Pick-A-Tune

Phuc Pham, Samuel Oates, and Matthew Zelenin

December 10, 2018

I Challenges and Constraints

The objectives of our project is to make a machine learning that would learn to categorize the songs and gives it back to the user as recommendation to the songs that they have input. Some of our constraints were that we would only limit the amount of songs that could be used to configure the recommend songs. We had to narrow the amount of keywords used to make split within the decision tree. Another constraint was that we had to figure out how many songs we were going to return as the recommend songs.

I.1 Major Challenges

Some of the major challenges comes with lack of understanding with the overall usage of decision tree works and the construction of a functional decision tree. We had to look over the notes that were given to us in class and dig a bit more into researching the overall build of decision tree. After our first check-in, we decided to ask our mentor for help on how the decision tree should be structure. They were able to provide us with links that were tremendously helpful to the design of the tree. One of the major problem for us was the data in-which we would be using to formulate our data. At first, we tried to use music brains, but the data was too big and it took over 20 minutes to look up a single songs, so we had to look for a different database and we were able to find Spotify API which helps cut down the search run time. Another major problem we ran into was the way to take information from the Spotify API and convert into readable data for our code. There was some online research and Matt and Sam were able to find a solution. Then there was a question on how we would test if the songs would be consider a recommend song for the user. Since this is more of a personal preference of the user, we had to use other means to test our songs. We thought to try to look at the similarity of the artist, mood, beats, or many other attribute. Hopefully what come up with could be consider correct.

I.2 Variations

The variations that came up was how to design the decision tree and which data base we could have use. As mention above, we finally were able to make a decision on Spotify API being our database which helps us with minimizing the runtime of the overall algorithm since you get a constant number of songs through quick search rather than music brain which took too long. We also did take a look at the previous project to see if we could use that implementation for our decision tree, but alas the assignment decision tree was highly confusing and Phuc could not seem to get the basic of it to use for our decision tree. Therefore, we ask the mentor for help which they provided us with links that ultimately helps create our current decision tree.

II Machine Learning Task

The machine learning task is to use the decision tree to correctly formulate a recommend playlist. We had to use multiple iteration of how it would split the tree and look at it if that was the best split in our mind mostly because it is difficult to pin point if a song would match another songs. We tried to make it so that some songs with similar mood and loudness would stay close together with their value and greatly distance songs that had little connection with one another. Trying to accomplish this was challenging to say the least since the AI would sometime split upon other value that group the songs together that did not match one another. By making sure that the decision tree function correctly, we tried to use some of the training data on it. We had five songs with very limit value and we slowly increased the amount of that the decision tree can split on.

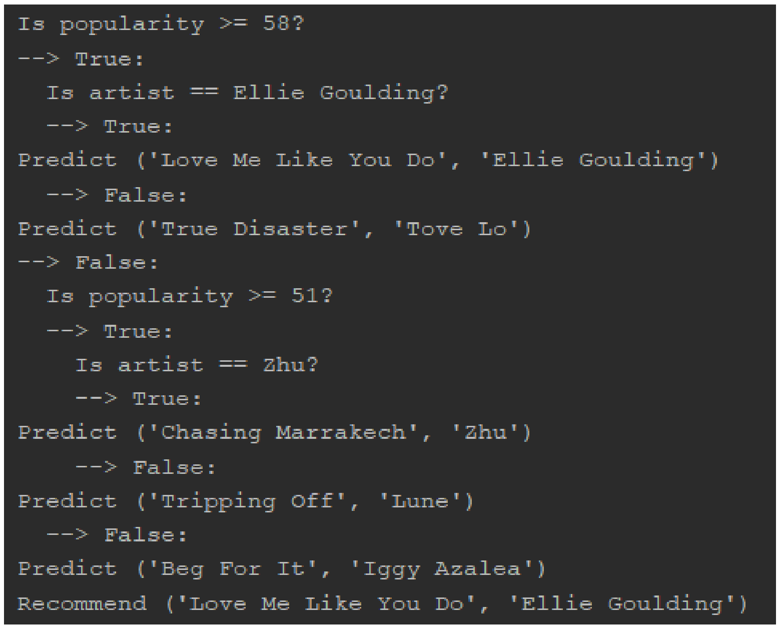
III Algorithm & Runtime

The algorithm that we chose is a machine learning algorithm called decision tree. Decision tree is a decision support that structure itself like a tree and at every node of the tree, a decision is made to split the data. The decisions are made by calculating how many datapoints would be group together into two branches,usually true or false, of a node. This is call the information gains and it used the input data to help the grouping of data. The information from the input data can be resource costs, utility, chance event outcomes, and many other attributes. There are many ways to calculate the information gain. In class, we learned to calculate the information gained using entropy, but after researching, we have found that gini impurity is seemed produce a better runtime because it does not require logarithmic computation. Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. This can be computed by calculating the probability of the data multiply by the probability of a miscalculation of that data. By using the decision tree with gini impurity, the runtime is worked out to be about O(mn log n) where O(m) is the number of attributes within used for every data and the partitioning of the branches takes about O(n log n).

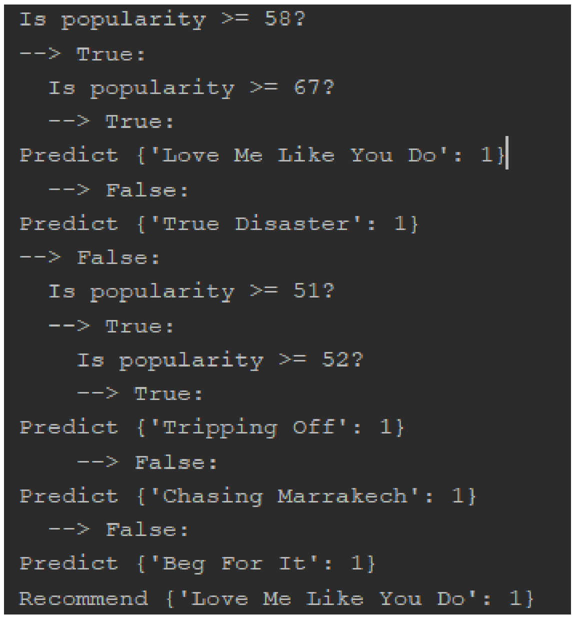
III.1 Limitations

We did not want to make the decision tree too complex because it would not have generalize the training data well enough for the output. We also saw that if we made small changes upon the training data, this cause the overall tree to change which does not make it robust.

IV Empirical Analysis

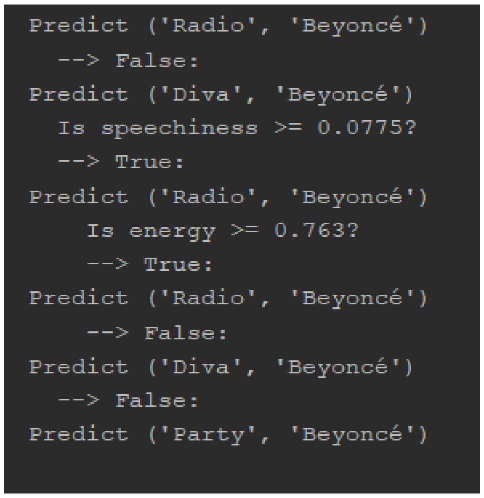


```
Is popularity >= 58?
--> True:
  Is artist == Ellie Goulding?
  --> True:
Predict ('Love Me Like You Do', 'Ellie Goulding')
  --> False:
Predict ('True Disaster', 'Tove Lo')
--> False:
  Is popularity >= 51?
  --> True:
    Is artist == Zhu?
    --> True:
Predict ('Chasing Marrakech', 'Zhu')
    --> False:
Predict ('Tripping Off', 'Lune')
  --> False:
Predict ('Beg For It', 'Iggy Azalea')
Recommend ('Love Me Like You Do', 'Ellie Goulding')
```



```
Is popularity >= 58?
--> True:
  Is popularity >= 67?
  --> True:
Predict {'Love Me Like You Do': 1}
  --> False:
Predict {'True Disaster': 1}
--> False:
  Is popularity >= 51?
  --> True:
    Is popularity >= 52?
    --> True:
Predict {'Tripping Off': 1}
    --> False:
Predict {'Chasing Marrakech': 1}
  --> False:
Predict {'Beg For It': 1}
Recommend {'Love Me Like You Do': 1}
```

The picture on the left is the results of our training data with having artists as one of the attribute that would splits the tree. The one on the right is the results of removing artists from the training data. We have decided to remove artist since it would make it too precise and the other thing we are trying to fix is that the algorithm seem to be recalling the same attribute to split the tree. This is not a result that we expected to happen so we tries to correct it the best we could. This imply that we need more attributes to help split the tree well.



```
Predict ('Radio', 'Beyoncé')
  --> False:
Predict ('Diva', 'Beyoncé')
  Is speechiness >= 0.0775?
  --> True:
Predict ('Radio', 'Beyoncé')
    Is energy >= 0.763?
    --> True:
Predict ('Radio', 'Beyoncé')
    --> False:
Predict ('Diva', 'Beyoncé')
  --> False:
Predict ('Party', 'Beyoncé')
```

This is the final product of reworking the code. Sam was able to take in data produce by Matt and expanded the attributes that could be use to split the decision tree. The code does not reuse the same attribute to split the tree multiple time, which meant that the problem with the original iteration has been fixed.

V Outside Source Code

The gini impurity is used to calculate the information gain that would be used to decide whether a split is good or not. It looks at how similar the data is in a list and give that list a value in which is used to calculate th information gains. This is made by Google Developer(Gini Source)

VI Improvements?

To improve the AI, we should have implement decision tree with pruning to see how well it would have work with our based data and advances the type of splits that we are making. We would have like to see if gini impurity is the fastest way to calculate information gains or if a better computational function could be use. We learned a lot more about the basic of the decision tree and how to computed information gains using both entropy and gini impurity. we all definitely mess around with the code to see which fits better and gini impurity was better, but not by a huge margin. Sam and Matt really enjoyed learning how to use the Spotify API to better improve the data to the point where they were able to get an abundant of data for testing and training.