and using SKLearn's built in accuracy checker, these are the results outputted from each model. It appears the Random Forest model is extremely accurate, and the neural network is not accurate due to some errors in formatting. I will have to go in and further explore the data and see

where I went wrong and reevaluate if this is a good machine learning model for the task at hand.



```
16 def train_and_evaluate():
17     X_train, X_test, y_train, y_test = load_data()
18
19     # Handle shape errors
20     if y_train.shape[1] > 1:
21         y_train = y_train.argmax(axis=1)
22         y_test = y_test.argmax(axis=1)
23
24     # Random Forest
25     rf = RandomForestClassifier(n_estimators=100, random_state=42)
26     rf.fit(X_train, y_train.values.ravel())
27     rf_preds = rf.predict(X_test)
28     rf_acc = accuracy_score(y_test, rf_preds)
29     print(f"Random Forest Accuracy: {rf_acc:.4f}")
30
31     # Neural Network
32     mlp = MLPClassifier(hidden_layer_sizes=(50,), max_iter=200, random_state=42)
33     mlp.fit(X_train, y_train.values.ravel())
34     mlp_preds = mlp.predict(X_test)
35     mlp_acc = accuracy_score(y_test, mlp_preds)
36     print(f"Neural Network Accuracy: {mlp_acc:.4f}")
37
38     # Logistic Regression
39     lr = LogisticRegression(max_iter=1000, solver='saga', multi_class='ovr')
40     lr.fit(X_train, y_train.values.ravel())
41     lr_preds = lr.predict(X_test)
42     train_and_evaluate()
```

Run training_models.py

```
"/Users/hayleezuba/PycharmProjects/Data Analysis/.venv/bin/python" /Users/hayleezuba/PycharmProjects/Data Analysis/src/training_models.py
Random Forest Accuracy: 0.9985
Neural Network Accuracy: 0.2030
/Users/hayleezuba/PycharmProjects/Data Analysis/.venv/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:1256: FutureWarning: 'multi_class' was deprecated in version 1.5
warnings.warn(
/Users/hayleezuba/PycharmProjects/Data Analysis/.venv/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means the coef
warnings.warn(
```

The logistic regression model is taking significantly longer to run and evaluate, so I should probably use a different approach. Overall, it seems like the random forest was the best choice model, returning almost 100% accuracy.

References:

Discovering Hidden Trends in Global Video games, Andy Bramwell, Available: <https://www.kaggle.com/datasets/thedevastator/discovering-hidden-trends-in-global-video-games>

License: not listed

Audio Features Dataset, available:

<https://www.kaggle.com/datasets/uciml/msd-audio-features/data> Original dataset:

Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The Million Song Dataset. In Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR 2011), 2

License: Unavailable

Spotify Track Features, available:

<https://www.kaggle.com/datasets/nanditapore/spotify-track-features/data>

Licence: Apache License, Version 2.0

Other references:

<https://medium.com/data-science/your-features-are-important-it-doesnt-mean-the-y-are-good-ff468ae2e3d4>