Instrument Classification

Sina Sabet, Cameron Long, Rafah Asadi, Alex Tareshawty

Problem Statement

Our goal with this project was to explore how well traditional classification techniques would work on sound data by creating models to classify instruments based on features derived from their sound. Aside from the basics of classification, we were interested in this subject because it provided for an interesting multi-class classification problem where the different classes aren't necessarily disjoint. For example, flute and clarinet are both woodwind instruments, so presumably there would be some overlap between the features of the two that a trumpet may not share. Another interesting facet of this was figuring out how to featurize the sound files we were working with. See the Methods section for more on this.

Data Corpus

The data we worked with was from the University of Iowa Musical Instrument Samples[1]. We worked with a variety of instruments; there were a total of 19 different instruments, which could be separated into three main categories: brass, woodwind, and string. The data for each instrument was split into about 40 .aif files, where each file held a short clip of sound, between 2 and 5 seconds. All main pitches (A, B, C, D, E, F, G), along with some flat pitches are available for each instrument. Various octaves for each pitch is also included.

In our application, we represented each of our music files as dictionaries of this shape:

```
{   "instrument":           string,
    "instrument_category":  string,
    "signal":               [int],
    "samplerate":           int,
    "samplewidth":          int,
    "nframes":              int,
    "sigstr":               string,
    "path":                 string   }
```

The important attributes are the "instrument", "instrument_category", "signal", and "samplerate". The "instrument" and "instrument_category" values were what we used to label our data, and the "signal" and "samplerate" values were used to featurize our data. For our specific case, "sigstr" is what we directly acquired from the data file, which is a hex-string representation of the

signal. From there we had to do some preprocessing to convert this to a numpy integer array to be used by our featurizers. After shuffling our data, we split it 80-20 into train and test sets.

Methods

The code for this project was written in Python 2. We heavily leveraged the scikit-learn library[2] for builtin models and labeling our data. Specifically, the LabelEncoder class was used to encode our categorical labels from a string representation like ["flute", "trumpet"] to a more model-friendly integer representation like [0, 1].

For featurizing our data, we leveraged a library called python_speech_features[3] to convert signal and sample rate values for each song into corresponding features using a process called the mel-frequency cepstrum transform. The library provided us with a function, mfcc, that would return to us an array of the feature coefficients. The function used analyzed 0.25s intervals of each signal with steps of 0.1s between each interval, and returned a 2-dimensional array with as many rows as there were intervals analyzed, and 13 columns corresponding to the mel-frequency cepstrum coefficients. As a consequence of this, signals of different length (ie. audio files of different lengths) resulted in feature arrays that were of different shapes. For each interval there were 13 coefficients, but there were a different number of intervals for each datapoint; longer sounds would have more feature rows than shorter ones. Since our classifiers were sensitive to the shape of our features, we needed to normalize our features so that each data-point's had features of the same shape. We experimented with summing and averaging across the rows of each data-point's features, and we settled on averaging due to it having superior performance.

Our experiment took two approaches for classifying:

1. Classifying input into Instrument: the input for each run is the features of a set of instruments chosen at random with sizes ranging from 2 to 10. The class label is the name of the instrument . Each particular classifier is run for 5 times using different inputs. The accuracy of the classifier is then calculated as the average of the accuracy of the five runs. RandomForest and Neural Nets do not generate the same accuracy results when trained on the same data. The accuracy output from each has have high variation. To overcome this issue,these classifiers are run 25 times, then the accuracy is calculated from 25 runs instead of 5.

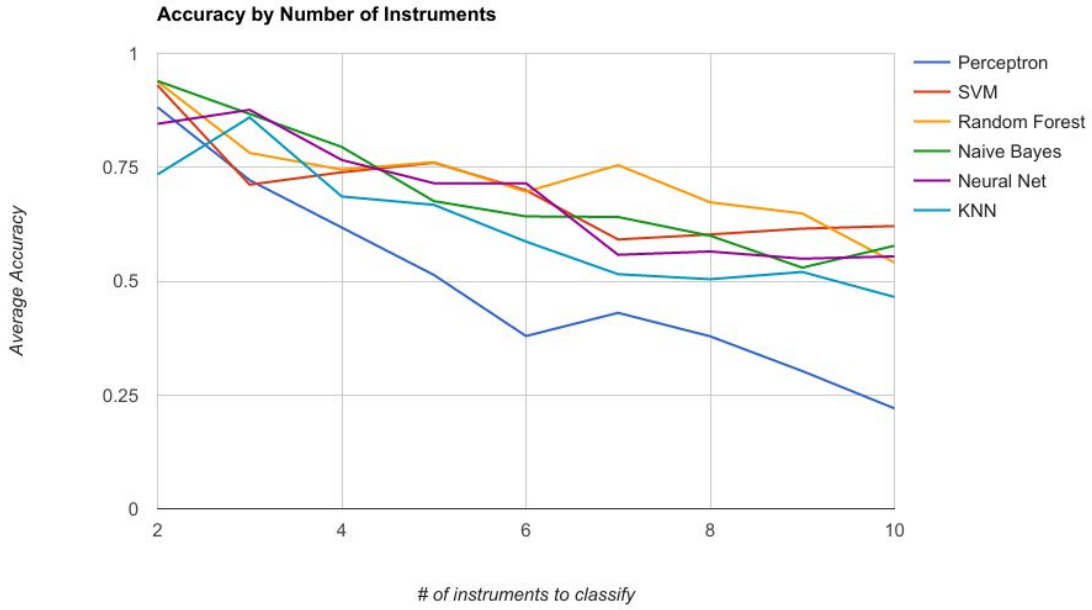
2. Classifying input into family of instruments: in this approach, the entire data set of 19 instruments was divided into one of three classes: woodwind, brass, or string instruments. The classifiers were then handed this instrument family class and trained on the larger dataset, each classifier was run for 5 runs and then the results were averaged for the final result. As random forest and Neural Nets do not generate the same accuracy results when trained on the same data, each run for these two classifiers consisted of 5 runs averaged together.

Our code is available at https://github.com/sinasabet81/ML_Project

Results

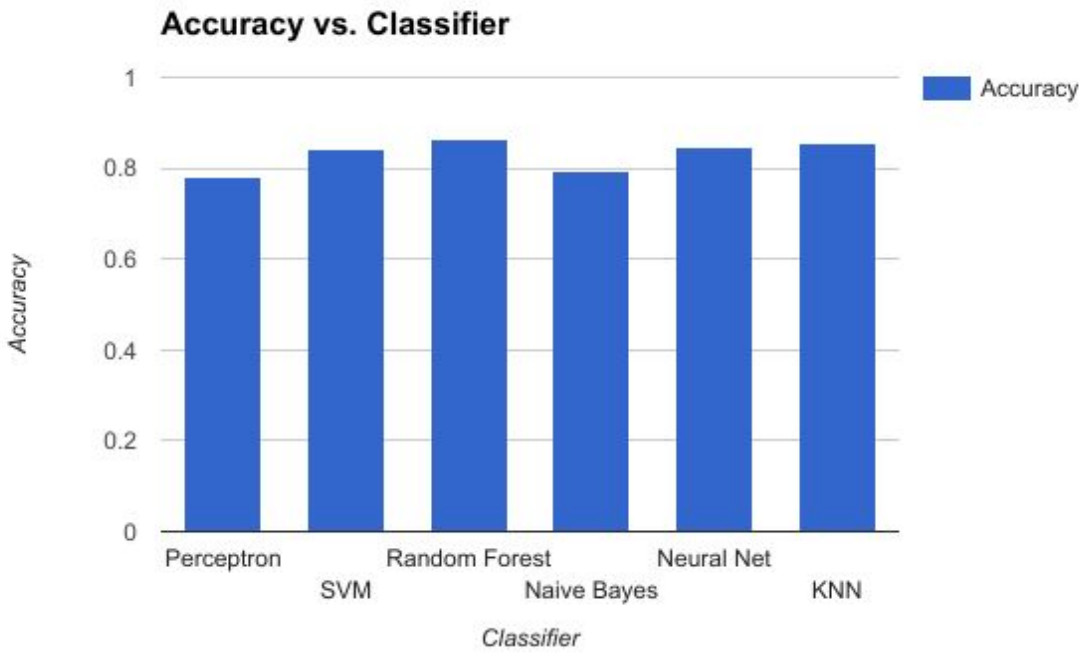
Classifying Individual Instrument (flute, oboe, trumpet, etc.)

	2	3	4	5	6	7	8	9	10
Percep	0.8819	0.7216	0.6177	0.5135	0.3793	0.4303	0.3787	0.30234	0.2204
SVM	0.93	0.7118	0.7391	0.7600	0.6996	0.5914	0.6025	0.6153	0.6209
R. Forr	0.9383	0.7817	0.7449	0.7611	0.6965	0.7547	0.6731	0.6486	0.5410
Bayes	0.9396	0.8672	0.7946	0.6757	0.6421	0.6408	0.5996	0.5295	0.5777
Net	0.8455	0.8762	0.7661	0.7150	0.7151	0.5578	0.5650	0.5492	0.5543
KNN	0.7343	0.8596	0.6858	0.6675	0.5868	0.5152	0.5041	0.5201	0.4651



Classifying Instrument Category (woodwind, brass, strings)

Classifier	Average Accuracy
Perceptron	0.78
SVM	0.84
Random Forest	0.8618461538
Naive Bayes	0.7938461538
Neural Net	0.8473846154
KNN	0.8538461538



The results of both classification approaches show that Perceptron classifier has the lowest performance. RandomForest has the best performance in both approaches. KNN had much better performance in second approach because parameter K=3, which is the true number of families of instrument.

Conclusions

The various classifiers we used performed respectably for our task. Our random forest performed best for classifying instruments by category at 86%. All of our models did well at classifying between two instruments, with most performing near 90% except for the KNN which was closer to 70. All of the models' performance decreased as the number of instruments went up, with the perceptron dropping the most.

To build upon this project, it'd be interesting to experiment with a wider array of featurizers. We simply worked with averaging the MFC coefficients, but perhaps combining these averages with another set of features, or even using a different set of features altogether would provide for some interesting results. For this purpose, we'd need a more in-depth understanding of signal processing than we possessed.

It'd also be interesting to work with a more complex nonlinear classification method, such as a recurrent neural network. A model like this may be better at working with featuresets that aren't of uniform shape, resulting in us doing a better job of classifying audio files that are further apart in length instead of having to do some normalization on the features.

Beyond that, it'd be interesting to see how a project with a larger scope would do with less "clean" data. Perhaps experimenting with sound files entailing multiple instruments could be interesting. If this is successful, you could go all the way to having a model that can identify which instruments are playing which notes at what times during a song. This type of application was a bit out of our scope, and the linear classifiers we used would likely lack acceptable performance, but with a more robust model and more intricate featurizers, perhaps something like this would be feasible.

References

1. http://theremin.music.uiowa.edu/MIS.html
2. https://github.com/scikit-learn/scikit-learn
3. https://github.com/jameslyons/python_speech_features