Rock, Pop, and Hip-Hop:
Using Decision Trees to Distinguish Musical Styles

DS4400: Machine Learning I

Prof. Rachlin

Introduction

We have seen how machine learning can be used to analyze text and distinguish political speech. In this homework we ask how well can we distinguish musical genres based on an auditory analysis of 3000 songs categorized into 3 genres: Rock, Pop, and Hip-Hop (Rap).

This is another class-driven collective experiment!

We have seen in class how there are many ways to build a decision tree. We discussed several feature selection criteria focusing in particular on the Gini Impurity Index and Entropy. We also saw how other stopping criteria may enable us to avoid over-fitting our data. In this homework, I want you to be more rigorous in your machine learning experimental methodology. You will split your training data, consisting of 2000 songs into two subsets: Training and Validation. Then you will determine model accuracy by performing 10-fold cross validation by testing 10 trained models on 10 validation sets. Your average accuracy will then drive hyperparameter tuning as you search for the best model. Finally, you will report the accuracy of your best model on a separately provided test set of 1000 songs. We'll see which team can come up with the most accurate musical genre predictor. I'm expecting validation accuracies of at least 75%. That's pretty good considering that simply guessing would only give us an accuracy of about 33% (because we have three classes.)

Instructions

All algorithms and evaluation metrics should be implemented by you. <u>You may not use Scikit-learn or other machine learning libraries in your solution</u>. You are free to use non-ML-specific python libraries such as pandas, numpy, random, matplotlib, collections, etc. You may also use any code developed in class as part of the labs except where I demonstrated Scikit-learn of course!

Part A. Decision-Tree Construction

This is the heart of the assignment. Extend the code I developed in class for building recursive binary trees. Implement a function called **dtree** which takes the following parameters:

- **train**: A training dataset **(pandas dataframe)**

- **criterion**: The attribute selection method used to find the optimal split. **(*gini or entropy)***

- **max_depth=None**: The maximum allowed depth of the tree. If None, there is no maximum. Note that the root node of the tree is at depth 0. **(integer >= 0)**

- **min_instances=2**: The minimum number of (heterogeneous) instances required to perform a further split. In one of the labs, I referred to this parameter as *min_vals*. Obviously you can't split only a single instance, and if all the instances have the same class there is no need to split further. **(integer >= 2)**

- **target_impurity=0.0**: The target impurity (measured using the chosen attribute selection measure) at or below which we halt node splitting**.** **(double in range 0.0 to 1.0).**

The output of the function is your model, represented as a tuple of tuples, as we did in class. I recommend each tuple contain the following information:

- feature / column name (splitting criterion)
- feature value threshold (splitting criteria)
- examples_in_split
- majority class
- impurity_score
- depth
- left_subtree (leading to examples where feature value <= test threshold)
- right_subtree (leading to examples where feature value > test threshold)

Part B. Model Accuracy

Use your model to evaluate your validation data (and eventually your test data). You'll need to implement a function to generate a series of predictions. It should be something like this:

def predict(model, data):

The function should return a series of predictions that you can then use to measure model accuracy.

Remember to use 10-fold cross validation to determine your overall validation error.

Part C. Hyperparameter tuning

Search for a combination of hyperparameters that produces the best *validation* accuracy. The available hyperparameters are:

- Feature selection and splitting criterion (Gini or Entropy)
- Stopping criteria (max_depth, min_examples, target_impurity)

Part D. Test your final model

Using your best hyperparameters, build a final model using all 2000 songs in the training dataset. Run this model against the test set and report your final accuracy. It is important that you follow the guidelines of the experiment:

1. Training data and only training data is used to build the model

2. Validation data is used to estimate model performance in search of the best hyperparameters.

3. Test data is for your final estimate of performance. Simply report this accuracy. Do not tune your model to maximize test accuracy and do NOT use any test data to train your model. Pretend like we don't know the musical genre of the test examples and simply report your overall accuracy. This is likely to be lower than the validation error of your best model.

What to Submit

- Code (python files or Jupyter notebooks). <u>If submitting a Jupyter notebook, include a PDF version of your notebook to facilitate grading and make sure all cells have been executed and are displaying output.</u>

- A printout of your best model. To print the model, it is sufficient to print the key attributes of each node: Test criterion, depth, # instances, impurity, majority class. You might try inserting a number of tabs before each node's printout equal to the depth of the node to show the hierarchy of your decision tree. At minimum, be sure to clearly show the selection criteria for all nodes at depth 0, 1, and 2. PLEASE DO NOT EXPECT THE TAs TO EXECUTE YOUR CODE IN ORDER TO SEE YOUR RESULTS!

- Summarize your accuracy results on the Google Sheet referenced on the Canvas assignment. Let's see who can find the best, most accurate, model! I may award a small amount of extra credit to the top solutions.