Inferring Music Preferences with Machine Learning Techniques

Connor Moore
University of Washington
185 W. Stevens WA NE
Seattle, WA
moorec22@cs.uw.edu

Megan Hopp
University of Washington
185 W. Stevens WA NE
Seattle, WA
hoppm@cs.uw.edu

Naomi Musgrave
University of Washington
185 W. Stevens WA NE
Seattle, WA
naomi.g.musgrave@gmail.com

ABSTRACT

Since the rise in commercial music applications, a large amount of data and metadata pertaining to music listening habits is available. There have been many efforts to evaluate music preference through analyzing music attributes (tempo, pitch, timbre, etc), and work done to learn and predict a user's listening habits from social data or simple analysis. The ability to infer music preferences is still incomplete, and there is much work to be done before a solid model is available. We explore various machine learning techniques, in an effort to both distinguish taste and infer similarity.

Categories and Subject Descriptors

I.5.1 [**Pattern Recognition**]: Models—*Neural net, statistical*; I.5.3 [**Pattern Recognition**]: Clustering—*Algorithms*

General Terms

Algorithms, Experimentation, Theory

Keywords

Sound analysis, machine learning, artificial intelligence, neural networks, restricted boltzmann machine, logistic regression, random forest, clustering, k-means clustering, affinity propagation

1. INTRODUCTION

Conducere is an attempt to use a number of machine learning models to document and classify music listening habits by user. Though there is quite a bit of work done in modeling social listening habits, *Conducere* is a more pure and agnostic attempt at distinguishing music tastes of individuals and focuses on analyzing quantitative musical features.

In an attempt to classify music, we gathered approximately 1100 samples (specifically, songs broken into features) scattered across 11 individuals. To analyze the songs and mine acoustic attributes, we used the EchoNest[1][1] API. The API, combined with the Spotify[2][2] API, allowed us to retrieve auditory features from each song.

After that, using the Python library scikit-learn[3][3], we put together three different learning models. While the results are not conclusive, with 1100 data points they are promising, and offer a clear path forward to potential future work.

2. DATA

Our goal for collection was to gather enough data for a proof of concept. We used tracks from our own playlists along with playlists from seven other spotify users to harvest musical feature data for our experimentation.

2.1 The Users and Playlists

While the playlists that we put together for this project ourselves were made with the project in mind, we did our best to be agnostic. We did not know what models we would be using, and did our best to represent our current listening habits.

While some playlists comprehensively represented a wide range of a user's listening habits, others were more specific niches of a user's mood, with names like "quiet yearning" and "Emerald City Vibes." This distinguishment produces some interesting results, discussed later in the paper.

2.2 Features

The EchoNest and Spotify APIs allowed us to retrieve 9 features: danceability, energy, liveness, loudness, speechiness, tempo, valence, instrumentalness, and acousticness. Whether or not these features would prove useful, and in what models they would distinguish themselves, remained to be seen at the time of collection. For reference, the features mapped to labels are found in Table 1.

3. MODELS

After collection, our team worked individually to build three separate models. We did our best to pick models that were the most promising, while at the same time producing

UW CSE 481i Winter 2016, Seattle, WA

[1]EchoNest is a music analysis framework, that recently paired with the Spotify API.
[2]The Spotify API allows access to public playlists, along with your own private playlists given a user ID and credentials.
[3]scikit-learn is an extensive machine learning package in Python.

Table 1: Features

Feature	Label
danceability	0
energy	1
liveness	2
loudness	3
speechiness	4
tempo	5
valence	6
instrumentalness	7
acousticness	8

contrasting results. For reference, with 11 labels, a random guess would yield an average mean of 0.0909, or 9.09%.

3.1 Random Forest

A random forest classifier is an ensemble learning method that fits a number of decision tree classifiers on various subsamples of the dataset and uses averaging to improve the predictive accuracy and control overfitting. It has a fairly good performance out of the box and is known to combat overfitting quite well. During training, individual decision trees within the forest are assigned weights according to their individual accuracy. This allows more accurate decision trees to be weighted more highly, and also prevents very inaccurate (overfitting) decision trees from holding too much power in the prediction process. Predictions are determined by a random forest through introducing a set of input data to the trained forest, and the predicted class is a weighted vote of all the trees in the forest. Each weight corresponds to the probability estimates for each tree from training.

For example, if a forest had the following 3 trees:

[T1 (avg. accuracy / weight = 0.4),
T2 (avg. accuracy / weight = 0.9),
T3 (avg. accuracy / weight = 0.1)]

After introducing a data point for binary class prediction [A=(1) or B=(-1)], the results are as follows:

[T1 = A (0.4), T2 = B (0.9), T3 = A (0.1)]
= (1)(0.4) + (-1)(0.9) + (1)(0.1) = -0.4

Since the result was less than 0, the forest would classify this data as B. It should be noted that a majority of the trees can potentially vote negatively against the final classification conclusion, if the positive votes are weighted highly enough to outweigh the negative–as seen in the example above.

3.1.1 Random Forest Multiclass Classification

We used random forests for multi-class classification, but the underlying principle of weighted classification votes is the same. Typically, the more trees in your forest, the better. We tried different sizes of the forest, along with many different parameters for fine-tuning both the classifier and the features we used in predictions.

Table 2 shows some average accuracies for various sizes of the forest, predicting over the full set of music features, with default settings. The results were produced with the random forest classifier setting the weights for each decision tree, shown in Table 3 for the forest with 1000 trees. Most features were ranked similarly, but a few were seen to be more important by the classifier. We also worked on feature selection, in order to only use the most relevant features. This yielded slightly better results of 30%-35% mean accuracy.

Table 2: Random Forest Results

Forest Size	Accuracy
1	0.158
10	0.226
50	0.237
250	0.243
1000	0.257

Table 3: N=1000 Random Forest Weights

Feature	Weight
danceability	0.112
energy	0.101
liveness	0.088
loudness	0.121
speechiness	0.108
tempo	0.102
valence	0.113
instrumentalness	0.138
acousticness	0.117

3.1.2 Random Forest Binary Classification

The most interesting results came in an experiment of splitting the input data by user and using the random forest as a binary classifier. We ran our classifier on a pruned train/test data set, based on user pairs, for every user matching in our full data set, so the random forest was only evaluating tracks/features of at most 2 users. We evaluated each run separately. For reference, with this new approach, a random guess would yield a mean accuracy of 50%. Table 4 gives a summary of results. We can see that at it's best it does extremely well in predicting the correct user for a set of track features. The average case is also a much bigger improvement over randomly guessing. Even the lower accuracy results have valuable conclusions to be made. The closer to 50% an accuracy produced by the classifier for a pair of users is, the closer those users' music tastes are. On the flip side, as we get farther from 50%, it indicates a larger distinction between the music preferences of the two users.

With these preliminary results, we wanted to see which user's music preference was the most distinct (i.e. which user, in all of his/her matchings, yielded the highest average accuracy). To evaluate this, we compiled the accuracies for each matching, grouped by user and took the average. Table 5 shows the results of these matchings and that user 1257662670 had the most distinct music taste through our

Table 4: Random Forest on $|Y| = 2$

Label	Accuracy
min reported	0.534
max reported	0.937
average	0.711

Table 5: Random Forest Matching Results

User ID	Avg. Accuracy
svetlanag	0.748
naomimusgrave	0.690
hunter5474	0.655
jungleprince	0.724
corne	0.696
connor	0.676
mlhopp	0.661
scott	0.673
1246241522	0.735
sana	0.693
1257662670	0.867

data.

Random forests are powerful for classification, particularly with their resistance to overfitting. While it is still possible to overfit a random forest, increasing the size of the forest along with the inherent weighting property make it difficult. In our experimentation, the RFC consistently performed better than the baseline of random guessing. Going forward, it would be interesting to expand the user-matching technique, forming 3-way comparisons and seeing what results are produced from this. It would also be an interesting experiment to try to pinpoint what makes a user's music taste 'distinct' and launch a more comprehensive investigation into which feature combinations correspond to this attribute.

3.2 Clustering

Clustering is an unsupervised classification model that attempts to separate n-dimensional data into a number of clusters. We explored two different kinds of clustering, k-means and affinity propagation. In both models, the metric is based on accuracy of the clusters. First, each cluster is labeled the majority element from that cluster. For example, if cluster 1 is

A: 35, B: 20, C: 45

then cluster 1 is labeled C. Second, we iterate over all labeled clusters, marking any they got correct and any they got wrong. We keep the total right and total wrong for each label. In this example, after seeing cluster 1, we would have

A: [0 right, 35 wrong],

B: [0 right, 20 wrong],

C: [45 right, 0 wrong]

We then use these final counts to determine accuracy for each label. In each model, we try every possible subset of independent features, and pick the approximate best, based on mean and standard deviation of accuracies.

3.2.1 K-means

K-means clustering will look for exactly k clusters, and will shift the mean of the k clusters to place the data as well as it can. We set k = C * len(labels), and tried a number of values for C. The approximate best results are shown in Table 6. Any value for C higher than 5 is likely overfitting.

Table 6: Best K-means Clustering Results

C	Features	μ	σ
1	0, 2, 3, 5, 6, 8	0.151	0.142
2	0, 3, 5, 6, 8	0.175	0.115
3	0, 5, 6, 8	0.200	0.096
4	1, 2, 6	0.221	0.133

Table 7: Best Affinity Propagation Clustering Results

damping	Features	Clusters	μ	σ
0.5	0	1002	0.911	0.050
0.55	0	984	0.898	0.048
0.6	6	857	0.371	0.112
0.65	6	701	0.332	0.125

3.2.2 Affinity Propagation

Affinity Propagation looks at all of the data, and in an iterative process looks for the best mean centers of the data, attempting to cluster all of the data in some center or another. It doesn't limit the number of clusters, and our data can be very spread at parts. As a result, it was difficult to limit overfitting. We split the data to try to reduce overfitting, fitting to a training data set, a subset of the entire data set. We attempted to use a damping factor, which will dampen your normalized data by adding a certain amount to each of it, bringing it farther from the origin of the means but reducing the difference between the distances of individual data points from the means. Table 7 shows four runs from different damping constants. Note that a large number of clusters compared to the number of samples can be strong indication of overfitting.

3.2.3 Combining the two models

Perhaps the most interesting result came from combining these two models. If k-means is too coarse, and affinity propagation overfits, it could be that a good idea to to combine the two models in some way. The following is what resulted from letting affinity propagation overfit with a damping of 0.52, and combining the two models by making the accuracy for each label 0.7 * kMeansPrediction + 0.3 * affinityPropagationPrediction:

['acousticness']

mean: 0.400

standard deviation: 0.140

Clustering is a plain and simple classification aglorithm. For our project, this means that uncertainty in the model can be as interesting as certainty. If you look closer at the data, some labels were clustered very diffinitively, consistenly getting an accuracy around 0.6 or 0.7. Others not so much. Some playlists never rose above 0.1. But uncertainty could indicate a similarity in music taste between two labels that got clustered together consistently. It could indicate that some music tastes are more explorative and spread out, having many data points in many different clusters. It seems that going forward, it would be very interesting to see a measurement of the spread of music tastes, analyzing numbers between labels as well as the individual data points.

3.3 Neural Network and Logistic Regression

The goal is to determine if it is possible to identify a user's listening habits. If it is possible to do so, a neural network will be able to extract a feature set that describes the listening habits, and be able to correctly identify songs that align with individuals' tastes. The model is a combination of a neural network and a classifier. The model is provided song data, presented as a vector of features, and an associated user profile.

3.3.1 Artificial Neural Network

In the first step, an unsupervised neural network, a Bernoulli Restricted Bolztmann Machine, builds a model of the feature space. During training, the RBM aims to maximize the product of probabilities assigned to the features it learns. The RBM constructs a bipartite graph of neurons to describe features. The parts are divided into hidden and visible nodes. Since this is a Restricted (as opposed to unrestricted) Boltzmann Machine, there are no connections between hidden nodes. Therefore, node activations are mutually independent of each other. This is desirable, since the given feature set is already dependent (for example, a song with high danceability is more likely to be highly energetic).

3.3.2 Logistic Regression

Given the features learned by the neural network, a logistic regression function performs classification. It assigns labels to the song data. Logistic regression is useful for computing the probability of class membership. It maps a data point in a n-dimensional feature space to a value, which in turn maps to a label. Due to this behavior, logistic regression performs well on transforming a complex input set to a subset of labels. In particular, logistic regression favors and disfavors outcomes using log odds, rather than probabilities. Probabilities treat positive and negative outcomes fairly, but we prefer to discount negative outcomes and reward positive.

3.3.3 Parameter And Feature Selection

In order to find the best combination of parameters for the RBM and logistic regression models, we performed a search over all possible combinations. The search operated over the following parameter sets, using all features in the data:
Artificial Neural Network:

- Learning rate: Determines how quickly the node weights are updated. Too fast, and the model may over-fit (a major risk since some data sets are very small). Too slow, and the neural network is too unintelligent. We explored values in the space 10e-3 to 1.
- Iterations: The number of iterations is another risk for overfitting, in particular with too many passes over the network. We explored values in the space of 1 to 51 iterations.
- Components: The number of binary hidden components in the model is correlated to the complexity of the feature space the network can be trained to describe. We explored 10 to 300 components.

Logistic Regression:

- Regularization: For the model, C determined the inverse of the regularization strength, with smaller values giving stronger regularization. Regularization is another parameter that helps prevent overfitting. We explored values in the space 1 to 100.

Table 8: Comparing Parameter and Feature Accuracies

	Trial 1	Trial 2	Trial 3
RBM Learning Rate	0.004641	0.001	0.001
RBM Iterations	21	1	31
RBM Components	300	300	300
Logistic Regularization	100	100	100
Features	0, 1, 2, 3, 4, 8	2, 6, 7, 8	0, 4, 6, 8
Avg. Accuracy	19.857%	18.5%	22%

Table 9: Neural Network and Logistic Regression Results

jungleprince	0.527897567
mlhopp	0.124806267
sana	0.2053707
hunter5474	0.2461662
connor	0.1625398
naomimusgrave	0.114718933
1246241522	0.190338333

The model was trained over a power set of all features, to determine the combination giving the best results. This training occurred using the parameters found above.

For reference, the data set used had 7 labels, giving a 14.28% chance of random guessing succeeding.

Although different explorations of the parameters and feature set yielded different results, the resulting accuracies would be very similar, consistently in the $16 - 22\%$ range.

3.3.4 Results

Initially, the model was trained on the full collection of data. There was a huge variance in number of songs for each user, and the neural network performed poorly when extracting a feature set. The model never correctly guessed the labels of users with very few songs. Some users had very large playlists with songs tightly falling within a particular genre or mood; the model would almost always correctly identify songs associated with these individuals.

With input data restricted to 100 songs per user, the model performed slightly better than random guessing. This improvement was seen across the board, with the model now able to identify more songs from each individual. Some individuals were consistently identified well, and others were rarely identified.

This performance aligns with existing research that uses neural networks to identify song genre and mood. Furthermore, it implies that the model can be used to identify the features describing a user's listening tastes, solely from their prior habits.

4. CONCLUSIONS

Our chosen models gave us very interesting results. Often they distinguished music tastes very well, depending on the model. These results of course were our aim, and it was often disheartening to see inconclusive models. However, after further analysis, we believe that the uncertainty could be as interesting as certainty. When a model fails to distinguish between two or more music tastes, it may not be a fault of the model, but in fact a very important aspect of music; often people have very similar tastes. Perhaps the largest conclusion to draw from this is that for *Conducere*, we need a model that acknowledges similarity between music tastes as well as differences.

It also seems that data could be better split by "taste" rather than by individuals, to give a more accurate model for music listening tendancies. This data would seem to corroborate the obvious, that people have a number of changing music tastes, and that taste is often shared between many groups of people.

5. FUTURE WORKS

For a model that acknowledges similarity, there are plenty of options going forward, some more complicated then others. The first step would be to explore combining like music tastes at the root of a model and waiting to distinguish between them once the classification of the combined tastes is determined. The similarity analysis done in section 3.1, with Random Forest, is an excellent starting point. It mirrors one of the larger issues in the clustering algorithms found in 3.2; some users' tastes are indistinguishable from others. Initially a weakness of a simple classification model, this knowledge could be armed to make a model strong. For example, imagine a clustering algorithm where users 1 and 2 have incredibly similar music tastes, and user 3 is quite a bit different. A clustering algorithm would perhaps get a result like this:

cluster A: [U1: 45, U2, 35], cluster B: [U3: 40]

Our traditional clustering measurement would choose cluster A to be U1 and cluster B to be U3, and give the following accuracies:

U1: 100%, U2: 0%, U3: 100%

μ: 66.7%

This isn't good for user U2, especially since all songs for that user made it to one place. Say instead we do a similarity analysis to find that U1 and U2 are similar enough to be combined. We combine them to a new sample set, called V1. We then have:

cluster A: [V1: 80], cluster B: [U3: 40]

V1: 100%, U3: 100%

μ: 100%

Now say a song gets classified in V1. We can then safely classify the song as both U1 and U2, so long as our similarity index is sufficient. The above example is of course a simplified version of the idea, but even something this simple shows the need for similarity analysis.

It could also be more interesting to look at a model where, say, person A has a music tastes 0, 1, 3 and person B has music tastes 1, 2, 4. This could then be used to suggest to Person A that they listen to selections from music taste 1, from Person B or otherwise, but not necessarily suggest 2 or 4 from Person B's selection. Much of the uncertainty in the analyzed models above can be used to study similarity, and may prove to be very conclusive.

6. ACKNOWLEDGMENTS

The *Conducere* team would like to thank Bruce Hemingway and the CSE 481i staff for their mentorship and assistance throughout the development of this project.

7. REFERENCES

[1] *The Echo Nest*, Jan. 2016, http://the.echonest.com/.
[2] *Spotify Web API*, Jan. 2016, https://developer.spotify.com/web-api/.
[3] *scikit-learn*, Mar. 2016, http://scikitlearn.org/.