Holden Craig
Machine Learning
4/25/24

MuSo

(Music Social)

A Supervised Classification Model for Predicting Musical Compatibility

Abstract

In my research I attempt to address the need for better predictions in musically based social platforms. In contrast to statistically based models, I propose a new model which uses a supervised classification model in order to predict a metric for users' musical compatibility. I develop techniques for the dataset creation based on users' degrees of separation (DOS) in an online social network. I also provide information regarding the model's structure, training, and accuracy. Possible sources of error in the model's predictions and future improvements are discussed.

Introduction

Today's internet, music, and online media worlds have skyrocketed in size and changed the veryway that people interact, experience, and connect with artists and each other. Prior to the onset of the digital age, people were exposed to a narrower span of music genres. An interesting effect of which was the larger number of listeners in each genre. As more music became generally accessible and genres subdivided into smaller specialized categories, the share of people interested in any one category saw a decrease. For avid artists and fans, music can be central to identity and cultural belonging. In the past, it was often more likely to find people who were into the same music as yourself because events would cater to popular taste as determined by music producers or record labels. This coincided with accessible media, so there was no notable disconnect for many listeners. Now that nearly all media is accessible this has changed. Listeners with specialized interests are having increasing difficulty finding each other, apart from infrequent and small live shows.

There have been some attempts to address this issue in the form of statistically based platforms that connect users by comparing or sharing listening history and reviews, for example the website stats.fm is one based on Spotify data. This works to a degree however it doesn't capture the full nuisances in their musical preferences or tastes. For example, both users may have listened to 5000 minutes of Led Zeppelin, but it is one user's number 1 artist and not in the other's top 50 artists. Or, two users both really enjoy one genre, but have no overlap in other ones.

I looked to explore a solution to this issue using machine learning, which provides the tools necessary to learn associations that humans might not necessarily pick up on at first.

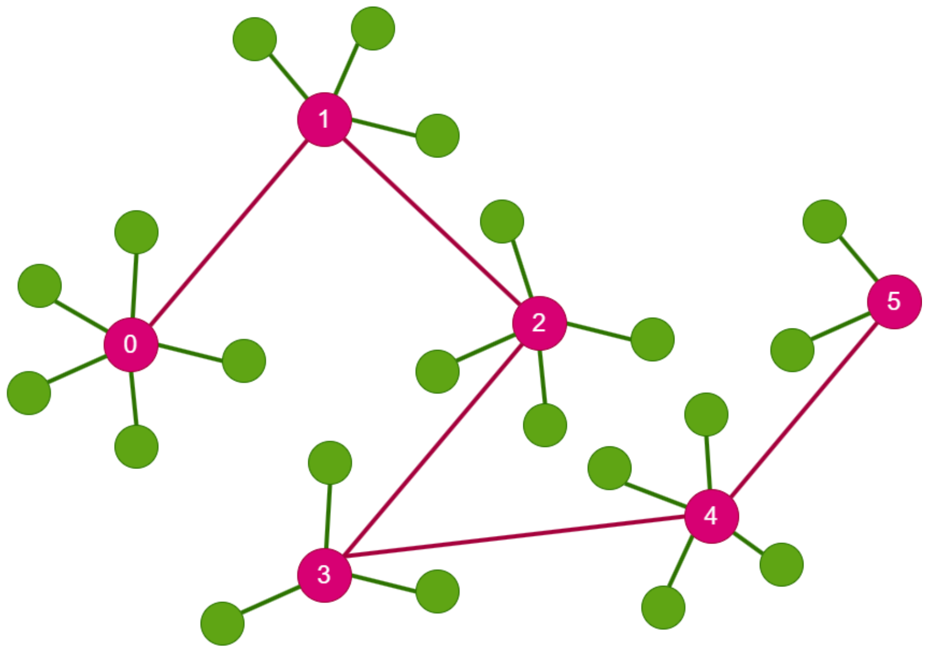
I trained a supervised classification model to infer users' mutual compatibility when it comes to the music they like most. The metric for mutual compatibility I was interested in was the degree of separation between two users in a music-based social network. If you have ever heard of the idea of six degrees of separation, you may understand the value of this information. Essentially, at any time a chain of six introductions stands between you and any other person on the planet. While the true value actually lies closer to 6.7 introductions and can vary largely between networks, the idea remains the same. The fewer degrees of separation between two people, the more likely they share similarities. And a degree of separation of one would indicate that two people are direct friends.

I used the API interface for a web platform known as Last.FM in order to construct a labeled dataset for the DOS (Degree of Separation) using two users' listening information. This labeled dataset was then used to train a supervised classification model to make predictions. The preprocessing necessary to produce inputs for the model and the corresponding DOSs was extensive and I will largely be discussing the dataset construction and feature engineering I undertook in order to produce my results.

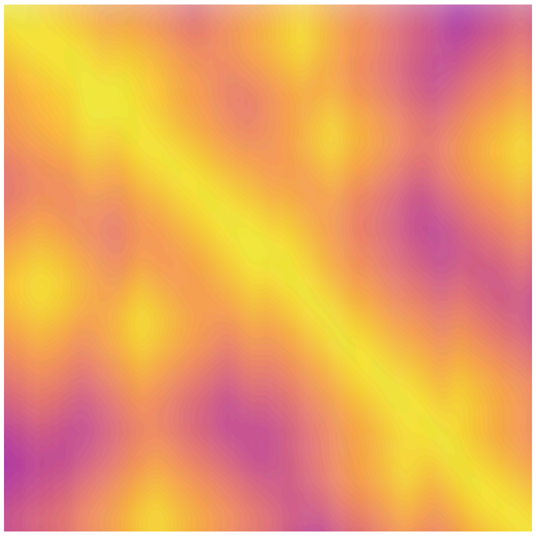
Methods

DOS (Degrees of Separation) Labels:

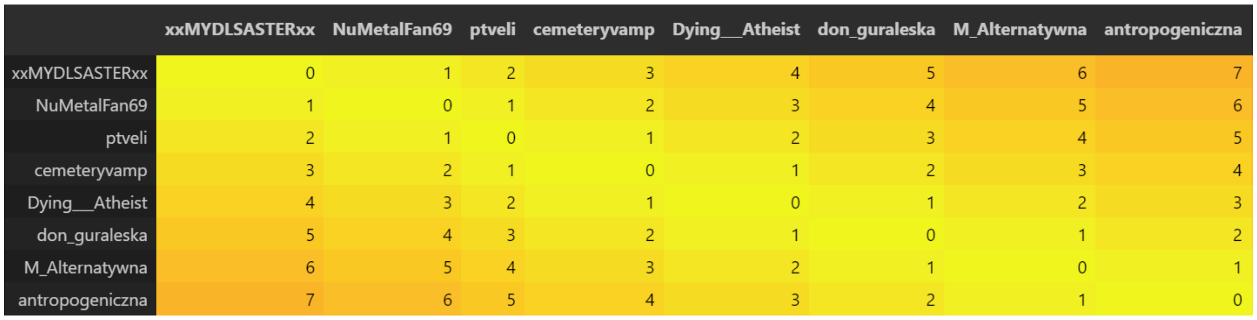
First I will discuss the process of obtaining the labels for the degree of separation between users in the Last.fm network. In order to achieve this I wrote a function that finds chains of users in the network with only one DOS between them and their neighbor. This may best be demonstrated graphically where each graph node represents a user. The connections between users indicate that they are directly friends.



The pink connections indicate a chain of users isolated from the network by the function. Then using this, a confusion matrix-like setup showing the degrees of separation between all users in the isolated chain can be determined. Here is an image indicating how the degrees of separation map out. Yellow represents no separation (Same user) and purple represents a large DOS. The small DOSs that occur further away from the central diagonal are a result of friends being interconnected to each other within the chain itself, like if the pink nodes were connected in the image above.



The associated values for the determined DOSs serve as the training labels for the classifier. Here is a zoomed in sample of the above table that displays some DOS values for users in the chain:

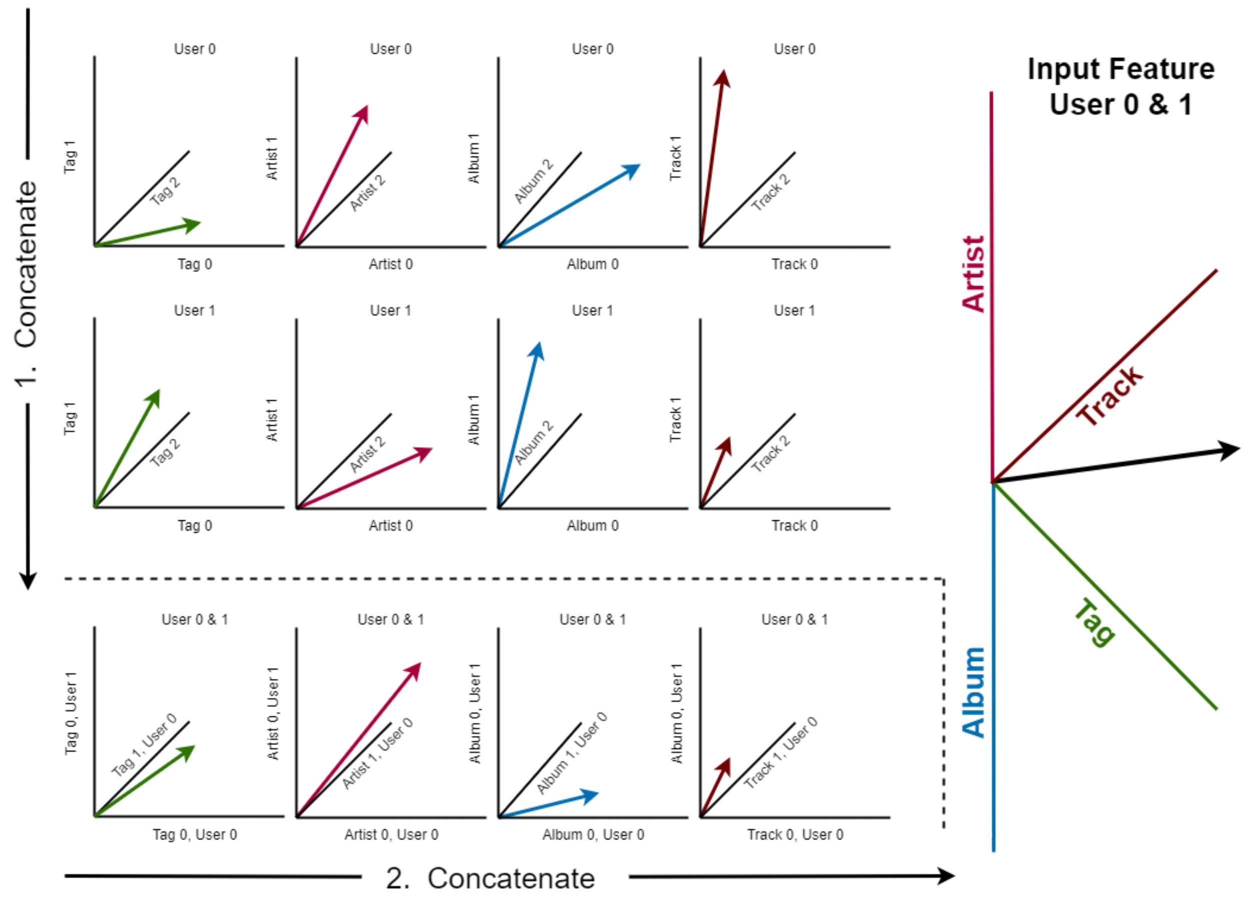


	xxMYDLSASTERxx	NuMetalFan69	ptveli	cemeteryvamp	Dying__Atheist	don_guraleska	M_Alternatywna	antropogeniczna
xxMYDLSASTERxx	0	1	2	3	4	5	6	7
NuMetalFan69	1	0	1	2	3	4	5	6
ptveli	2	1	0	1	2	3	4	5
cemeteryvamp	3	2	1	0	1	2	3	4
Dying__Atheist	4	3	2	1	0	1	2	3
don_guraleska	5	4	3	2	1	0	1	2
M_Alternatywna	6	5	4	3	2	1	0	1
antropogeniczna	7	6	5	4	3	2	1	0

User Data and Vectorization:

With the labels determined, I then had to obtain the actual data for the model to interpret. I decided to first retrieve the data and store it in a format that is somewhat human legible. I created a dictionary containing each user in the chain. For each user I obtained their listening history for the past 6 months. This included the number of plays for a variety of items like artists, albums, tracks, and tags (genres). Here is an example of the structure:

```
{"user_0": {
    "top_tags":    {"tag_0":    67890, "tag_1":    12345},
    "top_artists": {"artist_0": 5678,  "artist_1": 1234},
    "top_albums":  {"album_0":  456,   "album_1":  123},
    "top_tracks":  {"track_0":  34,    "track_1":  12}
    },
 "user_1": {...},
 ...
}
```
This data stored in this structure is friendly to interact with programmatically, but is not directly able to serve as a feature vector to a machine learning algorithm. In order to transform the data into something the model could understand, I had to create a feature space that encapsulated the underlying data for two users. This process involved first defining a feature vector for each category in the dataset. This way, the tags, artists, albums, and tracks would be each represented as vectors for both users. I then used concatenation to join the vectors together in order to produce the true final input. This diagram illustrates the process:



An important component of the vectorization process is the way by which the tag, artist, album, and track spaces are created. These spaces consist of a direction (an index) for each unique item in the dataset. The value at each index corresponds to the number of plays for that item. This may best be displayed in an example table:

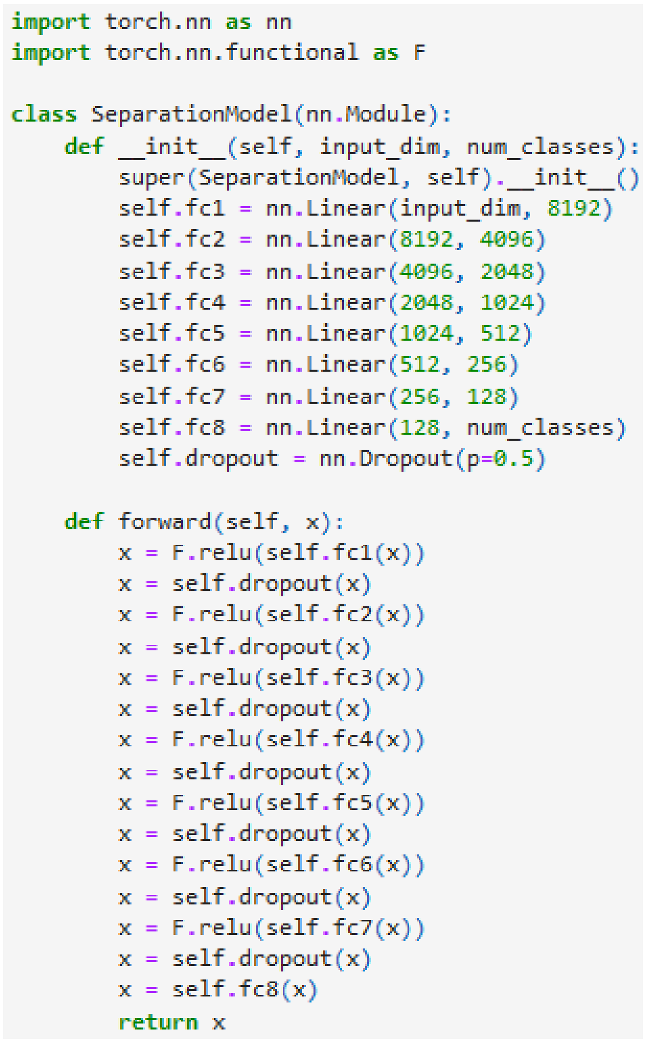
User 0				
Index	**Tag**	**Artist**	**Album**	**Track**
0	67890 (“tag_0”)	5678 (“artist_0”)	456 (“album_0”)	34 (“track_0”)
1	12345 (“tag_1”)	1234 (“artist_1”)	123 (“album_1”)	12 (“track_1”)
2	0 (“tag_2”)	0 (“artist_2”)	123 (“album_2”)	123 (“track_2”)
…	…	…	…	…

User 1				
Index	**Tag**	**Artist**	**Album**	**Track**
0	12345 (“tag_0”)	12 (“artist_0”)	1 (“album_0”)	0 (“track_0”)
1	0 (“tag_1”)	1 (“artist_1”)	0 (“album_1”)	12 (“track_1”)
2	12345 (“tag_2”)	1234 (“artist_2”)	123 (“album_2”)	12 (“track_2”)
…	…	…	…	…

This enables the vector that is produced for two users in the vectorization process to take on a fixed number of elements which is essential to defining a consistent input vector that will retain meaning across user combinations and listening preferences. There is one possible major drawback to this approach however. The model can only learn relationships that can be inferred from the training dataset. So if one input user listens to a significant amount of music that the model was not trained on, the predictions could be vastly inaccurate. The underlying assumption to my approach was that given a large enough sample of users, a variety of musical preferences would be captured and by extension be representative of the entire population. In theory, the model may still be able to make ok predictions if trained on smaller datasets. This would be true if the model learns to place more importance on the more general categories like the tags and artists as opposed to the tracks.

One last note regarding the vectorization process I took is the use of normalization techniques. In order to avoid large weights and exploding gradients, all the users’ listening data was normalized at each step. This was accomplished by subtracting the mean value for each item and then dividing by the standard deviation. The other benefit gained from normalizing the input vector was faster convergence during training.

Model Architecture and Training:

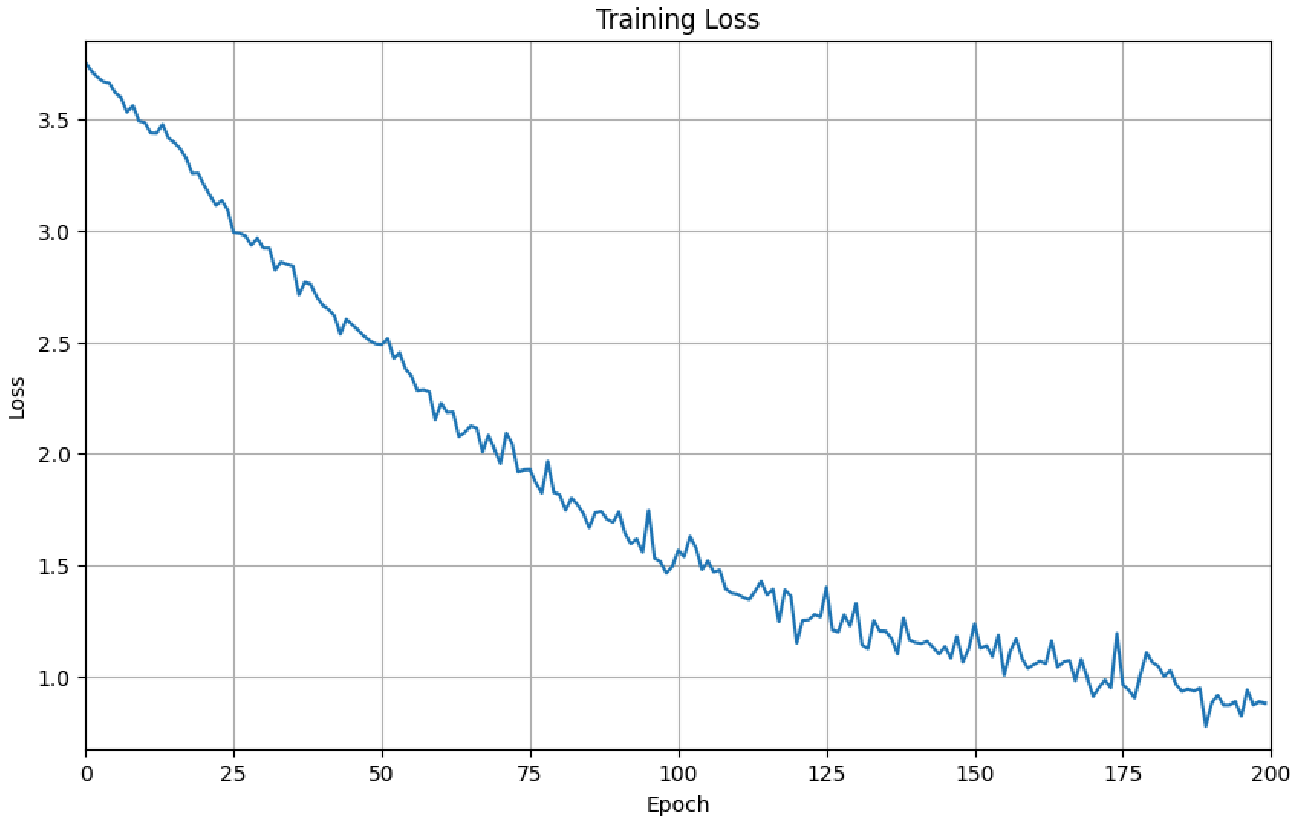


```python
import torch.nn as nn
import torch.nn.functional as F

class SeparationModel(nn.Module):
    def __init__(self, input_dim, num_classes):
        super(SeparationModel, self).__init__()
        self.fc1 = nn.Linear(input_dim, 8192)
        self.fc2 = nn.Linear(8192, 4096)
        self.fc3 = nn.Linear(4096, 2048)
        self.fc4 = nn.Linear(2048, 1024)
        self.fc5 = nn.Linear(1024, 512)
        self.fc6 = nn.Linear(512, 256)
        self.fc7 = nn.Linear(256, 128)
        self.fc8 = nn.Linear(128, num_classes)
        self.dropout = nn.Dropout(p=0.5)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = F.relu(self.fc3(x))
        x = self.dropout(x)
        x = F.relu(self.fc4(x))
        x = self.dropout(x)
        x = F.relu(self.fc5(x))
        x = self.dropout(x)
        x = F.relu(self.fc6(x))
        x = self.dropout(x)
        x = F.relu(self.fc7(x))
        x = self.dropout(x)
        x = self.fc8(x)
        return x
```

The neural network architecture consists of eight fully connected layers with rectified linear unit (RELU) activations between each layer. The RELU activation allows the model to learn non-linear relationships that may occur in the feature vector, making it more robust in learning associations between user data. Also, dropout is applied between each layer to assist in the prevention of overfitting to the training data.

The model was trained on 435 user combinations and corresponding DOSs. The input feature vector containing two users' data contains 18 432 floating point values. The model can classify the expected DOS to be between 1-41 based on the training dataset. Training was performed for 200 epochs with a learning rate of 0.0001 and a batch size of 1. The training was done on a

GeForce RTX 3060 for approximately 5 hours. Here is a graph that displays how the loss decreased during the training process:



Results and Discussion

Model Accuracy and Sources of Error:

The model was tested on a separate test user dataset which contained 350 user combinations and corresponding ground truth DOSs. The test users for which the model predicted DOSs were not previously seen by the model. This includes both of the users' listening data. This ensured that the model was performing extrapolation to new user inputs, which is valuable to characterize whether the model is capable of generalizing or not. To assess the model's accuracy I chose to deem a DOS prediction difference of less than three to be acceptable. For the 350 user combinations in the test dataset, I found that about 17.7% of the predictions were accurate by this metric. While this may not be a perfect result, it does still suggest that the model performs more accurately when compared to randomly guessing the DOS which produced an accuracy of 12.3%.

Possible sources of error in the model's predictions could be explained by various factors such as shortcomings in the underlying assumption that friends on the LastFM platform share a similar music taste. There are two social factors that could contribute to this theory. One is that friends in real life may commonly be friends on LastFM but share vastly different music tastes. The second is that there could be the presence of popular users on LastFM whose music preferences don't necessarily align with their followers.

Another source of error could arise from missing knowledge of the test users' listening preferences. Tags, artists, albums, and tracks not within the training dataset won't hold any meaning for the model. The corresponding DOS predictions could be negatively impacted due to this missing data. The error arising from this consideration could be alleviated by using a larger training dataset.

Lastly, the model is quite large due to the size of the input feature vector which could make the training process take a significant amount of time to infer important relationships between two users' listening data. The model could possibly be simplified to account only for users' genre or tag listening information. This is in a sense a more abstracted form of the other provided data, however this could eliminate potentially valuable information from the other categories.

Conclusion

This project was a valuable exploration into the capabilities of machine learning to infer complex relationships between users' listening data and musical preferences. Using a supervised classification model, I was able to predict the degree of separation (DOS) between two users in a music based social network called LastFM. A complex process of discovering chains of users in the network and generating a labeled dataset of user listening data and DOS values was implemented. Using this data, vectorization was performed to provide the model with a consistent input that was representative of two users' listening preferences spanning genres, artists, albums, and tracks. Two datasets were produced, a training dataset and a test dataset. The model was trained using the labels provided in the training dataset and achieved an accuracy of 17.7% on the test dataset. Possible improvements include simplifying the input feature vector and the model itself to improve the rate of convergence during training along with building a larger training dataset that spans a larger range of listening preferences. Future work concerning this project likely would include improving the model's accuracy by exploring possible shortcomings within my approach. Once a model with a satisfactory accuracy is achieved, I would be interested in using sensitivity analysis to explore how the listening data of a user impacts the model's DOS prediction. It would be interesting to learn about what the model identifies as most important in the input feature vector. Also, I would like to eventually create a form of visual interface that would provide friend suggestions based on the model's predictions. I believe this project has provided me with many insights into the full process of creating a machine learning model and challenged me to learn about the many considerations that must be taken into account when developing a new framework for predicting users' musical compatibility.

Resources

MuSo Repository V1 (No Model Testing)   MuSo Repository V2 (Model Testing Incl.)

MuSo Presentation Slides   MuSo Pretrained Model