College of Computing and Informatics
University of North Carolina at Charlotte

A project report on

Music Recommendation System

Project Guidance By
Dr. Srinivas Akella

Team details

Anirudh Narayanan	**Mohit Varma**	**Nisar Kalaria**	**Rahul Ratra**
anaraya7@uncc.edu	mvarma@uncc.edu	nkalari1@uncc.edu	rratra@uncc.edu

Introduction:

The objective of this project is to be able to recommend relevant music to a user based on the attributes of the music listened to by him in the past. We treat this problem as a regression problem, so that we can predict the song to user based on his past listening. For this task, we make use of Linear Regression and least squares. We have also implemented another way of doing recommendations using logistic Regression, where we predict genres for songs, for users who prefer to search by genre. Often we may have to predict genre, using the music type, since the music is not absolute in genre. That being said, this could affect our accuracy in that we may absolutely predict some genres, when in fact it could be matching other genres. These tools can be leveraged to get a good prediction of whether a new song might be of his choosing in the future. If it is predicted to be so, it is recommended to the user. We are using below dataset but as of now to train our model(https://github.com/mdeff/fma)

In today's day and age empirical analysis is critical. Not every user has similar interests. Each user can have his own set of interests which may or may not overlap. These non-overlapping interests in music are more so a reason for there to be relevant recommendations which are ideal to a particular user based on his past searches, added songs and frequent listens. This problem is definitely one where based on his most listened genres we recommend new songs in these common genres which match his interests in terms of hours of music listened and songs added to personal library. This empirical analysis is the backbone of music recommendation of today's era. The goal of this project is to be able to custom fit any user's interest and give close to accurate recommendations.

Motivation for project:

Today, music recommendation can be an interesting solution with the huge amount of songs in our databases. For example, if a website like amazon prime or Spotify wants to increase customer loyalty, it will be relevant to have a good recommender system. We can see that the recommender system is widely used, not just for music. Recommendation systems used in web stores are a common example. So this project provides impetus to this problem.

A recommender system is traditionally composed of:

Users: Users are called people in the system who have preferences for items and people who can also be a source of data. Each user may have a set of user attributes, then demographic is a user attribute if we use demographic user (age, gender, etc.). A template may be assumed for each client. For example, their song genre or the type of books they like to read (users).

Items: Things to be suggested by the program are known as Items. We may have a collection of item attributes or properties for each object. For example, an actor in a film, author of an appliance is a news article, or color. Preferences: These reflect the likes and dislikes of users. In our case, the item of recommendation is the track. We have built a track recommender.

Preferences: These represent the likes and dislikes of users in the space of preferences, for example, a user can rate the song on a scale of 1 to 10.

Songs: The music list consists of variety of tracks and has recently been filled with famous tracks. In order to increase variety to music styles, an extra number of songs were added manually. Each track, including the title, artist, album and genre, is annotated in terms of metadata information. There are 100,000 songs of different artists, albums and different genres in this chart. These 100,000 songs had the most relevant data which was usable for our project

Music Feature Component:

The music feature component extracts songs' hidden features based on song metadata. Our template has two types of fields. Linear fields are fields with numerical values that are continuous.

Trackname, genre, Echonest Features (240 features about sound acoustics), User ratings.

Training Data (User rated number of songs in the live environment)

In test and proof of concept, we rate a large number of different songs, and train based on this and give recommendations. For example two files (anirudh.csv, nisar.csv) have been included with song names and ratings, showing how both these users interact with the music and what they have rated it(random information), with this, some song suggestions have been generated (Please run Ipython Notebook-Jupyter Notebook for results of the same). The recommendation Engine Provides this information. The plots provided indicate how linear regression fit the above data.

In our proof of concept, we also performed testing on a subset of the rated track data, which we recommended, and used the Precision/Recall metrics to judge how good our recommender was. In a recommender system, precision is more important than recall, because we want to give the user data which is relevant to him, than avoid irrelevant data.

The fields with enumeration values are one-hot fields. The field values are encoded separately as one-hot vectors. In case a song has several values, we put the one-hot vectors together, for example, a song may have multiple artists. The meta vector of the corresponding song is the concatenation product of all fields.

Data preprocessing:

We choose to work with numerical data for our model, while doing this we realized that only some files make sense to take into consideration from the FMA dataset.

These files are

- rawtrack.csv(which contains track info)
- features.csv(it contains all the feature info)
- echonest.csv(whole bunch of features)
- rawgenre.csv()

We combined them into two different specific csvs, one containing trackspecificinfo (display training data) and the other containing the track features(used for training). We are then using this modified data to train the 3 model used for Music Recommendation explanation and results of which is provided below.

Track Specific Data contains, track_id (for us to keep unique), genre_id, genre_name, track_name for recommending not for training.

Track feature data consists of data of millions of artists and songs, relevant to the music itself, which are continuous variables). Track Features contains , track_id and 240 different music related echonest features (The Echo Nest and Spotify APIs provides broad and dee

Model: Linear regression with least squares:

Approach Implemented:

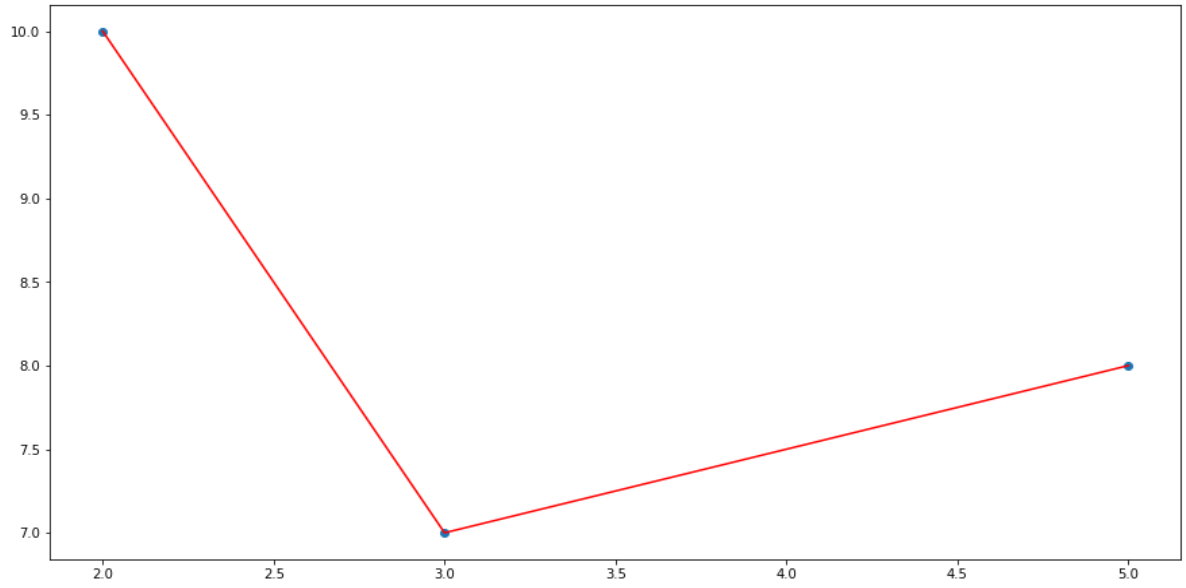
Simple linear regression is useful for finding relationship between two continuous variables. One is predictor or independent variable and the other is response or dependent variable. It looks for statistical relationship but not deterministic relationship. Relationship between two variables is said to be deterministic if one variable can be accurately expressed by the other.
Typically, linear regression is represented as:

$$Y(pred)= b0 + b1*x$$

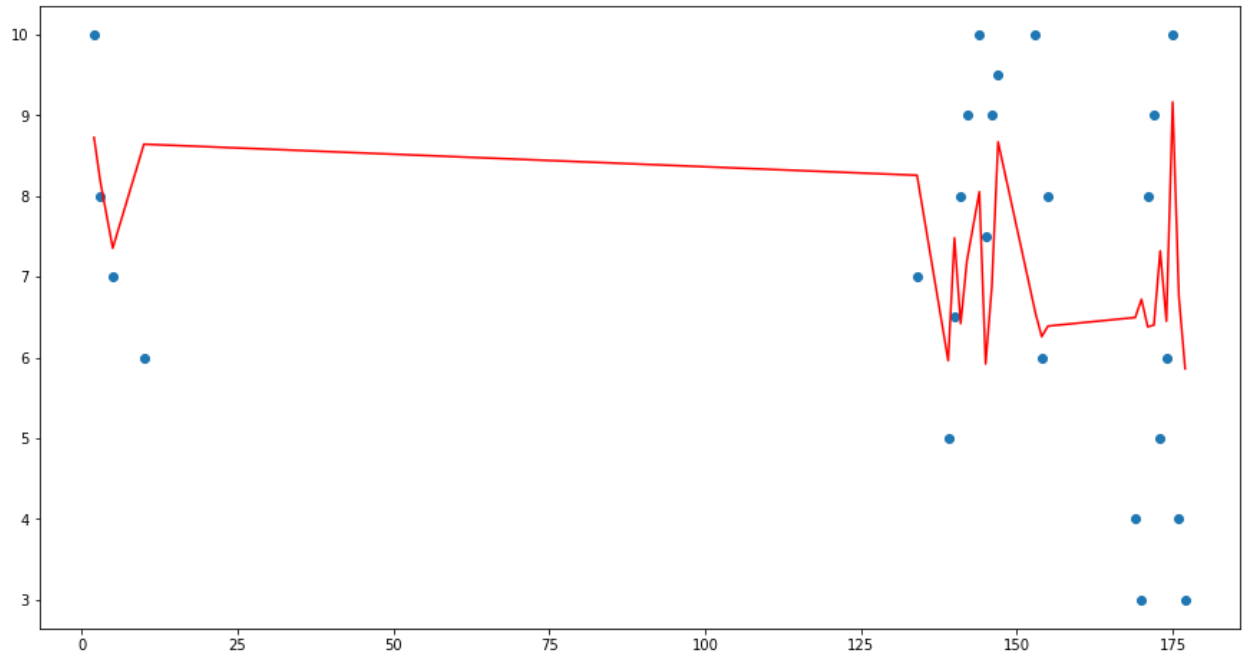
We have also taken into account of regularization, in order to achieve better accuracy. That being said, we have not really paid attention to using it consistently, since as long as our current accuracy holds good, we will not venture into new features. To minimize the error, the values b0 and b1 must be selected. If the sum of the squared error to evaluate the model is taken as a metric, then the goal is to obtain a line that best reduces the error.

Least Squares

Outputs for different scenarios:
X-Axis - Song ID
Y-Axis - Song Rating (Blue - Actual , Red- Predicted)



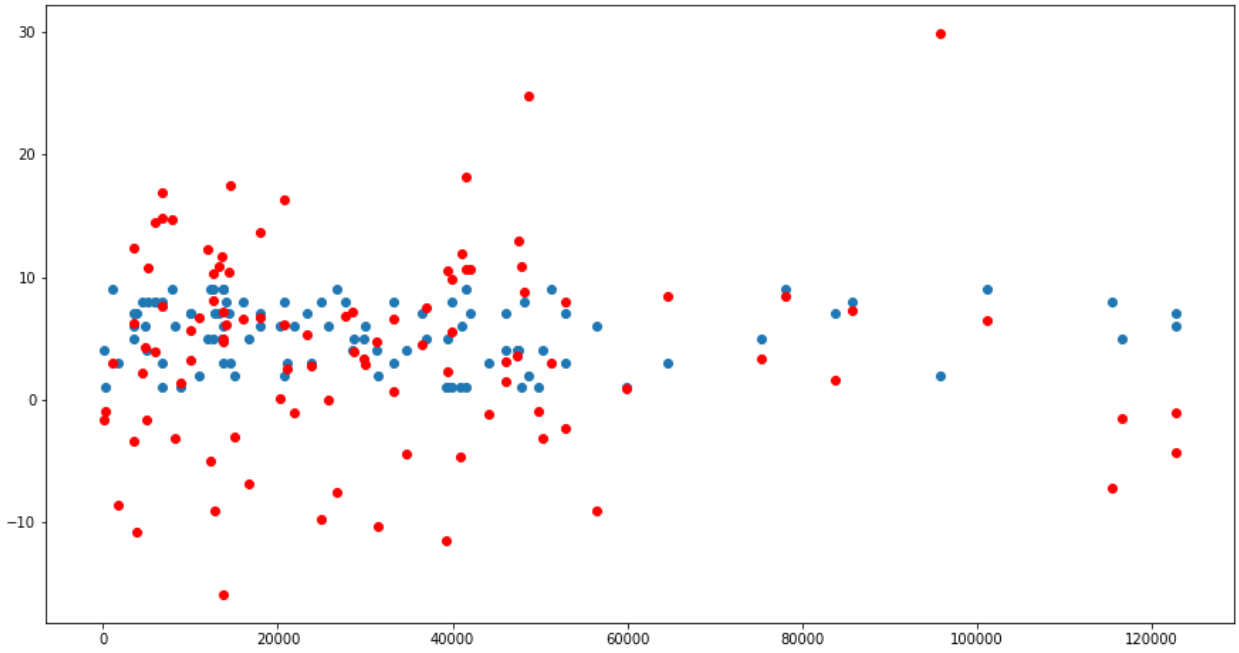
Above plot shows the regression plot for music recommendation system where blue dot are the training points and red line is the regression line. There wasn't enough data, so we can't exactly judge our performance with respect to this above user . These songs were ranked manually by one of the project members.



X-Axis - Song ID

Y-Axis - Song Rating (Blue - Actual, Red- Predicted

Above plot is better result than the previous plot not overfitting and covering most of the points. This was also manually ranked, but due to the large amount of data present, we could arrive at a conclusion which fits the pattern of the data, and not overfit it.



Least squares 100 sample
Blue: Actual
Red: Predicted
X-axis = track_id
Y-Axis = Rating

There are however some outliers which happen in the case of music recommendation, since not all songs are clearly differentiable.

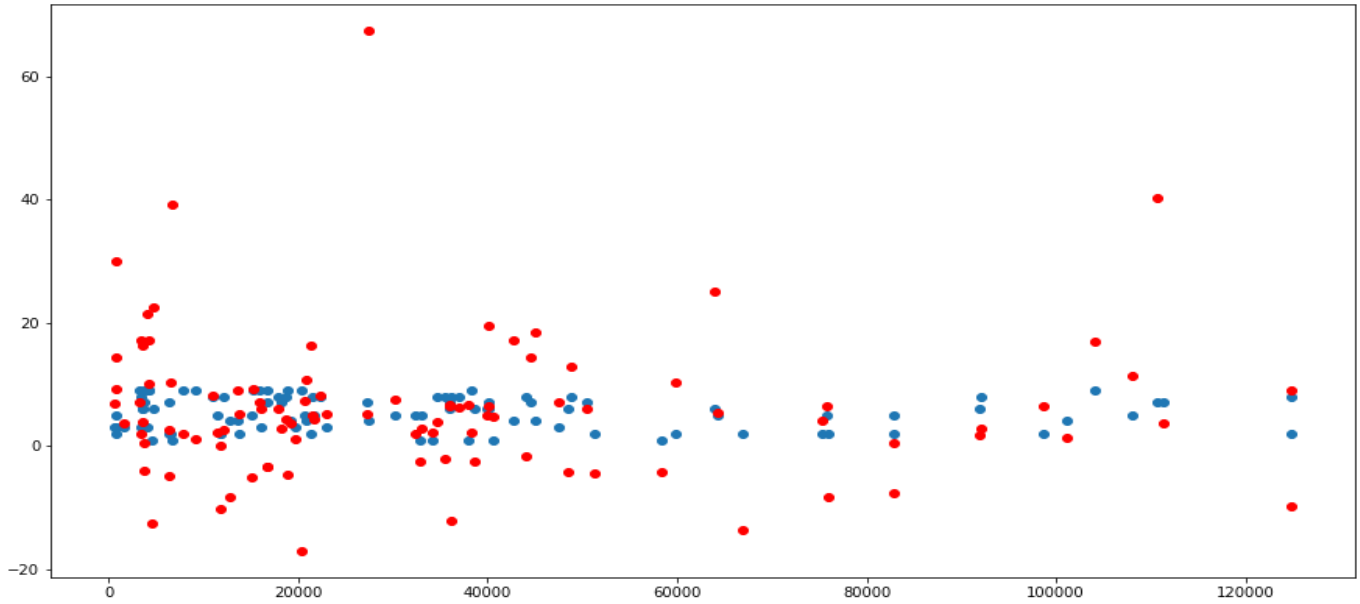
Least Squares Recommendation

```
HERE ARE YOUR RECOMMENDATIONS
[['LGNO' 'Rock' 634.0 10.0]
 ['The Sugar Society' 'Rock' 564.0 10.0]
 ['Has Been Blues' 'Blues' 462.0 10.0]
 ["I'll Be Blue" 'Blues' 463.0 10.0]
 ['Thinking Of You' 'Blues' 465.0 10.0]
 ['Big League' 'Electronic' 532.0 10.0]
 ['Mango Tree' 'Electronic' 533.0 10.0]
 ['I Can See You' 'Folk' 534.0 10.0]
 ['Angels Fear To Tread' 'Folk' 535.0 10.0]
 ['Riding On Your Fears' 'Folk' 536.0 10.0]
 ['Your Magic Motion' 'Folk' 538.0 10.0]
 ['Whoever You Are' 'Folk' 539.0 10.0]
 ['The Black Pirate' 'Folk' 540.0 10.0]
 ['So We Go Again' 'Folk' 541.0 10.0]
 ['Interpretations' 'Folk' 545.0 10.0]
 ['If You Have No One' 'Folk' 546.0 10.0]
 ['I Keep On Wondering (Interrupted)' 'Folk' 547.0 10.0]
 ['Venice, CA' 'Folk' 549.0 10.0]
 ['Ex Lucky' 'Rock' 562.0 10.0]
 ['Bessemer' 'Blues' 461.0 10.0]
 ['Orp' 'Electronic' 411.0 10.0]
 ['Lost on the Eastern Side of the Island' 'Electronic' 410.0 10.0]
 ['I wish I wish' 'Electronic' 387.0 10.0]
 ['My Room' 'Post-Punk' 359.0 10.0]
 ['Red coated young girlfriend' 'Post-Punk' 361.0 10.0]]
```

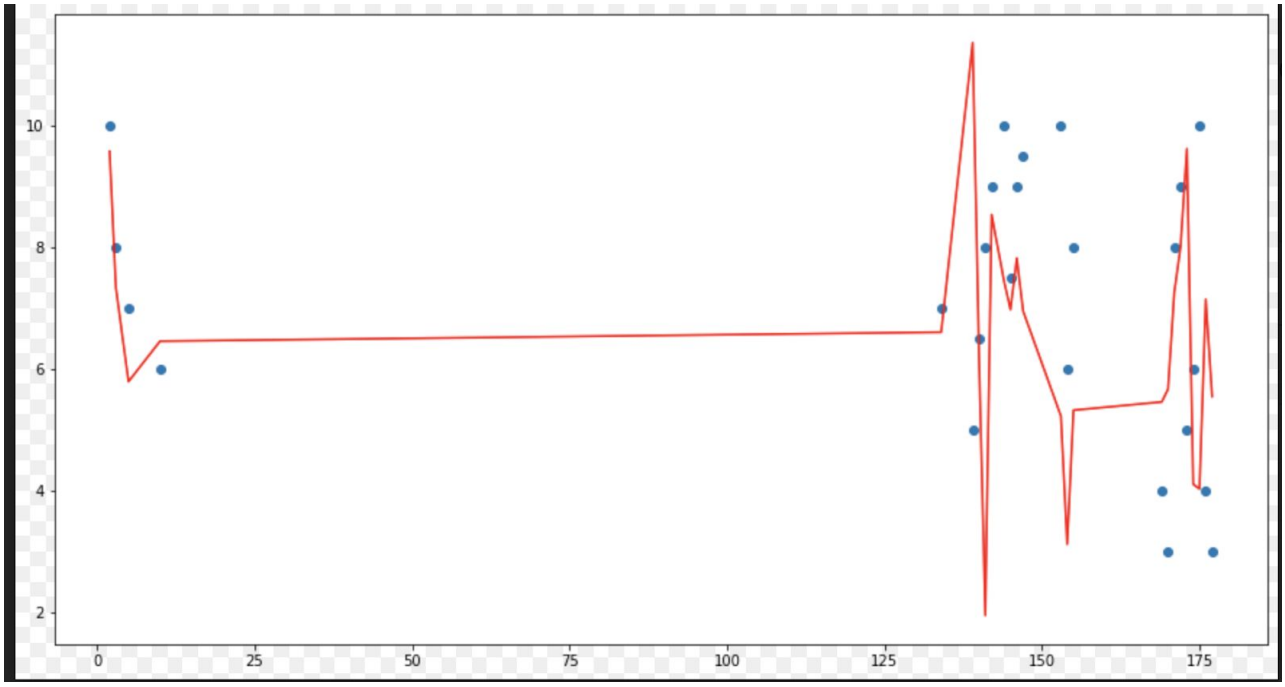
Least Squares Accuracy

```
---------
ACCURACY 76.07692307692308
PRECISION 0.5108695652173914
RECALL 0.5529411764705883

---------

---------
```

Model: Linear regression with **Stochastic Gradient Descent**



FMA Generated user data
X axis: Song ID
Y axis :Rating
Blue:Actual
Red:Predicted



FMA Generated user data

X axis: Song ID

Y axis :Rating

Blue:Actual Red:Predicted

Linear Stochastic Gradient Descent Recommendation

```
[['Electronic Interlude VI' 'Electronic' 400.0 9.332990147647529]
 ['My Room' 'Post-Punk' 359.0 9.160962574499782]
 ['Soundclash At Wonder Inn (ft. Qulfus)' 'Electronic' 407.0
  8.369152076883632]
 ['There goes my greenhouse' 'Post-Punk' 367.0 7.854766805182913]
 ['I wish I wish' 'Electronic' 387.0 7.012139378012305]
 ['I Was So Bored I Wanted To Hang Myself On The Dancefloor' 'Rock' 605.0
  6.992893163421387]
 ['Mango Tree' 'Electronic' 533.0 6.904661368808247]
 ['The Cup Bearer (Jony Mede)' 'Folk' 623.0 6.901089061116822]
 ['Little White Boxes' 'Post-Punk' 356.0 6.689051457021093]
 ['Real Prisons' 'Folk' 604.0 6.635276235253148]
 ['Dumb Me Down' 'Rock' 563.0 6.574780526494773]
 ['Oh Kap Soon' 'Electronic' 402.0 6.5090955030346604]
 ['Lover of Nightmares' 'Folk' 603.0 6.433569320426523]
 ['Borful Tang Loves Nomeansno' 'Electronic' 398.0 6.378274439120021]
 ['Wonder Inn' 'Electronic' 404.0 6.2943100731819905]
 ['John And Mary - The Restaurant' 'Electronic' 414.0 6.1128692064366765]
 ['Big League' 'Electronic' 532.0 6.0423155485333195]
 ['Burning Buddhas' 'Folk' 617.0 5.945320410423665]
 ['Shadow Law' 'Rock' 569.0 5.941354133624741]
 ['Dancing On Water' 'Folk' 629.0 5.899388862843508]
 ['John And Mary - The Meeting' 'Electronic' 412.0 5.797230520831279]
 ["I'll Be Blue" 'Blues' 463.0 5.745737320210379]
 ['Snowbank Treatment' 'Folk' 612.0 5.732464990419997]
 ['Orp' 'Electronic' 411.0 5.683537500104101]
 ["Liberty's Bell" 'Rock' 567.0 5.592350835618254]]
```

Linear Stochastic Gradient Descent Accuracy

```
---------

---------
ACCURACY 86.65563168949383
PRECISION 0.7205882352941176
RECALL 0.44144144144144143

---------

---------
```

Here a high precision indicates that we are recommending the right songs to the user, with comparison to all the songs which he expects or could be recommended. A bad recall indicates, that we are recommending a lot of songs, that he doesn't expect either. The metrics we use are Accuracy, Precision, Recall. We thought it better than RMSE, since it is a recommender, and it is easier to assess with these metrics.

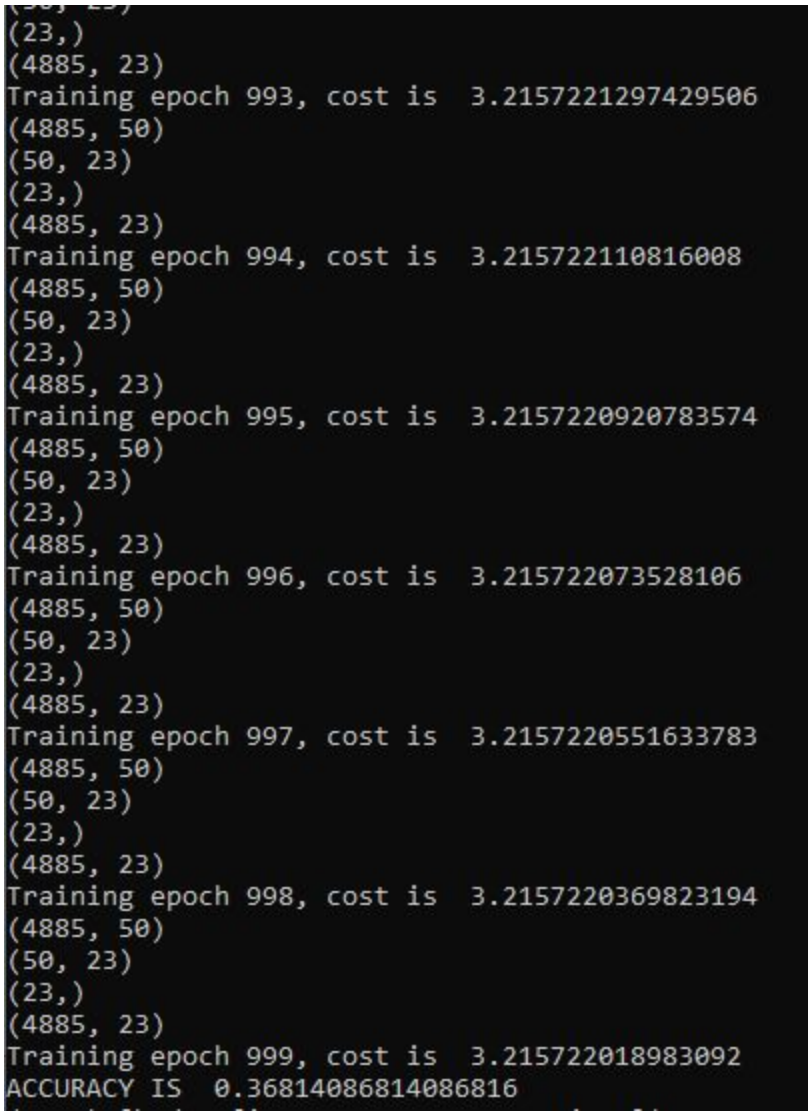
Model: Linear regression with **Stochastic Gradient Descent**

Working:

We initially provide user with set of random songs and will ask user to rate those songs based on a scale of 1 to 10. Where 10 being the highest and 1 being the lowest. Once user provide his ratings, we are going to provide these inputs which consists of track ids of songs and ratings to the recommendation engine. Recommendation engine will run linear regression on these inputs, based on which coefficients will be generated. In parallel we will run the linear regression on random 100 songs. On matching the coefficients proximity, we are going to predict the users with top 5 songs which are closer to his inputs provided.

Model: Logistic regression with **Gradient Descent:**

In this particular training, we are predicting **genres**. Our recommendation mechanism here is different. Earlier, our genres were mere features, in one-hot format, but here they are our output variable. We can clearly see how we predicted genres. The advantage of doing this, is that we give users the functionality to search by genre, of songs whose genre is not specific. Our use of echo nest features, and sound related features to predict genres is what makes our project unique.



We have run it for 100 epochs, and have calculated our accuracy. We used apache SPARK here, to parallelize our gradient descent, and we were able to predict the output for the users.
The above output is for logistic regression. The scores you see (cost) is of the logarithmic loss functions. As you can see, the loss function is decreasing in each epoch, indicating that we are having a step by step reduction in loss, which is improving our model. This data is however not LINEAR, and it isn't very easy to get a high accuracy on the data. As we can note, the machine learning algorithm does have a way to reduce loss. This is proof that our model is working.

The final line is accuracy, indicating how accurate was our prediction of the genres.

LOG REG CODE:

```
class LogisticRegression(object):
    def __init__(self, input, label):
        self.x = input
        self.y = label
        self.W = np.zeros((self.x.shape[1], self.y.shape[1]))
        self.b = np.zeros(self.y.shape[1])

    def least_squares(self,csv):
        """Parses a urls pair string into urls pair."""
        #print("CEEYESVEE",csv,csv.shape)
        #nums = csv.split(",")
        #num1 = float(nums[0])
        #num2 = float(nums[1])

        #X = np.asmatrix(np.array(num1))
        #T = np.asmatrix(np.array(num2))
        #X = add_ones(X)
        T = np.asmatrix(csv)
        #T = np.hstack((np.ones((T.shape[0], 1)), T))
        xtx = T.T.dot(T)

        #arr = ["weights",np.array([1,1*num2,num2*1,num2*num2])]
        arr = ["weights",xtx]

        return arr

    def interval_calc(self,csv1,csv2):
        #nums = csv.split(",")
        #num1 = float(nums[0])
        #num2 = float(nums[1])

        #X = np.asmatrix(np.array(num1))
        #T = np.asmatrix(np.array(num2))
        X = csv1
        T = csv2
        #print("X",X)
        #print("T",T)
        X = np.asmatrix(X).T
        #X = np.hstack((np.ones((X.shape[0], 1)), X))
        interval = X.T.dot(T)
```

```
        arr = ["weights",interval]
        #return np.array(arr)
        return np.squeeze(np.array(arr))

    def train_parallel(self, scon,lr=0.1, L2_reg=0.00):
        print(self.x.shape)
        print(self.W.shape)
        print(self.b.shape)
        print(self.y.shape)
        lamb = 0.00000004
        dot_hypo = np.dot(self.x, self.W)
        loop_through = scon.parallelize([ x for x in self.x])
            links2 = loop_through.map(lambda word: self.interval_calc(word,self.W)).reduceByKey(lambda
a,b:np.vstack((a,b)))
        for link,rank in links2.collect():
            print(link,rank)
            interval = rank

        #dot_hypo = np.asmatrix(interval)
        dot_hypo = np.array(interval)
        print(dot_hypo)
        print("dothyposhape",dot_hypo.shape)
        print(type(dot_hypo))
        print(type(dot_hypo[0]))
        print(self.b.shape)



        p_y_given_x = softmax_function(dot_hypo + self.b)
        d_y = self.y - p_y_given_x
        self.W += lr * np.dot(self.x.T, d_y) - lamb * L2_reg * self.W

def train(self, lr=0.1, L2_reg=0.00):
        print(self.x.shape)
        print(self.W.shape)
        print(self.b.shape)
        print(self.y.shape)
        lamb = 0.00000004

        dot_hypo = np.dot(self.x, self.W)
        print("dothyposhape",dot_hypo.shape)
        print(dot_hypo)
        print(type(dot_hypo))
        print(type(dot_hypo[0]))
```

```
        print(self.b.shape)
        p_y_given_x = softmax_function(dot_hypo+ self.b)
        d_y = self.y - p_y_given_x
        self.W += lr * np.dot(self.x.T, d_y) - lamb * L2_reg * self.W
        self.b += lr * np.mean(d_y, axis=0)

    def negative_log_likelihood(self):
        sigmoid_activation = softmax_function(np.dot(self.x, self.W) + self.b)
            cross_entropy = - np.mean(np.sum(self.y * np.log(sigmoid_activation) + (1 - self.y) * np.log(1 -
sigmoid_activation),axis=1))

        return cross_entropy


    def predict(self, x):
        return softmax_function(np.dot(x, self.W) + self.b)
```

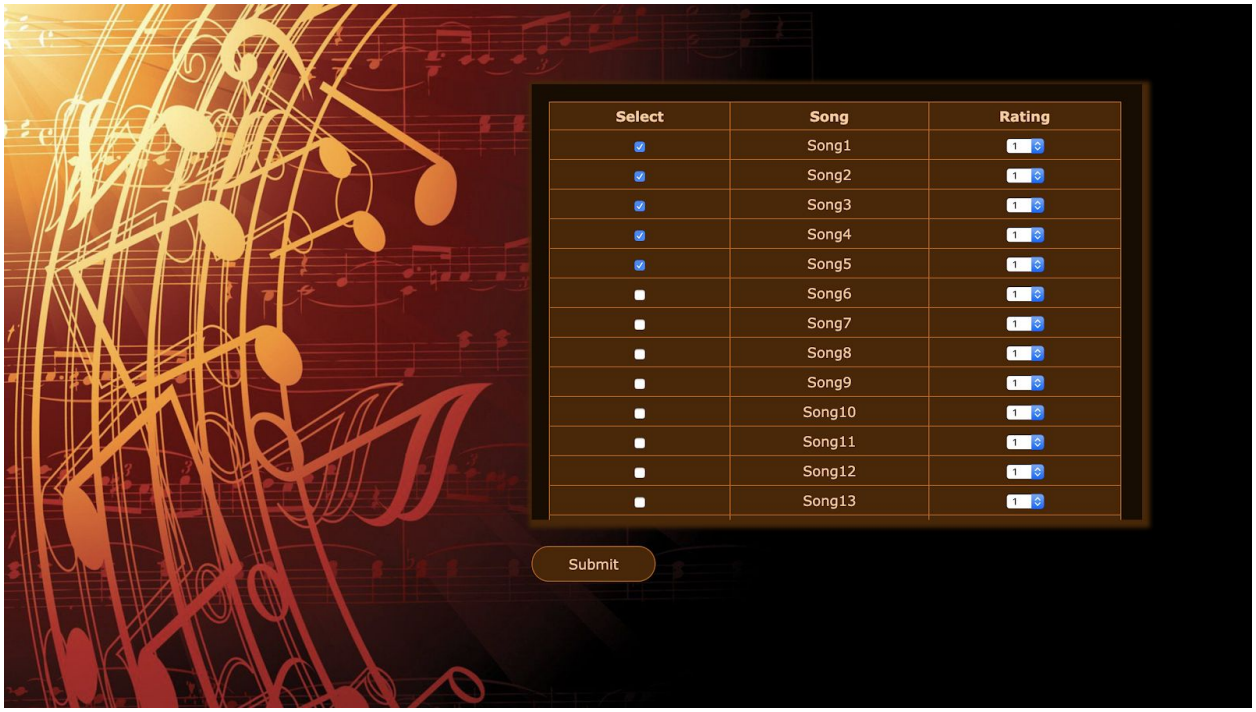
TEST CASES:

One of our test_cases, was running the program for the input “500_rock.csv”, which fed this data of well rated rock songs, and this data was enough to give us a good test case. We noted how the new data which was recommended contains a lot of rock songs . In this case, we can verify our output. The below is a proof of how the model works.

```
[['LGNO' 'Rock' 634.0 10.0]
 ['The Sugar Society' 'Rock' 564.0 10.0]
 ['Ex Lucky' 'Rock' 562.0 10.0]
```

The top 2 recommendations were rock recommendations.

FRONTEND INTEGRATION FOR USER EXPERIENCE:



Division of responsibilities of the team members:

Name	Task
Anirudh	• Environment setup(AWS,Spark MLib) • Logic and Code Development • Research and Brainstorming • Model Evaluation and Testing • Final Report and presentation
Mohit	• Data Pre-Processing • Research and Brainstorming • Model Evaluation and Testing • Final Report and presentation
Nisar	• Data Pre-Processing • Research and Brainstorming • Model Evaluation and Testing • Final Report and presentation
Rahul	• Environment setup(AWS,Spark MLib) • Logic and Code development

	• Research and Brainstorming • Model Evaluation and Testing • Final Report and presentation

Future Scope:

In the current implementation we are not storing the user history and running the model every time user comes. We can implement storing user history, and which can help us to make model more robust.

Source Code:

Least Squares

```
import numpy as np
from abc import ABC, abstractmethod

# Super class for machine learning models

class BaseModel(ABC):
    """ Super class for ITCS Machine Learning Class"""

    @abstractmethod
    def train(self, X, T):
        pass

    @abstractmethod
    def use(self, X):
        pass


class LinearModel(BaseModel):
    """
        Abstract class for a linear model

        Attributes
        ==========
        w       ndarray
                weight vector/matrix
    """

    def __init__(self):
        """
            weight vector w is initialized as None
        """
        self.w = None

    # check if the matrix is 2-dimensional. if not, raise an exception
    def _check_matrix(self, mat, name):
        if len(mat.shape) != 2:
            raise ValueError(''.join(["Wrong matrix ", name]))

    # add a basis
```

```
    def add_ones(self, X):
        """
            add a column basis to X input matrix
        """
        self._check_matrix(X, 'X')
        return np.hstack((np.ones((X.shape[0], 1)), X))

    ##################################################
    #### abstract funcitons ##########################
    @abstractmethod
    def train(self, X, T):
        """
            train linear model

            parameters
            ----------
            X     2d array
                  input data
            T    2d array
                  target labels
        """
        pass

    @abstractmethod
    def use(self, X):
        """
            apply the learned model to input X

            parameters
            ----------
            X     2d array
                  input data

        """
        pass


# Linear Regression Class for least squares
class LinearRegress(LinearModel):
    """
        LinearRegress class

        attributes
        ===========
        w    nd.array  (column vector/matrix)
             weights
    """
    def __init__(self):
        LinearModel.__init__(self)

    def least_squares(self,csv):
        """Parses a urls pair string into urls pair."""
        #print("CEEYESVEE",csv,csv.shape)
        #nums = csv.split(",")
```

```
        #num1 = float(nums[0])
        #num2 = float(nums[1])

        #X = np.asmatrix(np.array(num1))
        #T = np.asmatrix(np.array(num2))
        #X = add_ones(X)
        T = np.asmatrix(csv)
        T = np.hstack((np.ones((T.shape[0], 1)), T))
        xtx = T.T.dot(T)

        #arr = ["weights",np.array([1,1*num2,num2*1,num2*num2])]
        arr = ["weights",xtx]

        return arr

    def interval_calc(self,csv1,csv2):
        #nums = csv.split(",")
        #num1 = float(nums[0])
        #num2 = float(nums[1])

        #X = np.asmatrix(np.array(num1))
        #T = np.asmatrix(np.array(num2))
        X = csv1
        T = csv2
        print("X",X)
        print("T",T)
        X = np.asmatrix(X)
        T = np.asmatrix(T)
        X = np.hstack((np.ones((X.shape[0], 1)), X))
        interval = X.T.dot(T)
        arr = ["weights",interval]
        return arr

    # train lease-squares model
    def train(self, X, T):
        X = self.add_ones(X)
        print("xshape",X.shape)
        print("Tshape",T.shape)
        xtx = X.T.dot(X)
        #print(xtx)
        print("xtx shape",xtx.shape)
        self.w = np.linalg.pinv(xtx)
        print(X.T.shape)
        print(T.shape)
        interval = (X.T).dot(T)
        print("interval shape",interval.shape)
        self.w = self.w.dot(interval)
        print("w shape",self.w.shape)
        self.w = self.w.T
        print(self.w)
        return self.w.T
        ## TODO: replace this with your codes

    def train_parallel(self, X, T,sc):
```

```
        #X = self.add_ones(X)
        print("xshape",X.shape)
        print("Tshape",T.shape)

        loop_through = sc.parallelize([x for x in X])
        links = loop_through.map(lambda word:
self.least_squares(word)).reduceByKey(lambda a,b:a+b)
        for link,rank in links.collect():
            print(link,rank)
            xtx = rank

        loop_through = sc.parallelize([ (x,y) for x,y in zip(X,T)])
        links2 = loop_through.map(lambda word:
self.interval_calc(word[0],word[1])).reduceByKey(lambda a,b:a+b)

        for link,rank in links2.collect():
            interval = rank

        xtx = np.linalg.pinv(xtx)

        for link,rank in links2.collect():
            interval = rank

        rank = np.asmatrix(rank)
        rank = rank


        print("FINAL WEIGHTS = ")
        weights = xtx.dot(rank)
        self.w = weights
        return self.w

    # apply the learned model to data X
    def use(self, X):
        X = self.add_ones(X)
        hypothesis = self.w.T.dot(X.T)
        return hypothesis.T
        ## TODO: replace this with your codes


import collections # for checking iterable instance
```

Stochastic Gradient Descent

```
class LMS(LinearModel):
    """
        Lease Mean Squares. online learning algorithm

        attributes
        ==========
        w        nd.array
                 weight matrix
        alpha    float
```

```
            learning rate
"""
def __init__(self, alpha):
    LinearModel.__init__(self)
    self.alpha = alpha

# batch training by using train_step function
def train(self, X, T):
    iterations = 3000
    X = self.add_ones(X)
    self.w = np.zeros((1,X.shape[1]))
    #print("X is ")
    #print(X)
    #print("W is")
    #print(self.w)
    for i in range(iterations):
        temp = self.w.dot(X.T)
        error = temp.T - T
        new_x = error.T.dot(X)
        self.w = self.w - ((self.alpha)*new_x)
    print("train response")
    return self.w
    pass  ## TODO: replace this with your codes

# train LMS model one step
# here the x is 1d vector
def train_step(self,x,t):
    x = np.insert(x,0,1)
    x = x.reshape(1,len(x))
    #print(x)
    self.w = np.zeros((1,x.shape[1]))
    #print(self.w)
    #self.w = np.zeros((1,x.shape[1]))
    mult_x = x
    error = self.w.dot(mult_x.T) - t
    #print(error)

    update_value = ((self.alpha)*error)*x
    self.w = self.w - update_value
    print(self.w)
    return self.w


def train_step_full(self, x, t):
    x = self.add_ones(x)
    self.w = np.zeros((1,x.shape[1]))
    print(x[0])
    for i in range(x.shape[1]):
        #print(self.w)
        mult_x = np.reshape(x[i],(len(x[i]),1))
        error = self.w.dot(mult_x) - t[i]
        update_value = ((self.alpha)*error)*x[i]
        self.w = self.w - update_value
        #print(self.w)
```

```
    print("returning stochastic")
    return self.w

    pass  ## TODO: replace this with your codes

# apply the current model to data X
def use(self, X):
    X = self.add_ones(X)
    hypothesis = self.w.dot(X.T)
    return hypothesis.T
    pass  ## TODO: replace this with your codes
```

Questionnaire:

Question:1 Why we are using this particular model?

Answer: This is more of recommending songs to user based on his preferences and hence for this approach multiple linear regression works better than any other approach.

Question:2 What other models have we tested and worked with?

Answer: We tried with multiple linear regression with least squares moving on we tried SDG

And then we do tried logit regression to predict the genres.

Question3 : How are we measuring accuracy?

Answer: So we are observing how close our predicted scores are when compared to actual scores.We have given a buffer of 0.5 , so for example if the actual score of a song is 8.75 and are predicted is 9.25 then we say we predicted accurately . Anything above this score is false positive and anything below this is false negative.

Question4: What is our current Accuracy?

Answer: Multiple linear regression :76%

Stochastic Linear regression :86%

Logit Regression: 40%

Question: How we distributed Load (spark training)?

Answer: All matrix multiplication in least squares , SDG and logit is done in spark using parallelization.

Question: What is your dataset? What userData did you generate?

Genre Based Data Selection (Similar Music Type and Genre Selection), using script.

Question:What are the features you used?

Answer: We used Echonest features of audio , FMA Dataset by spotify. Echonest features by spotify. Also we used Genre data, in One hot encoding.

Question: Did you use any feature shortlisting using PCA ?

Answer: We did use PCA for feature transform. Because the accuracy before PCA was not good enough.

Question: Where are we using Cloud Computing:

Answer: We are using Mapreduce to simplify the arduous task of matrix calculation in least squares. The reason for this is it would be faster to do this matrix calculation in parallel rather than in sequence. We focus on performing fast updates, in parallel.

Question: Why not Use Naive Bayes?

Answer: We believe a linear regression (SGD) or Least Squares based approach, makes better sense than this. Most of our features are quantitative, except our genre feature which is one hot encoded, and embedded into our input dataset.

Naive bayes serves better in a classification environment. This being a regression based approach would do better with Gradient Descent or Alternating Least Squares.

Question: What data Cleaning and preprocessing approach?

Answer: The approach we have adopted, uses taking out nulls, duplicates, clearing out columns with majority nulls, and making the data as clean as possible.

Question: Why FMA dataset:

Answer: Provides actual track names, genre types in words, which we believe is better to demonstrate, and has a lot of good spotify recommended echonest music features. It i

Question: What difficulties did you face?

Answer: Hyperparameter Tuning(problem with learning rate, regularization)

Parallel Training, with apache spark

Making apache spark work with the flask backend

Providing better recommendations and adding one hot features to our dataset.

Collaborating Multiple datasets together , to make meaningful information.

References:

[1]https://cse.iitk.ac.in/users/cs365/2014/_submissions/shefalig/project/

[2]https://medium.com/@briansrebrenik/introduction-to-music-recommendation-and-machine-learning-310c4841b01d

[3]https://towardsdatascience.com/how-to-build-a-simple-song-recommender-296fcbc8c85

[4]https://gist.github.com/yusugomori

[5]https://github.com/anirudhnarayanan