Interpreting and Classifying Music

New Mexico Supercomputing Challenge

Final Report

April 4, 2018

LAMS_155

Los Alamos Middle School

Team Members:

Andrew Corliss

Maximilian Corliss

Phillip Ionkov

Ming Lo

Teacher:

Ellie Simons

Mentor:

Li-Ta Lo

Executive Summary:

The purpose of this project was to create a program that could identify different genres of music. This is a difficult goal because of the inherent intricacies in music. Many different characteristics such as tempo and key signature define musical genres, as well as arbitrary human concepts such as time period. Our approach to solving this problem was using a few characteristics that were easily identifiable by computers, namely frequency ("pitch") and volume. We began using some supervised machine learning algorithms (outlined in Scikit-learn) such as KNearestNeighbor and the Decision Tree Classifier on datasets such as the Iris dataset and the Tensorflow Simple Speech Dataset. Our final classifier can predict the Simple Speech Dataset with 86% accuracy.

Problem Definition:

Music is a complicated and diverse art form. It has evolved over many centuries to form many different types, or genres. Each genre has different characteristics, such as key signature, time signature, and tempo, that are used to distinguish separate varieties. We want to make a program that given an audio file, can classify which music category it belongs to. Eventually, this program could be used by the music industry through classification of genres, as well as outside its initial scope with applications in speech recognition and other fields.

Methods:

To solve this problem we decided to explore a variety of supervised Machine Learning algorithms outlined on Scikit-learn[1] . Scikit-learn is a website full of machine-learning programs and their potential uses and available from the sklearn Python module. Supervised machine learning starts by training a machine learning model on training data. Then, the trained model is used to predict the classification of unknown data. We have set up a shared Jupyter server in Google Cloud which provides more disk space and computing power than our individual laptops. It also allows us to easily share our code. We used algorithms like K Nearest Neighbors (a classifier that represents data on a graph and determines its label based on proximity to known values using Euclidean distance). We also used the Decision Tree Classifier,

which creates a decision tree based on labels from training data and applies this tree to test data. Then we implemented these methods to a speech dataset from TensorFlow Speech Recognition Challenge on Kaggle, [5] with the aforementioned classifiers and others from Scikit-learn [1] to find out which methods work best for audio categorization.

Once we get familiar with some basic machine learning algorithms, we moved to the next step: reading music from audio files and transforming it to a set of values that can be used by the machine learning code. We used the LibROSA[7] package to read music files.

Results:

We have not reached the point of classifying music, but we have used many classifiers for the aforementioned machine learning exercises. When testing our classifiers, we used both training and test data. These types of data were split randomly between 80% of data used for training data and 20% used for test data.

We used measures to show and compare the performance of different classifiers we used; prediction accuracy and confusion matrices. A confusion matrix is a table that shows the data points that were predicted by the computer and whether the prediction was valid or incorrect.

First we experimented with implementing classifiers using the Iris Dataset [3]. It contains information about 50 samples for each of three iris species. There are four measures for each sample: the length and the width of the sepals and the petals. Table 1 shows the confusion matrix from Figure 1, which classified Iris flower species from their petal and sepal sizes:

	Setosa (actual)	Versicolor (actual)	Virginica (actual)
Setosa (prediction)	5	0	0
Versicolor (prediction)	0	12	2
Virginica (prediction)	0	0	11

Table 1

We can see that the classifier predicted 5 data points as Setosa flowers, then predicted 14 data points as Versicolor, and 11 data points as Virginica. However 2 data points were incorrectly classified as Versicolor and were actually Virginica.

Figure 1 shows the code for the KNearestNeighbors classifier used for the Iris Dataset. The KNearestNeighbors classifier uses training data, or "neighbors", in order to classify an unknown datapoint. K is a parameter that describes the number of nearest neighbors that surround the unknown testing point to compare to. The classifier will compare the unknown data point with points from known data to find K "neighbors" with the most similar features. The majority of neighbors determines which classification the unknown datapoint is given. For example, if an unknown Iris data point had more Versicolor neighbors than Setosas, the classifier determines the point as a Versicolor flower.

The dataset for the Iris flowers are split randomly between 80% training data and 20% test data. Therefore, the accuracy varies between 90%, 93%, 97%, and 100% for this type of classifier.

```
In [1]:
    1 from sklearn import datasets
    2 from sklearn.model_selection import train_test_split
    3 from sklearn.tree import DecisionTreeClassifier
    4 from sklearn.neighbors import KNeighborsClassifier
    5 from sklearn.metrics import confusion_matrix

In [2]:
    1 iris = datasets.load_iris()

In [3]:
    1 x = iris.data
    2 y = iris.target

In [4]:
    1 # randomly split data into 80% of training set and 20% of testing set
```

```
    2 x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

In [5]:
    1 classifier = KNeighborsClassifier()
    2
    3 classifier.fit(x_train, y_train)
Out[5]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                             metric_params=None, n_jobs=1, n_neighbors=5, p=2,
                             weights='uniform')

In [6]:
    1 classifier.score(X=x_test, y=y_test)
Out[6]: 0.96666666666666667

In [7]:
    1 y_pred = classifier.predict(x_test)

In [8]:
    1 confusion_matrix(y_pred=y_pred, y_true=y_test)
Out[8]: array([[5,  0,  0],
               [ 0, 12,  2],
               [ 0,  0,  11]])
```

Figure 1, K Nearest Neighbors code for Iris dataset

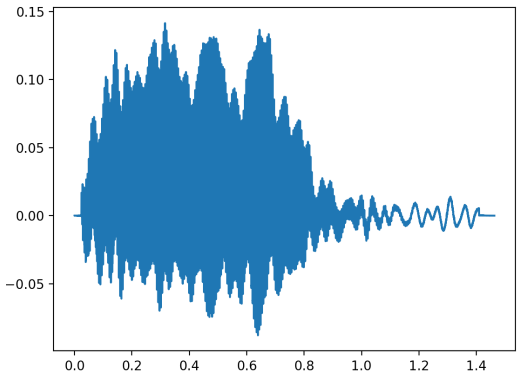
Figure 2 shows the code used in the Decision Tree classifier for the Iris dataset. The Decision Tree classifier has different values of tipping points to classify an unknown datapoint, similarly to a flowchart. After reaching a decision, it moves on to another decision until the datapoint is classified. The first four inputs are exactly the same as those shown in Figure 1 because the Scikit learn classifiers are similar and it is very easy to switch between different

classifiers. The accuracy for this classifier varies between 87%, 90%, 93%, 97%, and 100%. This is because it also splits the dataset randomly between training data and test data.

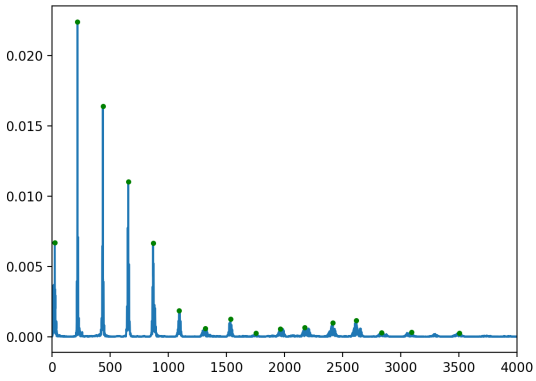
```
In [5]:
     1 # create a DecisionTree classifier
     2 clf = DecisionTreeClassifier()
     3
     4 clf.fit(x_train, y_train)
Out[5]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                max_features=None, max_leaf_nodes=None,
                min_impurity_decrease=0.0, min_impurity_split=None,
                min_samples_leaf=1, min_samples_split=2,
                min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                splitter='best')
In [6]:
     1 clf.score(X=x_test, y=y_test)
Out[6]: 0.93333333333333335
In [7]:
     1 y_pred = clf.predict(x_test)
In [8]:
     1 confusion_matrix(y_pred=y_pred, y_true=y_test)
Out[8]: array([[ 5,  0,  0],
               [ 0,  9,  0],
               [ 0,  2, 14]])
```

Figure 2, Decision Tree code for Iris dataset

Our next step in achieving our goal was to be able to read music files and analyze them. We used LibROSA to read the audio files. We used the NumPy FFT to transform the waveform into the frequency spectrum. Figure 3a shows the waveform of a viola playing an A3 note. Figure 3b shows the frequency spectrum of the waveform of the viola. Figure 4a shows the waveform of a violin also playing an A3. Figure 4b shows the frequency spectrum of the waveform of the violin. Even though both instruments are playing an A3, the waveform and frequency spectrum are different. This is how a computer can discern between the different instruments. This is also how we did our speech recognition later on.



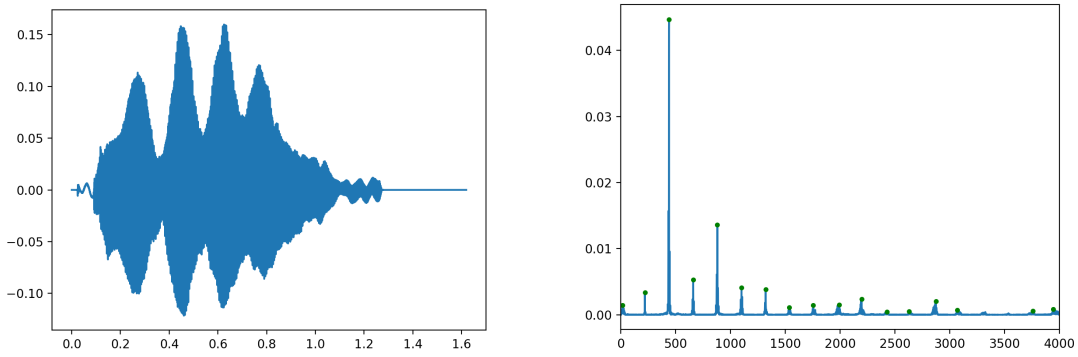
3a



3b

Figure 3a, Waveform of a viola playing A3

3b, viola frequency spectrum



4a 4b

Figure 4a, Waveform of violin playing A3

4b, Frequency spectrum of violin playing A3

After this, we used our knowledge of audio and LibROSA, testing classifiers for speech recognition. Using a dataset from Kaggle, we sorted four words: yes, no, up, and down. We used frequency spectrum (pitch) as and achieved an average 50% with a Decision Tree classifier. We also experimented with different classifiers, such as MLP (multi-layer perceptron) to try to find which is the most effective. Our current version uses a MLP classifier that achieves around 74% success and uses a frequency spectrum feature.

Conclusion:

In conclusion, we split the very difficult goal of music classification into smaller steps. We were able to classify simple speech samples such as “yes”, “no”, “up”, and “down” with a highest accuracy of 74% with the MLP. We also used LibROSA to read music files. However, we were unfortunately unable to achieve our final goal. We learned several of the basic machine learning algorithms and how to apply them. We plan to extend these techniques that we learned in order to achieve our goal.

Acknowledgements:

First, we need to thank our mentor, Ollie Lo, for guiding us in the learning process of programming. We would also like to thank the Mesa Public Library for giving us a quiet work environment where we attained lots of progress. In addition, we would like to thank all of our database sources: FMA, MNIST, Kaggle, and Scikit-learn. And, most of all, we appreciate the judges, advisors, and everyone involved with the Supercomputing Challenge for your diligent work in order to teach us skills that we will need later in life.

Team Members: Andy Corliss, Max Corliss, Phillip Ionkov, Ming Lo

Bibliography:

[1]: http://scikit-learn.org/stable/ - Website with many varied approaches to classification, regression, etc.

[2]: https://www.youtube.com/playlist?list=PLOU2XLYxmsIIuiBfYad6rFYQU_jL2ryal - Playlist for basic machine learning by Josh Gordon and the Google Developers youtube account

[3]: https://en.wikipedia.org/wiki/Iris_flower_data_set - Wikipedia article explaining the Iris flower dataset

[4]: https://en.wikipedia.org/wiki/MNIST_database - Wikipedia article about MNIST dataset

[5]: https://www.kaggle.com/c/tensorflow-speech-recognition-challenge - Website that hosts machine learning competitions.

[6]: http://playground.tensorflow.org - Website that allows the user to play with different types of neural networks.

[7]: https://librosa.github.io/librosa/ - Python package for music and audio analysis.