ASSIGNMENT 5- Music Pattern Predictor

Abhishek Pai Angle, Aum Jain, Dev Moxaj Desai, Madhumitha S, Tanay Sharma

April 24, 2020

Abstract

This assignment serves as an introduction to pattern prediction. Our aim is to create a music pattern predictor that is able to predict the mood of the user, based on their activity and recommend music to them according to their mood. For this we'll be using Machine Learning- classification algorithms.

1 Introduction

To begin with, our data set consists of 60 songs. Our job is to build an algorithm that can gauge the mood of the listener depending on his/her history and then recommend songs. Also, if a new song is played by any user, our algorithm should be able to classify it.
When a user has finished listening to one song, they can pick one of the following three options:

1. **Next:** in which case we play our next song recommendation
2. **Skip-next:** means the user isn't interested in our recommendation and wants to hear the next recommendation.
3. **Search:** implies the user did not like our suggestion and wants to play a different song. If this song isn't already in our database, the need to classify it arises.

2 Approach

This problem statement can be overwhelming if just read all together at once, which is precisely what happened. But after our discussion, we decided to split the work needed to be done into sections, to make the problem more comprehensible and to make work distribution easy.

1. Preparing the database.
2. Writing the code:

(a) Algorithm to classify additional songs

(b) Predicting the mood of the listener

(c) Implementing KNN and selecting the next song to be recommended

(d) Updating parameters in case a song is skipped

(e) Scoring the model

2.1 Preparing the database

We were given a list of 60 songs, along with the artists. Now, to classify the songs we had to choose parameters. Since the songs given were all pretty popular, we decided to classify based on the following-

- Happy - Sad
- Romance - Breakup
- Motivation
- Party
- Year of Release
- Prominent Mood
- Artist

We've rated a song as happy or sad on a scale of -1 to 1: -1 being the saddest and 1 being the happiest. Same for romance - breakup. Motivation and party were rated on a scale of 0 to 1 (0 being the least to 1 being the most motivational/upbeat-party song). For each song, the parameter that had the highest value (magnitude wise) was set as the Prominent Mood. You can view the data set here- [1].

The idea is to treat each of these parameters as a dimension in the vector space. In case of artists, each artist will be one dimension. Each song will be given a value for every parameter, thus defining its position in the vector space.

This in no way is the complete visualisation of our n-dimensional vector space but 3-dimensions is the max that could be plotted . If the only parameters considered were 'Happy-Sad', 'Romance-breakup' and 'Party', this is what our vector space would look like (along with colouring based on the Motivational column). You can check out the code for this here- [4]

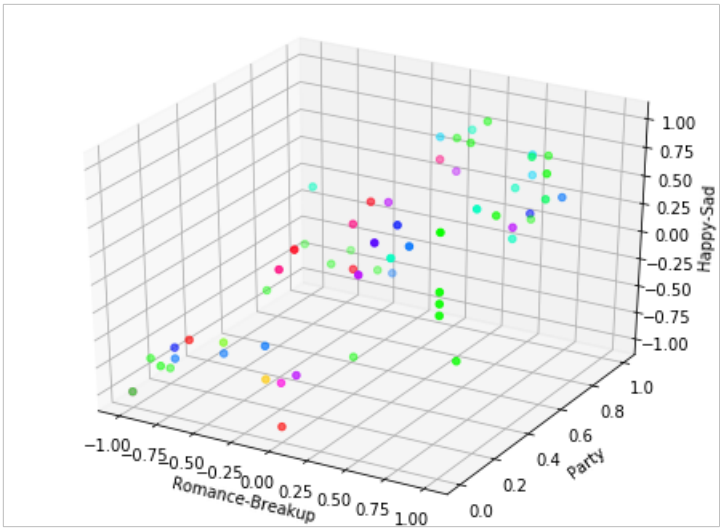

Figure 1: 3-Dimensional Vector Space

So, now that we've gotten our database ready with required classifications, we move on to writing the code. We'll be giving each of these parameters weights, based on their importance when suggesting a song.

2.2 The Code

As mentioned earlier, we've divided this into five sub-sections as follows:

2.2.1 Algorithm to classify new songs

If a new song is played by any user, our algorithm should be able to rate its parameters, since it's impractical and time consuming to manually rate any more songs in a real life scenario of millions of songs.
This is how it **works**: say a user listens to a new song that's not in our database, we need to plot its position in the vector space. For this, we make use of his/her listening history, and we position the new song at a point *equidistant* from the songs he previously listened to.

2.2.2 Predicting the mood of the listener

Our next task is to predict the mood of the listener. This is fairly straight forward. We find the weighted mean of the previous five songs heard by the user and this gives us a point which coincides with the mood of the listener. Note: The weightages given are- **0.5, 0.25, 0.15, 0.05, 0.05**, 0.5 for most recently heard and so on. It is sort of intuitive to give the most recent song

greater weightage, as the user would prefer to hear something similar to his most recent song.

2.2.3 Implementing KNN and selecting the next song to be recommended

A brief on KNN: KNN stands for K Nearest Neighbours. It classifies unlabeled points based on its nearest neighbours.

For example, to classify the red dot(X) as Classe A or Classe B as in Figure 2, the algorithm:

- Calculates the distance from X to all points in the data.
- Sorts the points in the data by increasing distance from X.
- Predicts the majority label of the 'k' closest points.

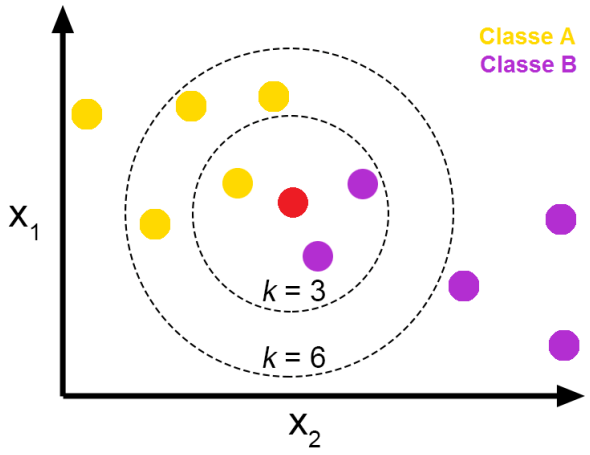

Figure 2: KNN classification

We first implement this method, to obtain a sorted list of neighbours for the mood point(from nearest to farthest).

Now to select the song to be suggested, we use the mood of the listener and find the song closest to that point. This gives us the song most similar to the viewer's taste, and thus our recommendation.

One thing to be careful about is that we can't repeat songs, i.e predict the same song again, or predict the previous. Therefore, after finding our nearest neighbours, we must exclude previously played songs.

Also, if the current recommendation is skipped, the next recommendation (second nearest neighbour) should be played.

Code snippet for this-

```
# this function calculates the k nearest neighbors of 'point' which
    is a n-tuple and playlist is a list of tuples
def getKnearest(point, playlist, dist_fn, k):
   distances = [(dist_fn(playlist[i], point), i) for i in
       len(playlist)]
   distances = sorted(distances, reverse=False)
   KNN_indices_distances = distances[:k]
   return KNN_indices_distances

#eazy to compute and was making more sense xD
def Euc_dis(point1, point2):
   dis = 0
   for i in length(point2):
      dis += abs(point2[i]-point1[i])
   return dis


def driver(playlist, point, k):
   nearest_list = getKnearest(point,playlist,Euc_dis,k)
   nearest_indices = map(lambda x: x[1],nearest_list)
   return nearest_indices

def suggest(X,y,point,j):
    next_songs=driver(X,point,60)

    for i in range(60):
      if next_songs[i] not in played and next_songs[i] not in
          skipped:
        suggestion = next_songs[i]
        break
    print("Would you like to play this song next ?")
    print("1: ",y[suggestion])
    print("2:  Skip")
    print("3:  Search")
    print("4:  Exit")

    return suggestion
```

2.2.4 Updating parameters in case a song is skipped

This section deals with the scenario of a song suggestion being rejected. If a user enters **skip - next**, it means they weren't interested in the song we

recommended and want to hear the next song. In such a case, we would have to make changes in our source code, i.e. modify the ratings of that song since it did not concur with the mood of the listener. Here's the function "update" that performs this-

```
def update(Song1 ,Song2):
    alpha= 0.01
    for i in range(1,5):
        Song2[i] =Song2[i] + alpha*(Song2[i]-Song1[i])
```

2.2.5 Scoring the model

This is a function to check the accuracy of the model. Our accuracy metric for grading the model is as folows:

$$Accuracy = 1.0 - (number of skipped songs / number of suggestions)$$

3 Final Code

```
import numpy as np
import pandas as pd

dataset=pd.read_csv('Cleaned_Input.csv')
Cols = list(dataset.columns)
X=dataset.iloc[:,1:].values
y=dataset.iloc[:,0:1].values

#Songs is the old songs in the history of the person that we use to
    classify this new song(indices)
#data is the list of songs(not the name of songs, the coefficients)
#artist is the artist of the song coz the artist shall not depend
    on the history right
#The artist is a number, representing the column of the artist in
    the data
#Song_Names is the list of song names(As the user searches via
    names)
def get_coorrds(Name_of_new_Song, Songs, data, artist, Song_Names,
    timestamp):
   new_song, narr = np.zeros([1,len(data[0])]),
       np.array([Name_of_new_Song], dtype=str)
   for i in range(4):
      #List all the values for a particular emotion
      L = [data[j][i] for j in Songs]
      new_song[0,i] = sum(L)/len(L)
```

```
   #making the index of artist 1

   new_song[0,artist], new_song[0,timestamp]=1,1

   #append the new song to the data

   data = np.concatenate((data, new_song))
   #Since the user searches for a new song by name

   Song_Names = np.append(Song_Names, narr)
   return new_song


#this function is to update the data
def update(Song1 ,Song2):
    alpha= 0.01
    for i in range(1,5):
        Song2[i] =Song2[i] + alpha*(Song2[i]-Song1[i])

# this function calculates the k nearest neighbors of 'point' which
    is a n-tuple and playlist is a list of tuples
def getKnearest(point, playlist, dist_fn, k):
   distances = [(dist_fn(playlist[j], point), j) for j in
       range(0,len(playlist))]
   distances = sorted(distances, reverse=False)
   KNN_indices_distances = distances[:k]
   return KNN_indices_distances

#eazy to compute and was making more sense xD
def Euc_dis(point1, point2):
   dis = 0
   for i in range(len(point2)):
      dis += abs(point2[i]-point1[i])
   return dis


def driver(playlist, point, k):
   nearest_list = getKnearest(point,playlist,Euc_dis,k)
   nearest_indices = list(map(lambda x: x[1],nearest_list))
   return nearest_indices

def suggest(X,y,point,j):
    next_songs=driver(X,point,60)

    for i in range(60):
      if next_songs[i] not in played and next_songs[i] not in
          skipped:
        suggestion = next_songs[i]
        break
```

```
    print("Would you like to play this song next ?")
    print("1: ",y[suggestion])
    print("2:  Skip")
    print("3:  Search")
    print("4:  Exit")
    print("5:  Introduce New Song")

    return suggestion
def search():
    print("Select the Song")
    for i in range(60):
        print(i+1,y[i])
search()
played = []
skipped = []
x=int(input())
t=0
prev_song=[X[x-1]]
l=suggest(X,y,prev_song[len(prev_song)-1],t)
played.append(x-1)
while(len(prev_song)<5):

    x=int(input())
    if(x==1):
        prev_song.append(X[l])
        played.append(l)
        t=0
        l=suggest(X,y,prev_song[len(prev_song)-1],t)
    if(x==2):
        t=t+1
        skipped.append(l)
        l=suggest(X,y,prev_song[len(prev_song)-1],t)

    if(x==3):
        search()

    if(x==5):
      Name_of_New_Song = str(input("Enter name of new song"))
      artist = 'Artist_' + str(input("enter name of artist"))
      timestmp = 'Timestamp_'+str(input("TimeStamp of song"))
      artist_ind = Cols.index(artist)-1
      time_ind = Cols.index(timestmp)-1
      get_coorrds(Name_of_New_Song, played, X, artist_ind, y,
          time_ind)
      print("Song Added")


while(x!=4):
  k = len(prev_song)
```

```
point=prev_song[k-1]*0.5+ prev_song[k-2]*0.25+
    prev_song[k-3]*0.15+ prev_song[k-4]*0.05+ prev_song[k-5]*0.05
l=suggest(X,y,point,t)
x=int(input())
if(x==1):
    prev_song.append(X[l])
    played.append(l)
    if len(played)>10:
      played.remove(played[0])
    t=0
if(x==2):
    skipped.append(l)
    if len(skipped)>10:
      skipped.remove(skipped[0])
    t=t+1
if(x==3):
    search()
if(x==5):
    Name_of_New_Song = str(input("Enter name of new song"))
    artist = 'Artist_' + str(input("enter name of artist"))
    timestmp = 'Timestamp_'+str(input("TimeStamp of song"))
    artist_ind = Cols.index(artist)-1
    time_ind = Cols.index(timestmp)-1
    get_coorrds(Name_of_New_Song, played, X, artist_ind, y,
        time_ind)
    print("Song Added")
```

4 Bugs Faced

1. We had a hard time working with dataframes having spaces in columns. *Solution:* solved the issue by removing all null characters.

2. We struggled with the shape of the arrays for concatenation, as concatenation was required for classification of new songs. We realized it quite late and then looked at the shape of our array using *.shape()* function.
   Solution: If the *'inplace'* argument is set to true in the *getdummies()* function it makes necessary changes in the same data frame. It is set to false by default It was hard to figure out why the dataframe wasn't getting modified

5 Work Contribution

Preparing the dataset - Tanay and Madhumitha
Updating parameters function - Dev
KNN and classification - Tanay
Song selection - Abhishek
Report - Madhumitha

6 Conclusion

In conclusion, this assignment has given us a lot to learn. Since we were expected to code the classification algorithms from scratch, we got a better understanding of how they work. And this being our first assignment involving pattern recognition, has provided some great insights.
From another angle, one big takeaway for us, is to get more accustomed to using GitHub so everyone's contribution is visible which will make us all more involved. Also, to discuss with Sanju in case we're stuck, and not just... remain stuck.
But overall, this project was really interesting, and we're excited for what's next!

References

[1] Songs Dataset- https://github.com/Intelligent-Agents-XX/music-pattern-predictor/blob/master/Songs-

[2] KNN image- https://towardsdatascience.com/knn-k-nearest-neighbors-1-a4707b24bd1d

[3] LaTeXreference- https://www.overleaf.com/learn/latex/Learn-LaTeX-in-30-minutes

[4] 3D plot code-


```
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

fig = plt.figure()
ax = plt.axes(projection='3d')

z_points = dataset['Happy-Sad']
x_points = dataset['Romance-Breakup']
y_points = dataset['Party']
ax.scatter3D(x_points, y_points, z_points,
    c=dataset['Motivational'], cmap='hsv');
```