CS 4641 Machine Learning
Name Redacted — Section B Final Report

1 Introduction to Dataset

1.1 Music Features from Kaggle (← link)

Every year I look forward to Spotify's end of year wrap up. As a consumer of sounds, I love seeing which "conventional categories," AKA genres, of music I sink thousands of hours in. This past year a new one made itself to the top of my list: Cpop. After some research, that stands for Chinese Pop. I've literally never heard of songs being labeled in this kind of genre before which led me to wonder how can someone categorize all the various kinds of music in the world.

TODO: Feature description. When I found this dataset, I was already interested in seeing if machine learning could analyze the sound waves of a piece of music and determine it's genre. However, that seems a too big of a computational task so I decided to use the features that are provided by this dataset. This dataset contains **1,000 datapoints** with **30 features per datapoint**. To prove that the features describe a piece pretty well, here are some of the features:

- tempo: speed at which a passage of music is played
- beats: basic unit of time. Think about it as the rhythm you would tap your foot to when listening to a piece
- chroma_stft: The Short Time Fourier Transform can be used to determine the sinusoidal frequency and phase content of local sections of a signal as it changes overtime (kind of like those visualizers that jump up and down)
- 20 mel-frequency ceptral coefficients that make up a mel-frequency cepstrum: representation of the short term power spectrum of a sound

1.2 Supervised Learning Problem

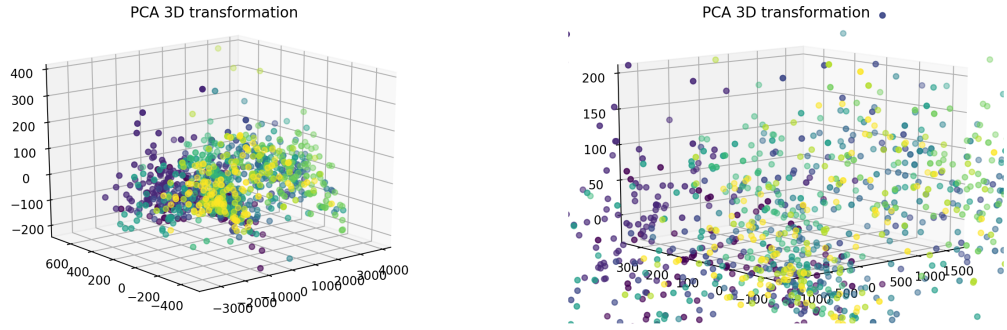
The underlying supervised learning problem is to **classify the genre** of pieces of music using the features of it's waveform. This dataset is **not trivial**. At first glance, there is no feature that determines the genre of a song trivially. To prove the non-triviality of this

dataset, I tried to fit the centered data using sklearn.svm.LinearSVC (with a one-vs-the-rest multiclass scheme) 100 times and averaged the results.

- avg training accuracy: 0.62
- avg testing accuracy: 0.71

The training accuracy is not high enough to prove that the dataset is linearly separable. But to further research the triviality of the dataset, I did a few more experiements:

- 3D PCA Plot



  The graph on the right is a zoomed in version of the left graph. It is clear that there are many data points mixed together and the dataset is at least not linearly separable in 3 dimensions.
- Linear Regression: I one-hot-encoded the labels and tried to fit a linear regression. The testing and training accuracy was 0.26 and 0.31 respectively. This serves as more evidence that the dataset is non-trivial.

After these additional experiements, I have convinced myself that my **dataset is non-trivial**.

1.3 Performance Metrics

I will be using **K-Folds Cross Validation** for hyperparameter tuning. In K-Folds Cross Validation, I will split the data into a training and testing set. Then I will split the training data into k subsets and use k-1 subsets to train and leave the last subset as validation. I'll be using sklearn's GridSearchCV to accomplish this.
I also plan on analyzing the **confusion matrix** of resulting test accuracies for each learning method and using **confidence intervals** as a means of comparing learning methods.

2 Description of Algorithms

2.1 Random Forests with Bagging

A random forest consists of a large number of individual decision trees that form an ensemble. An ensemble bascially uses multiple algorithms to obtain more accurate predictions than any

one individual of the ensemble. In a random forest, the ensemble is made up of relatively uncorrelated decision trees. Bagging (Bootstrap Aggregation) is a method that allows each decision tree in the random forest to randomly sample the dataset with replacement. Since decision trees are very sensitive the training data, this greatly increases the variablility of the decision trees. Here are the hyperparameters I will optimize:

- n_estimators: [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
  Number of trees in the forest. Affects the size of the random forest.
- max_depth: [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
  The maximum depth of each decision tree. affects the complexity of each tree.
- max_features: ['sqrt', 'log2', None]
  The max size of the random subsets of features to consider when splitting a node.

2.2 Support Vector Machines with a non-linear kernel

SVMs attempts to separate datapoints by its labels by finding the best hyperplane using the large margin principle. For non-linear data, it can use the kernel trick to map the data to a higher dimension in hopes of finding a more suitable hyperplane. However, SVMs are inherently binary classifiers so it is not suitable for my dataset that has 10 unique labels. I will be using the **one-versus-all** classification technique to train my multiclass SVM. Here are the hyperparameters I will optimize:

- C: [1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03, 1.e+04]
  Regularization paramenter represents how much I want to avoid misclassifying each training example. Large C will choose the a plane that correctly classifies the most training examples. Small C will choose a plane with a larger margin even if that plane wrongly classifies more training examples
- kernel: ['poly', 'rbf', 'sigmoid']
  The kernel type to use when mapping the data to a new dimension space
- gamma: [1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03]
  Kernel coefficient for rbf, poly, and sigmoid.

2.3 Neural Network

1
2
3