**ELEC 301**

**Final Project Report**

**December 18, 2019**

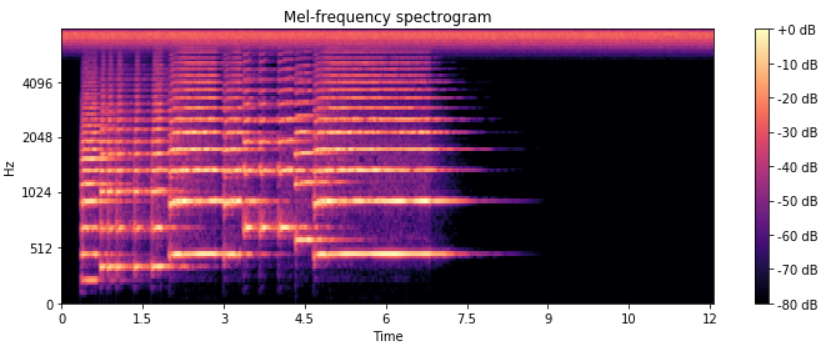
The Richest B’s:

Zach Alvear (zja1), Joshua Bae (jsb9), Rene Carballo (rdc2), Anirudh Kuchibhatla (ask9)

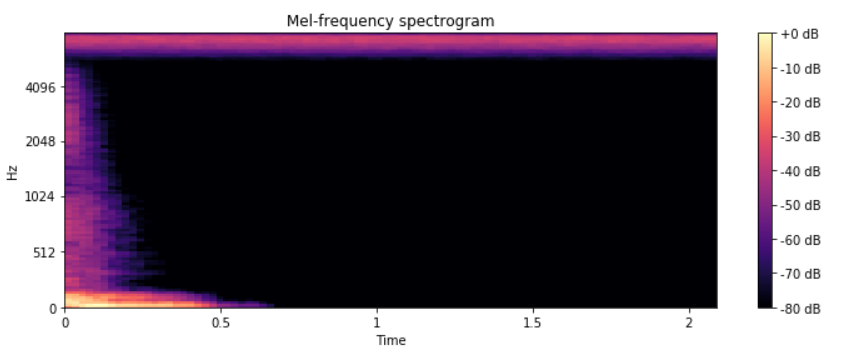
**Methods Used**

**Exploration**

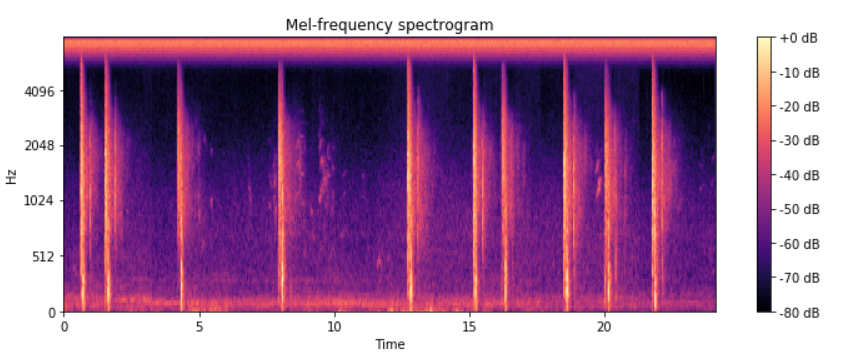
The very first thing the team did upon starting the project was listening to the different sound files that were provided for the competition. In all of the sound files the team listened to, the team heard a high pitched whistling noise that did not seem to be related to the expected sound that the file was supposed to be playing. In order to confirm that this was an unwanted part of the signal, the team plotted the spectrograms of 4 different instruments. These spectrograms are shown below:

****

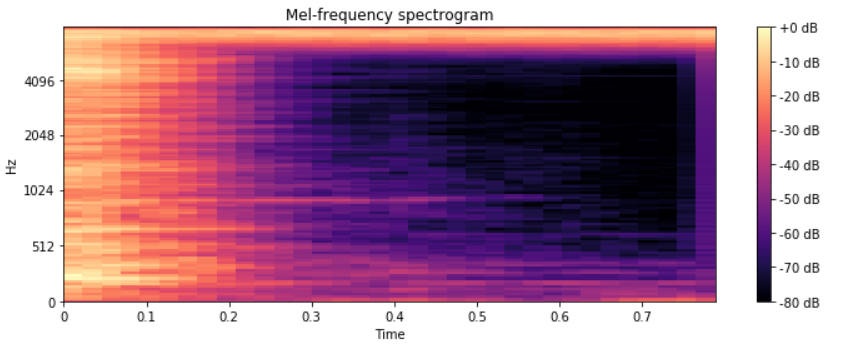
**Figure 1: Trumpet Spectrogram**

****

**Figure 2: Bass Drum Spectrogram**

****

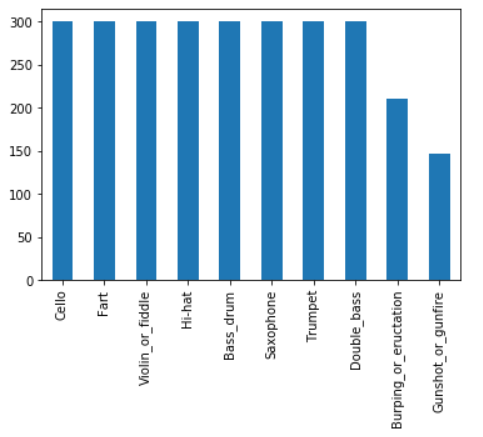
**Figure 3: Gunshot Spectrogram**

****

**Figure 4: Hi-Hat Spectrogram**

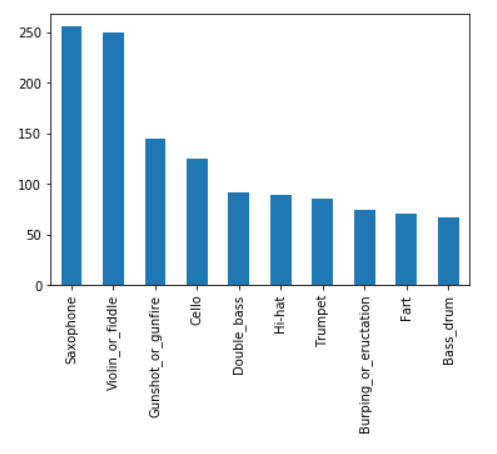
In all of these spectrograms, the team consistently noticed constant power across time above approximately 6000 Hz and presumed that this was white noise that was unrelated to the actual instrument sound since all of the sounds here come from harmonically different instruments. Thus the team knew that the team had to filter this common white noise out of all of the data, which is further discussed in the **Preprocessing Section** of this report.

After listening to the data, the team analyzed the distribution of the different instruments to ensure the team had a fair representation of each of them. The team plotted each instrument by the number of times it appeared in the test data, which is shown below:



**Figure 5: Full Training Set Counts**

As can be seen from the plot above, it appears that most of the instruments have a fair representation amongst the data set. However, this considers the entire training data set. One important point that the team needed to consider was that the training set was not entirely manually verified. More than half of the data set was verified by a machine algorithm, which was estimated to only have 65-70% accuracy. This meant that the team could not completely trust the labels of the machine verified labels. The team then decided to see the distribution of the manually verified data in case the team decided to use only the data the team was sure was correct and plotted the results below:

****

**Figure 6: Verified Training Set Counts**

This was a much more uneven distribution compared to before, which was somewhat worrying as using this subset of data would give the team less accurate results. However, each instrument still had more than 50 verified data points. The team decided to use both datasets in the model and test them to see which one gave us more accurate results.

In order to better understand the data, the team searched for different Python libraries that the team could use for audio analysis. Two packages we made use of were librosa and scikit-learn (sklearn).

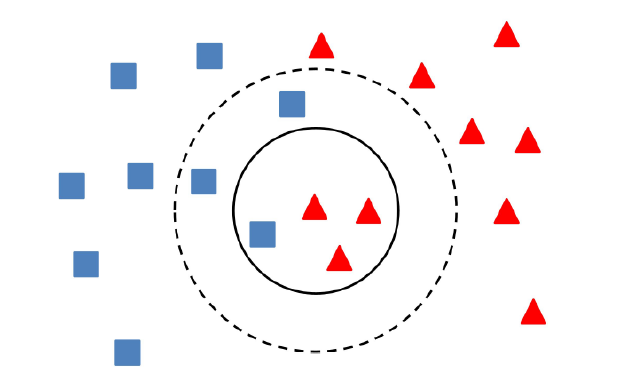
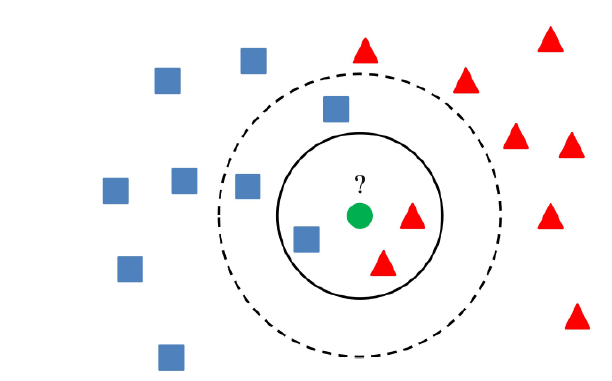
Librosa is a python package for music and audio analysis; since the project dealt with classifying audio files, it became a perfect starting point for analysis. Considering the fact that the data provided was in the form of wavelet files, using librosa made the most sense after a quick online search. It did a fantastic job of converting the sampled, digitized sound waves to more manageable forms to work with. For displaying various spectrograms and models, feature extraction (frequency and time domains), and filtering purposes, librosa proved to be extremely effective. Its many submodules allowed us to extract very useful features from the dataset provided. The features we deemed pertinent to training our algorithms will be further discussed in the **Feature Selection** section.

Sklearn, with its powerful machine learning capabilities, allowed the team to experiment with various models to arrive at the best one for predictions. It provided a simple and efficient set of tools to conduct data analysis and test our intuitions. The implementation was quick and painless as much of the analysis could be done with the call of a function.

Finally, as part of the deep learning process, the team also used Tensorflow’s Keras API. This library allowed for the development of high-level neural networks which could be run on top of Tensorflow. Its core data structure is a model, which is a process for organizing layers.

**KNN Classifier**

K-Nearest Neighbors (KNN) was one of the classifiers mentioned in class. The algorithm works by comparing the “distances” (usually the 2-norm) between a given test point and the rest of the training data, taking the K - shortest distance points, picking the most common label among those points, and assigning that label to the test point. Some positives of KNN are that it does not need to be “trained” and has a low error rate. Some downsides are that it does not work well at higher dimensions of feature vectors and has a higher computational complexity than some other algorithms.



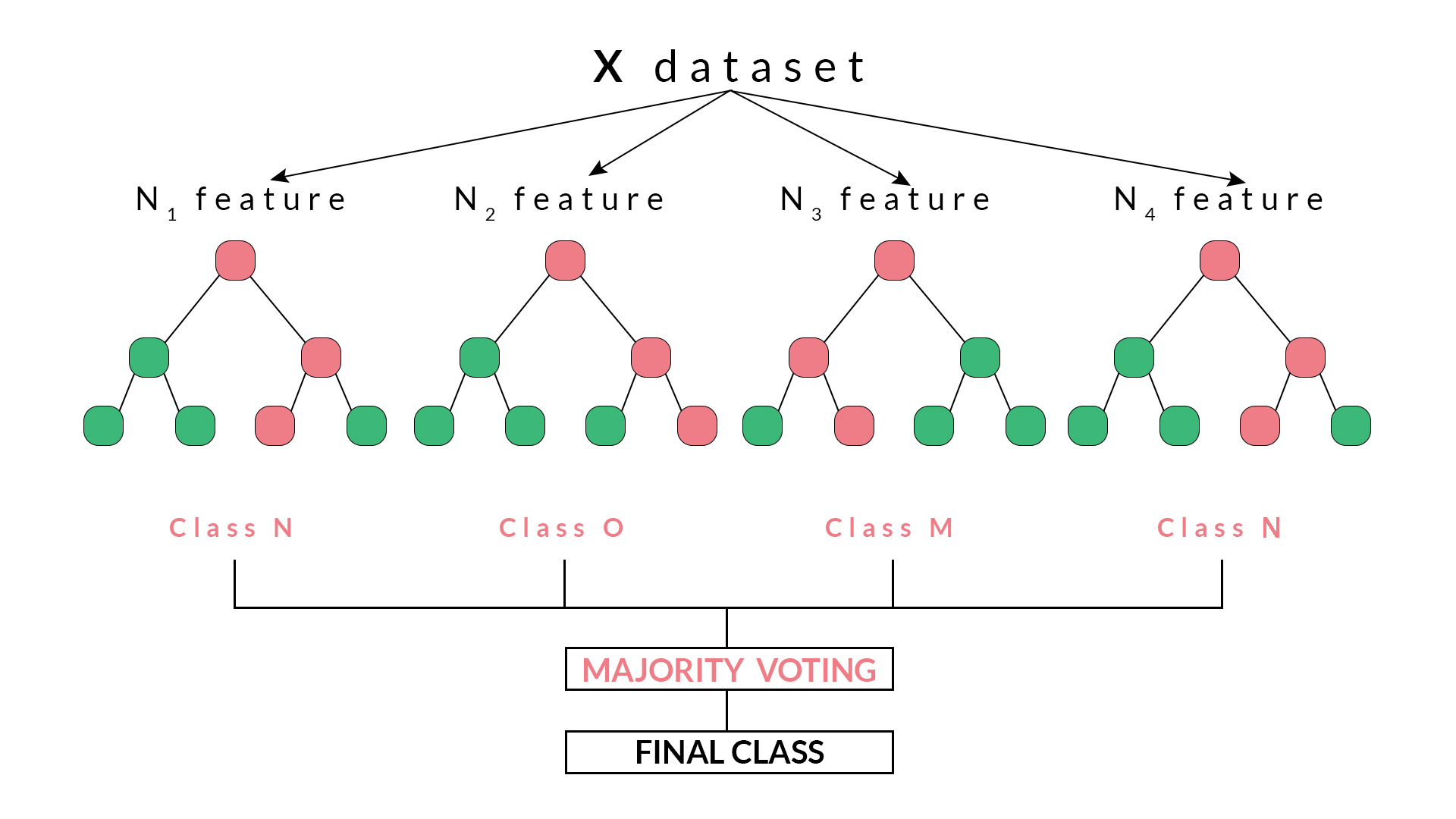
**Figure 7: Illustration of the KNN Algorithm**

Source: Class Slides

**Random Forests**

In the preliminary investigation of data science algorithms, the team learned about Random Forests Classifier. The fundamental idea behind the algorithm is that a “forest” of decision trees based on each feature is used to create the model for classifications. The model creates a classification from each decision tree and uses majority voting to decide on the final classification for the data. Although all of the decision trees are uncorrelated in terms of outcome, the forest creates a fairly accurate model due to the balance of random guessing and user determined weights, such as the number of trees in the forest.

Below is a graphical representation of the random forests classifier:



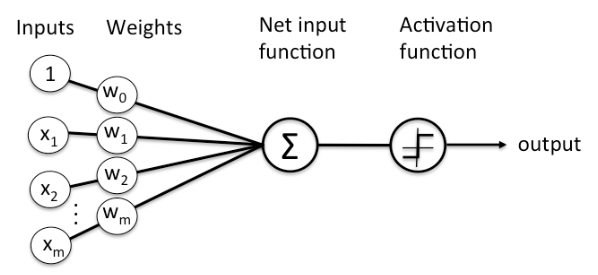
**Figure 8: Random Forest Classifier**

Source: <https://blog.quantinsti.com/random-forest-algorithm-in-python/>

**Neural Networks**

Deep learning consists of networks composed of several layers. Nodes make up these layers, take in input data with a set of coefficients (weights), and dampen or amplify that input by these values. Next, these input-weight products are summed and passed through a node’s activation function. The activation function determines how much a node’s output should progress through the network to affect final predictions.

Below is a diagram of what one node might look like:



**Figure 9: Neural Network Structure**

Source: <https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/>

A cluster of these nodes makes up what is commonly known as a node layer. With multiple layers in the model, a starting layer’s output becomes the input to the subsequent layer. Typically, there exist an input layer, hidden layers, and an output layer (multilayer perceptron configuration). Depending on if the problem at hand is linearly separable, one might want to incorporate more than one hidden layer to arrive at the best possible results. As for deciding on how many layers or nodes to use in one’s algorithm, it is often encouraged to experiment or use intuition to find the best solution. In general, it is very difficult to analytically calculate the exact number for each to address a specific real-world predictive modeling problem. Using similar past examples that have worked well is another method of constructing the best neural network.

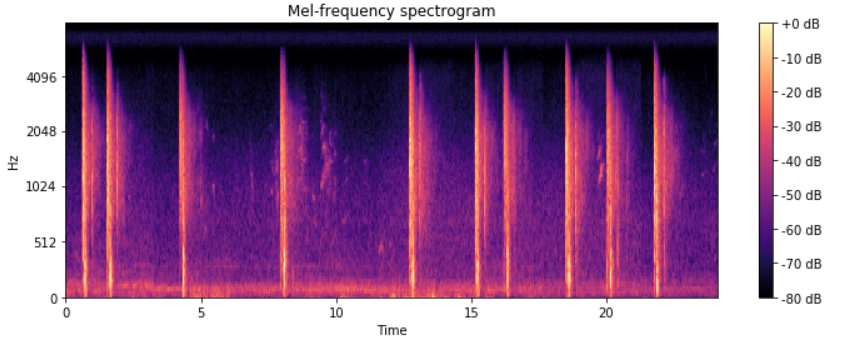
Batch size and epoch are two more hyperparameters to explore as part of this in-depth analysis of neural networks. Batch size defines the number of samples that will be pushed through the network. Batch gradient descent will learn by processing the entire dataset of features into the learning process at once. However, there are two other methods that can make the training method much more effective: mini-batch (batch size between 1 and size of the training set) and stochastic (batch size = 1) gradient descent. Experimenting between different batch sizes is an essential part of the algorithm development process.

Epochs, on the other hand, define the number of times that the learning algorithm will work through the entire training dataset. A single epoch indicates that each sample in the input data has had one chance to update the internal model parameters. Traditionally, the number of epochs for a given neural network will be relatively large to allow the learning algorithm to run until the error is optimally minimized. Its value determines how important the training model treats its input, so it must be carefully chosen.

**Data Science Process**

**Preprocessing**

As noted in the previous section, after listening to some of the sound files the team realized that most of them sounded very noisy and full of static. This could hamper the algorithms to correctly establish the proper weighting of features. By analyzing the spectrograms of most of the audio files, the team determined that the noise was mostly constrained to a frequency above 6000Hz. Thus, the team decided to first preprocess the data by Butterworth and low pass filtering the data. By sending the data through a low-pass 4th order Butterworth filter, relevant information below 6000Hz was retained while anything above the defined limit was eliminated.This minimized data distortion and allowed the models to be trained on more accurate features. The use of this filter is common in audio analysis. The result of the Butterworth Filter on the Gunshot can be seen in the figure below. When compared to the Gunshot spectrogram in Figure 3, it can be observed that the line at the top is no longer present. This indicates that the noise is no longer present and when listening to the filtered audio, we indeed did not hear the high-pitched noise.



**Figure 10: Filtered Gunshot Spectrogram**

Since a significant portion of the audio files was classified by non-humans (thereby complicating the training process with inaccurate data), the models that the team trained used both the dataset including and excluding these rows of data. During the testing process, the team could then determine which model yielded the most accurate predictions.

**Model Selection**

The team chose KNN as the in-class model, as it had a theoretically high performance, and because it would work well with the relatively small number of features the team was using. Furthermore, some of the data analyzed seemed to show promise for kNN when visualized. It was also a simple model that could quickly be implemented and thus would be useful for testing the significance of different features.

The team also chose to use the Random Forests Classifier, which was quite different from the other models that were chosen. After all, how is it that the sum of many random decision trees could be able to classify data accurately? After preliminary research, the team found that this algorithm had a reputation for being able to classify data with more accuracy than the other models the team had learned in class. Another reason that this classifier sounded interesting is that it would give random priority to each feature, which could prove to be helpful in identifying complex audio signals.

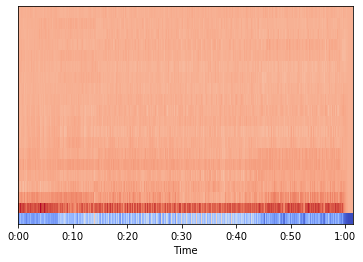
For the final model, the team decided on using a neural network model for classification. This method resulted in creating the most accurate results out of three methods the team mentioned in this report. Deep learning (stacked neural networks) has been praised for its incredible accuracy when it comes to classification, so the team made sure to take advantage of that for this project. However, that is not to say that there are no downsides to neural networks that would hinder the team’s implementation of it. While neural networks are extremely useful for this type of task, it suffers from needing large quantities of data to train a decent model. Thankfully, this was not a major issue since there was so much data to be extracted from each wavelet file corresponding to every instrument.

**Feature Selection**

After some research online dealing with audio classification, the team decided that most of our features would come from spectral analysis. Compared to waveforms in the time domain, information from spectrograms in the frequency domain seemed to provide much more pertinent information for classification. Thankfully, the libraries that the team used (ex. librosa) contained a wealth of functions that allowed for such feature extraction.

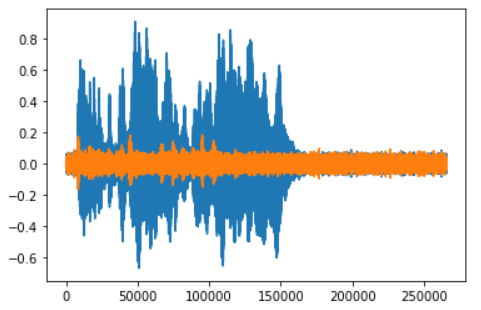
Some useful features the team thought would help with the classification process were chromograms, spectral centroids, spectral bandwidth, spectral rolloff, and zero crossing rate. The chrograms were computed from a waveform. The spectral centroid was computed by extracting the mean from each frame of a signal from a magnitude spectrogram after normalization. The spectral bandwidth of the p’th order was also computed for each frame of a signal. The spectral rolloff found the center frequency for a spectrogram bin such that at least 85% of the energy of the spectrum in a certain frame is contained in this bin and the bins below. Finally, the zero crossing rate how often the signal in question crosses the time axis.

One feature that the team decided to explore was Mel Frequency Cepstral Coefficients (MFCC). This feature ended up being one of the most influential features in the models we developed. These coefficients make up what is known as the Mel-frequency cepstrum. Simply put, the cepstrum is the information about the rate of change in spectral bands. It is defined as the spectrum of the log of the spectrum of the time signal (taking the Fourier transform of the warped logarithmic spectrum). These coefficient values are based on the Mel scale which relates the perceived frequency of a tone to the actual measured frequency. The scale adjusts the original frequency to match the frequencies that a human ear can hear. Each of these coefficients was able to provide, in a way, an image representation of each audio sample. We encapsulated each of the 20 sets of coefficients with the set’s mean and standard deviation to feed into our models, otherwise, there would be too many features that may lead to overfitting.

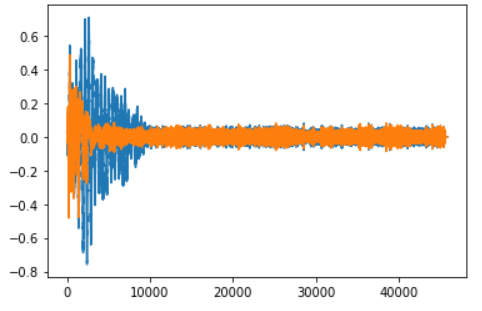


**Figure 11: MFCC plotted against time**

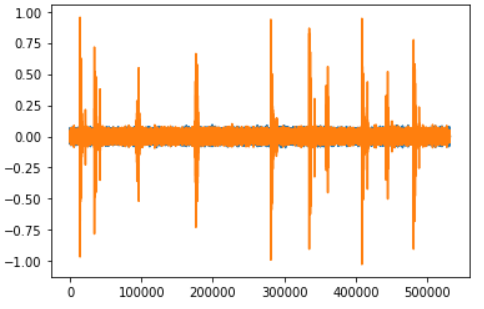
Another feature the team decided to explore was the harmonic and percussive components of the signal. The harmonic component of a signal corresponds to the melody and pitch of the music, whereas the percussive component corresponds more to the beat and the rhythm of the music. The team decided to analyze these features a bit more as the team believed that they could be used to differentiate melodic instruments like the saxophone and double bass from percussion/non-instruments like the gunshot and hi-hat. As a first step, the team plotted these components of different instruments to identify any patterns. These plots can be seen below (the blue signal corresponds to the harmonic component whereas the orange signal corresponds to the percussive component):

****

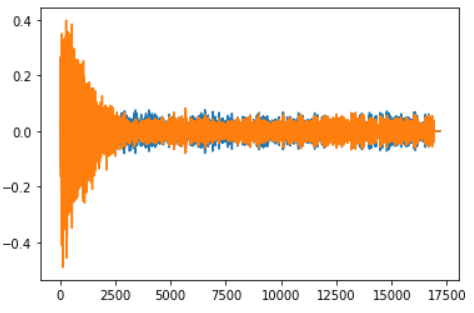
**Figure 12: Trumpet Harmonic and Percussive Strength over Time**



**Figure 13: Double Bass Harmonic and Percussive Strength over Time**

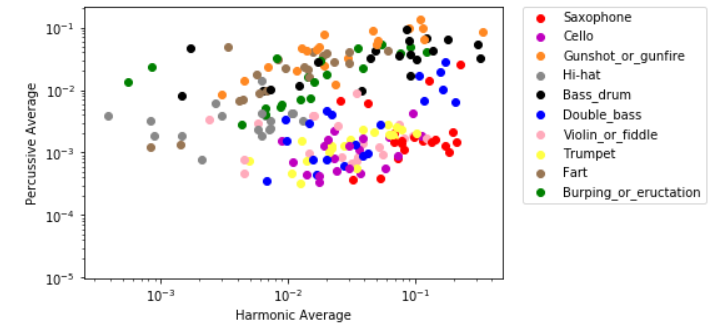


**Figure 14: Gunshot Harmonic and Percussive Strength over Time**



**Figure 15: Hi-Hat Harmonic and Percussive Strength over Time**

As can be seen from these plots, it can be seen that the trumpet and double bass had larger harmonic signals whereas the gunshot and the hi-hat had higher percussive signals. This matched the initial expectation, and to see whether there were overall trends in the relation between harmonic and percussive components for different instruments, the team took the average of both components of 20 different signals of each of the instrument types and plotted the result. The team did this to identify distinctive regions for different instruments to see if it would be possible to use the k-Nearest Neighbors Algorithm with great accuracy to identify different instruments. Even if this did not result in distinctive regions, the team could still find significant trends for different instruments, which would contribute significantly to classification and could be inputted into other models along with other distinctive features. The plot the team made is shown below (this is shown on a logarithmic scale):



**Figure 16: Harmonic vs Percussive Average plotted for 10 Instruments**

It appeared that most instruments had defined regions within this graph, however, these regions greatly overlap, suggesting that while this is an important feature, it is not enough to distinguish it from other instruments. Overall, this plot was very promising and indicated adding a few more significant features could yield a high accuracy. The graph also matched the initial expectations and the results from each of the 4 harmonic vs. percussive plots above. String/woodwind/brass instruments had lower percussive components and higher percussive components, while percussion instruments and non-instruments had the opposite.

**Parameter Tuning**

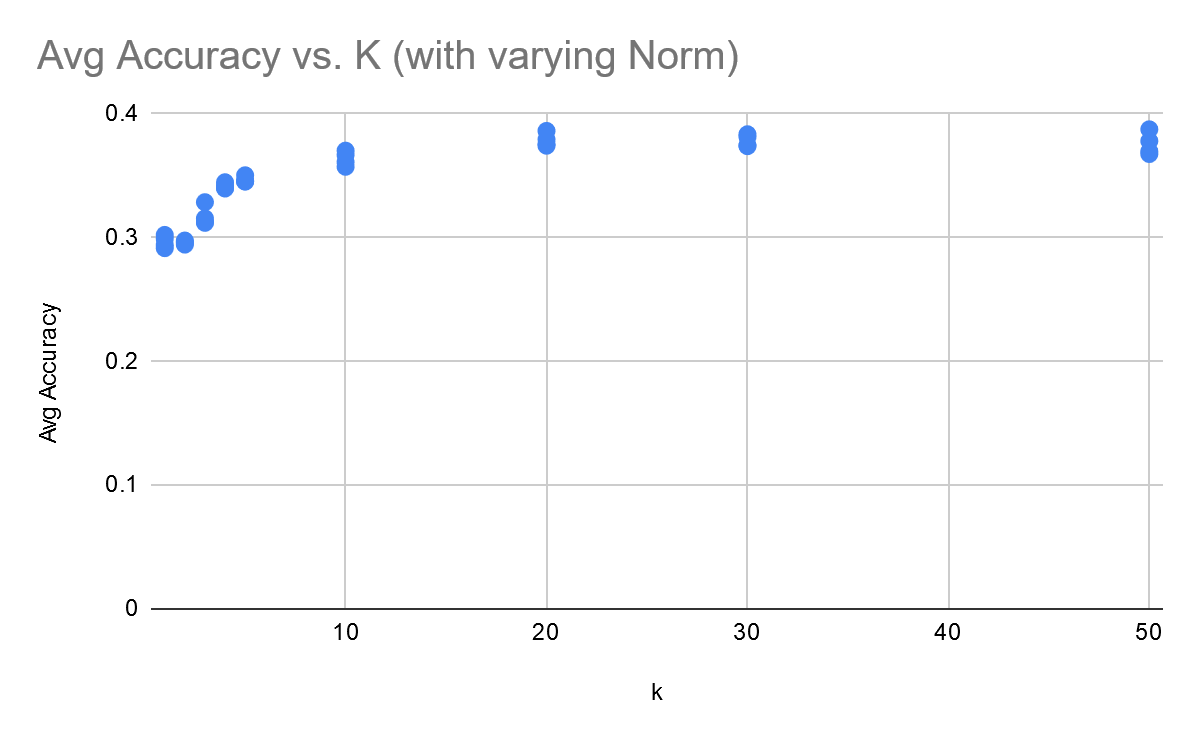
**KNN**

With the concern of high dimensional data in mind, the team started KNN using only the percussive and harmonic averages of the data, to see how well the model would perform. The team also used a custom-built implementation of KNN, so as to better learn how the algorithm worked, and better analyze the given data. The results below are for various norms and values of k, with an average accuracy calculated over 10 random test/train splits of 20/80 percent respectively (100 percent of the data being used):

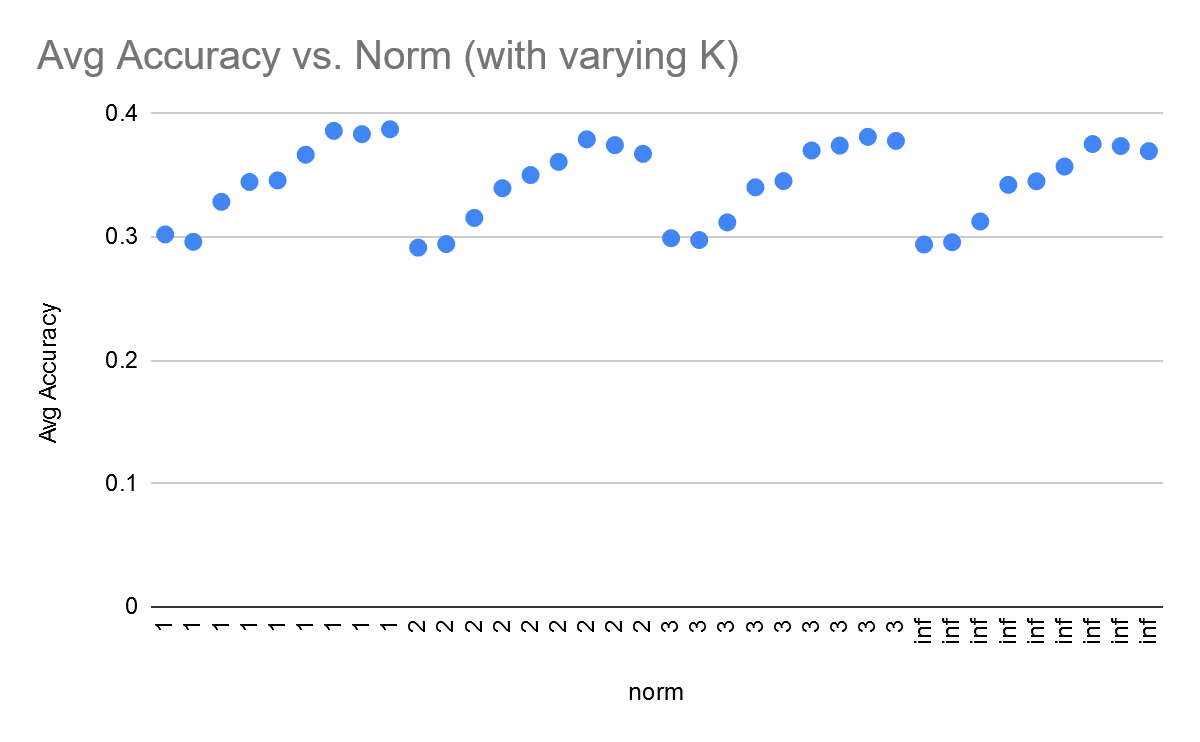
**Percussive and Harmonic Average Features Only**

|  |  |  |
| --- | --- | --- |
| **K** | **Norm** | **Average Accuracy** |
| 1 | 1 | 0.302173913 |
| 2 | 1 | 0.2961956522 |
| 3 | 1 | 0.3286231884 |
| 4 | 1 | 0.3447463768 |
| 5 | 1 | 0.3460144928 |
| 10 | 1 | 0.3668478261 |
| 20 | 1 | 0.3862318841 |
| 30 | 1 | 0.3835144928 |
| 50 | 1 | 0.3875 |
| 1 | 2 | 0.2914855072 |
| 2 | 2 | 0.294384058 |
| 3 | 2 | 0.3155797101 |
| 4 | 2 | 0.339673913 |
| 5 | 2 | 0.3503623188 |
| 10 | 2 | 0.3610507246 |
| 20 | 2 | 0.3793478261 |
| 30 | 2 | 0.3746376812 |
| 50 | 2 | 0.3675724638 |
| 1 | 3 | 0.2990942029 |
| 2 | 3 | 0.2976449275 |
| 3 | 3 | 0.3119565217 |
| 4 | 3 | 0.3403985507 |
| 5 | 3 | 0.3454710145 |
| 10 | 3 | 0.3702898551 |
| 20 | 3 | 0.3742753623 |
| 30 | 3 | 0.3813405797 |
| 50 | 3 | 0.3780797101 |
| 1 | inf | 0.2940217391 |
| 2 | inf | 0.2958333333 |
| 3 | inf | 0.3126811594 |
| 4 | inf | 0.3423913043 |
| 5 | inf | 0.3452898551 |
| 10 | inf | 0.3572463768 |
| 20 | inf | 0.3755434783 |
| 30 | inf | 0.3739130435 |
| 50 | inf | 0.3697463768 |

**Figure 14: Accuracy Scores for values of K and Norm**



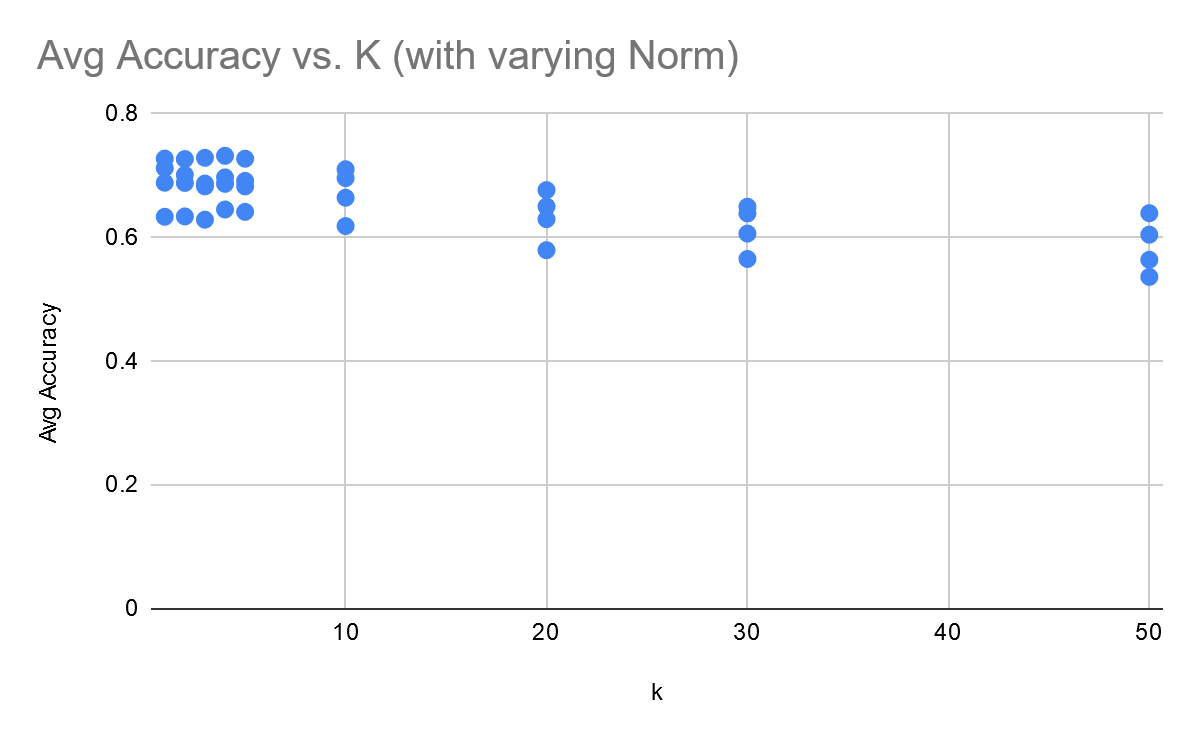
**Figure 17: Accuracy Scores versus k**

****

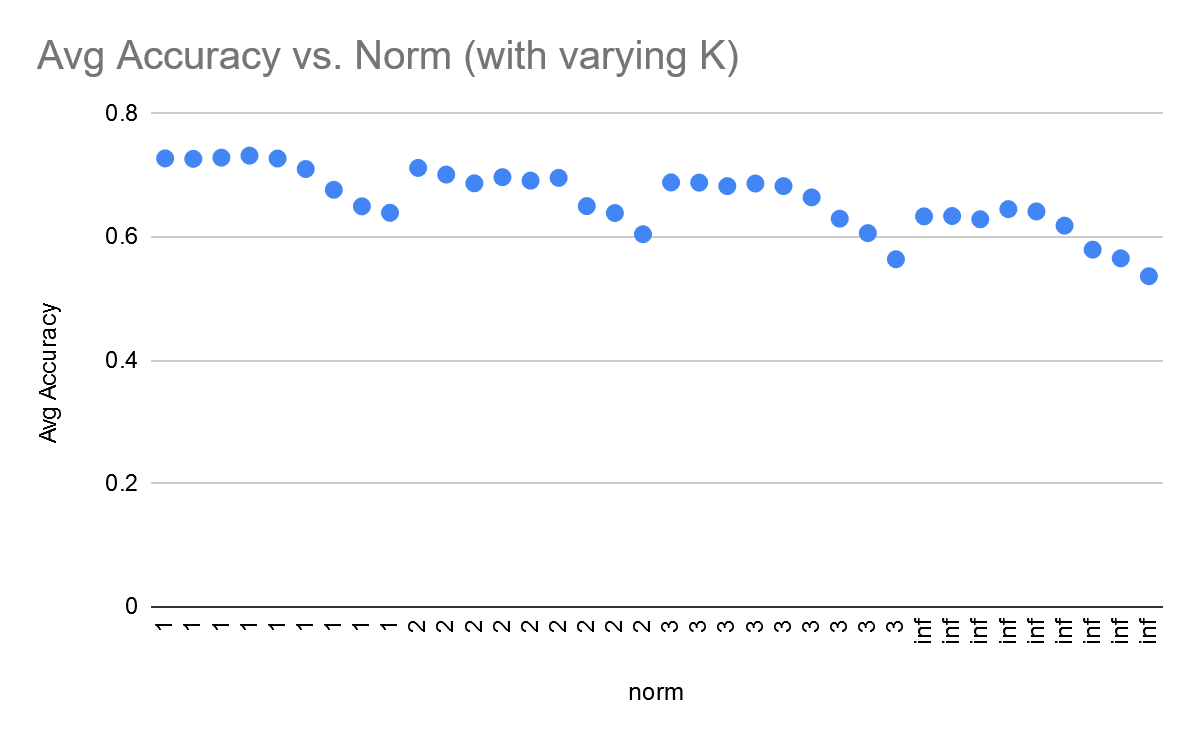
**Figure 18: Accuracy Scores versus Norm**

While the accuracy was not the best, two important trends were noticed: Firstly, the accuracy tended to increase with increasing k, but the rate of increase also decreased with increasing k, apparently reaching a limit. Secondly, the norm seemed to have little effect on the accuracy of the data, with only fractions of a percentage point between the different norms.

Given the percent accuracy at this point, the team decided to continue with more features. KNN was then run with 20 MFCC coefficients only, to see how it performed without the harmonic and percussive average features. The results below are for various norms and values of k, with an average accuracy calculated over 10 random test/train splits of 20/80 percent respectively (100 percent of the data being used):



**Figure 20: Accuracy Scores versus k for 20 MFCC Coefficients Only**

****

**Figure 21: Accuracy Scores versus Norm for 20 MFCC Coefficients Only**

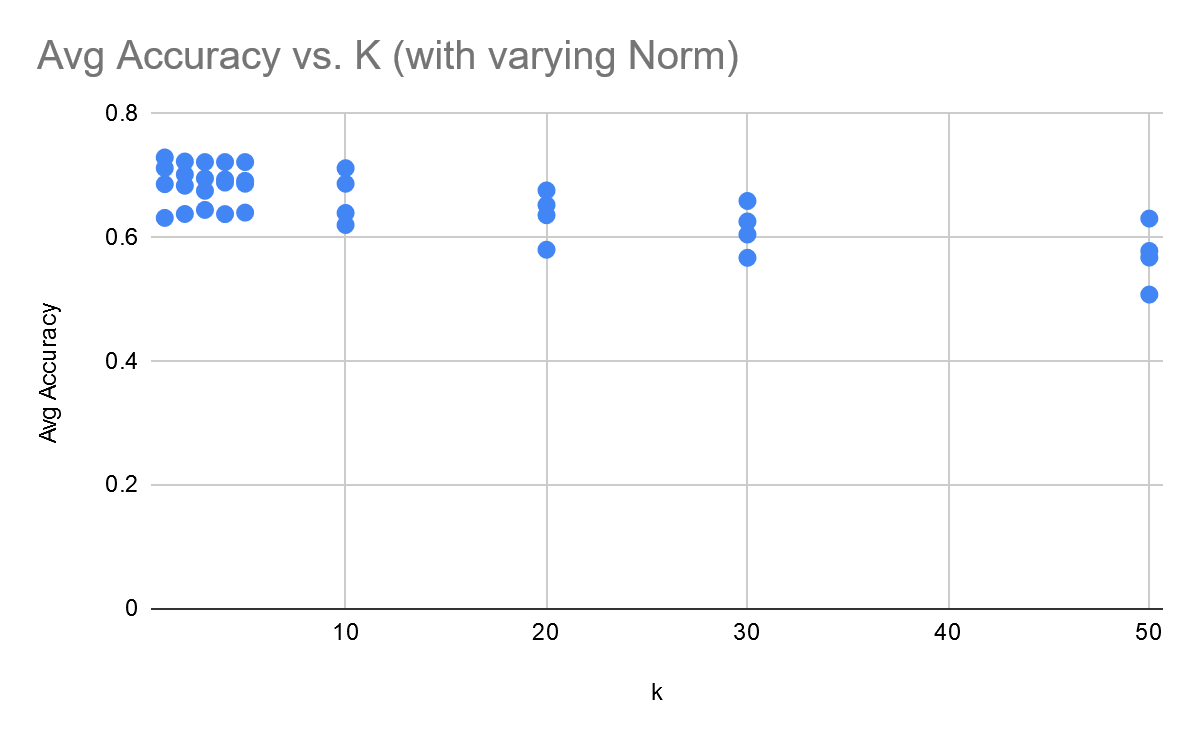
Interestingly enough, the trends noticed with just the percussive and harmonic averages were almost reversed when looking at the MFCC. The accuracy tends to decrease with increasing k, and the norm seems to have more of an impact on the data - smaller norm yields higher accuracy scores.

Next, MFCC and the percussive/harmonic average were combined. The results below are for various norms and values of k, with an average accuracy calculated over 10 random test/train splits of 20/80 percent respectively (100 percent of the data being used):

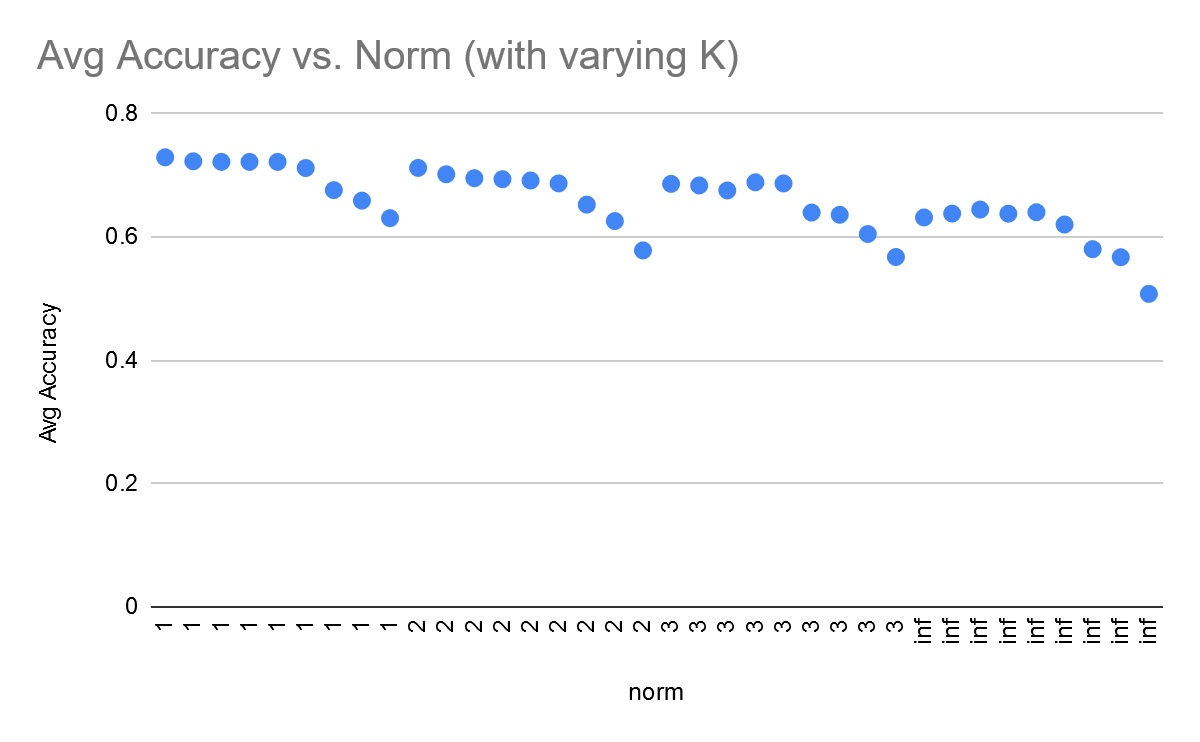
**20 MFCC Coefficients + Harmonic/Percussive Average**

|  |  |  |
| --- | --- | --- |
| **K** | **Norm** | **Average Accuracy** |
| 1 | 1 | 0.7295289855 |
| 2 | 1 | 0.7230072464 |
| 3 | 1 | 0.7221014493 |
| 4 | 1 | 0.7221014493 |
| 5 | 1 | 0.7221014493 |
| 10 | 1 | 0.7121376812 |
| 20 | 1 | 0.6762681159 |
| 30 | 1 | 0.6592391304 |
| 50 | 1 | 0.6307971014 |
| 1 | 2 | 0.7123188406 |
| 2 | 2 | 0.7019927536 |
| 3 | 2 | 0.6956521739 |
| 4 | 2 | 0.6940217391 |
| 5 | 2 | 0.6918478261 |
| 10 | 2 | 0.6871376812 |
| 20 | 2 | 0.6527173913 |
| 30 | 2 | 0.6260869565 |
| 50 | 2 | 0.578442029 |
| 1 | 3 | 0.6864130435 |
| 2 | 3 | 0.6838768116 |
| 3 | 3 | 0.6757246377 |
| 4 | 3 | 0.6889492754 |
| 5 | 3 | 0.6871376812 |
| 10 | 3 | 0.6398550725 |
| 20 | 3 | 0.6362318841 |
| 30 | 3 | 0.6050724638 |
| 50 | 3 | 0.5677536232 |
| 1 | inf | 0.631884058 |
| 2 | inf | 0.6382246377 |
| 3 | inf | 0.6447463768 |
| 4 | inf | 0.6380434783 |
| 5 | inf | 0.6403985507 |
| 10 | inf | 0.6204710145 |
| 20 | inf | 0.5804347826 |
| 30 | inf | 0.5673913043 |
| 50 | inf | 0.5079710145 |

**Figure 20: Accuracy Scores for values of K and Norm for 20 MFCC Coefficients + Harmonic/Percussive Average**



**Figure 22: Accuracy Scores versus k for 20 MFCC Coefficients + Harmonic/Percussive Average**

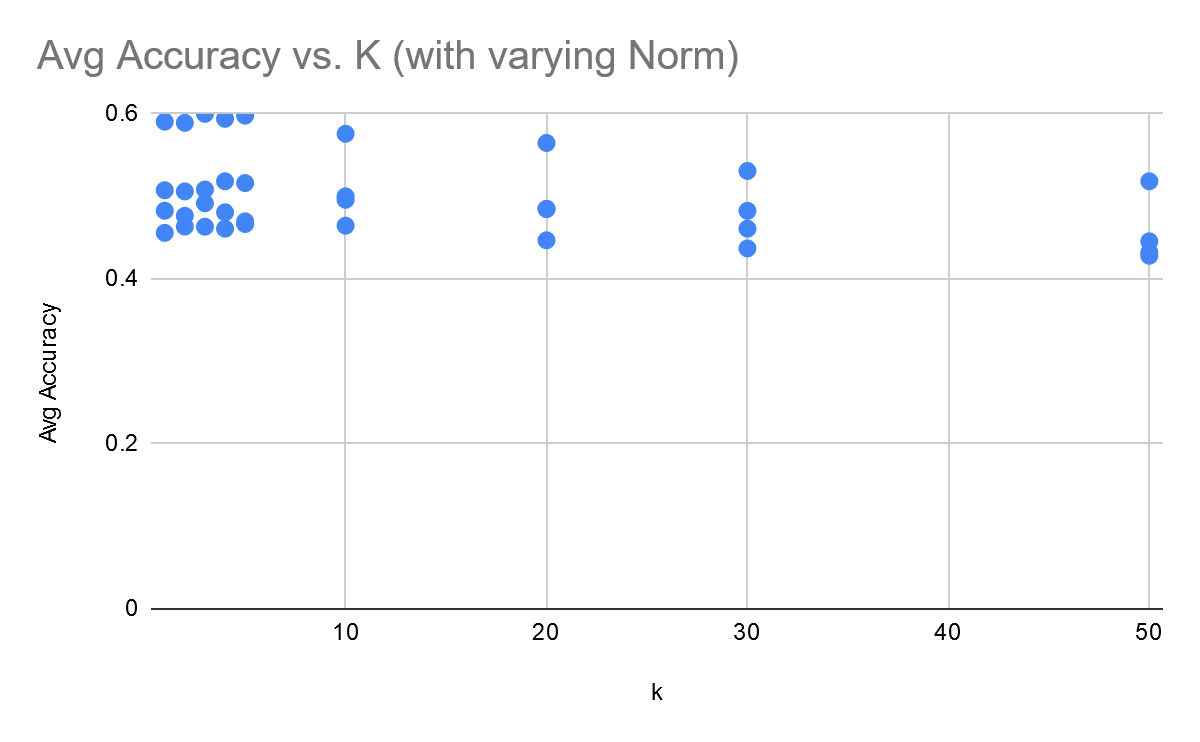
****

**Figure 23: Accuracy Scores versus Norm for 20 MFCC Coefficients + Harmonic/Percussive Average**

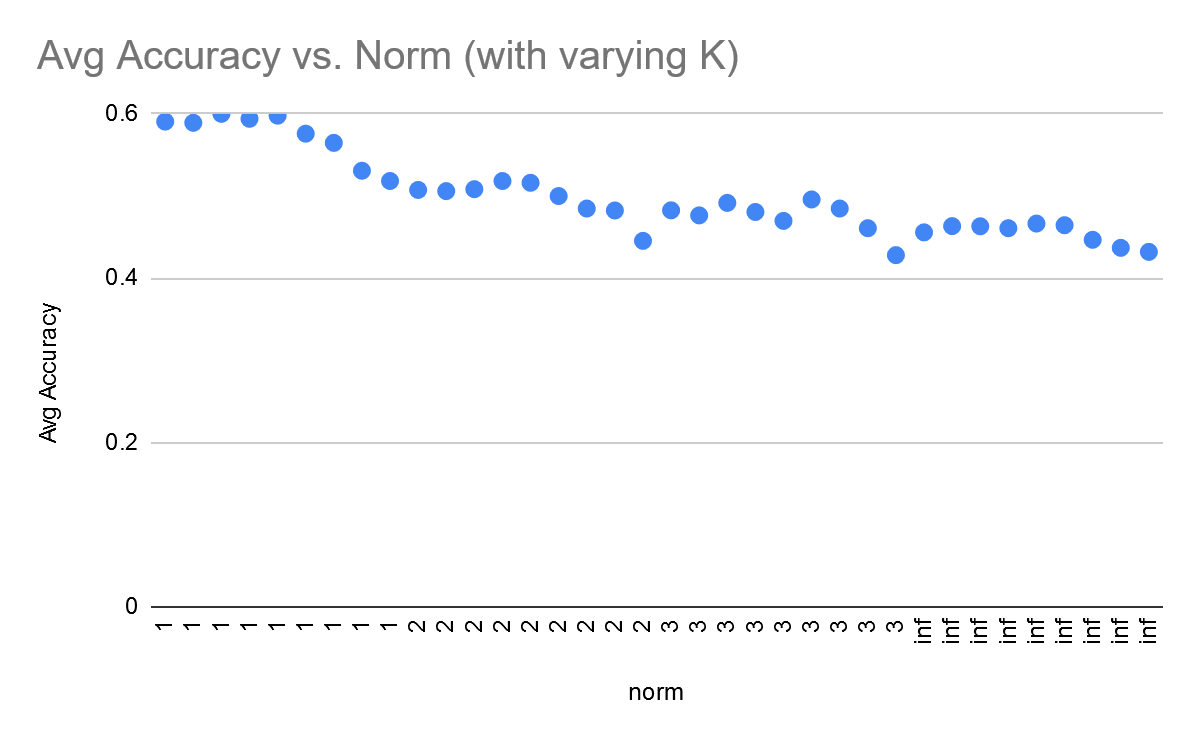
The trend is very similar to just the MFCC coefficients, with decreasing accuracy for higher k and higher norm, with the highest value being .2 percent higher than just the MFCC. This might imply that the percussive and harmonic averages do not add much in terms of better separating the data, at least when compared to the MFCC coefficients.

After more research, the team decided to run KNN with all the features that had been found: MFCC, harmonic/percussive averages, and others such as chroma stft. The results below are for various norms and values of k, with an average accuracy calculated over 10 random test/train splits of 20/80 percent respectively (100 percent of the data being used):

**All Discovered Features**



**Figure 24: Accuracy Scores versus k with All Discovered Features**

****

**Figure 25: Accuracy Scores versus Norm with All Discovered Features**

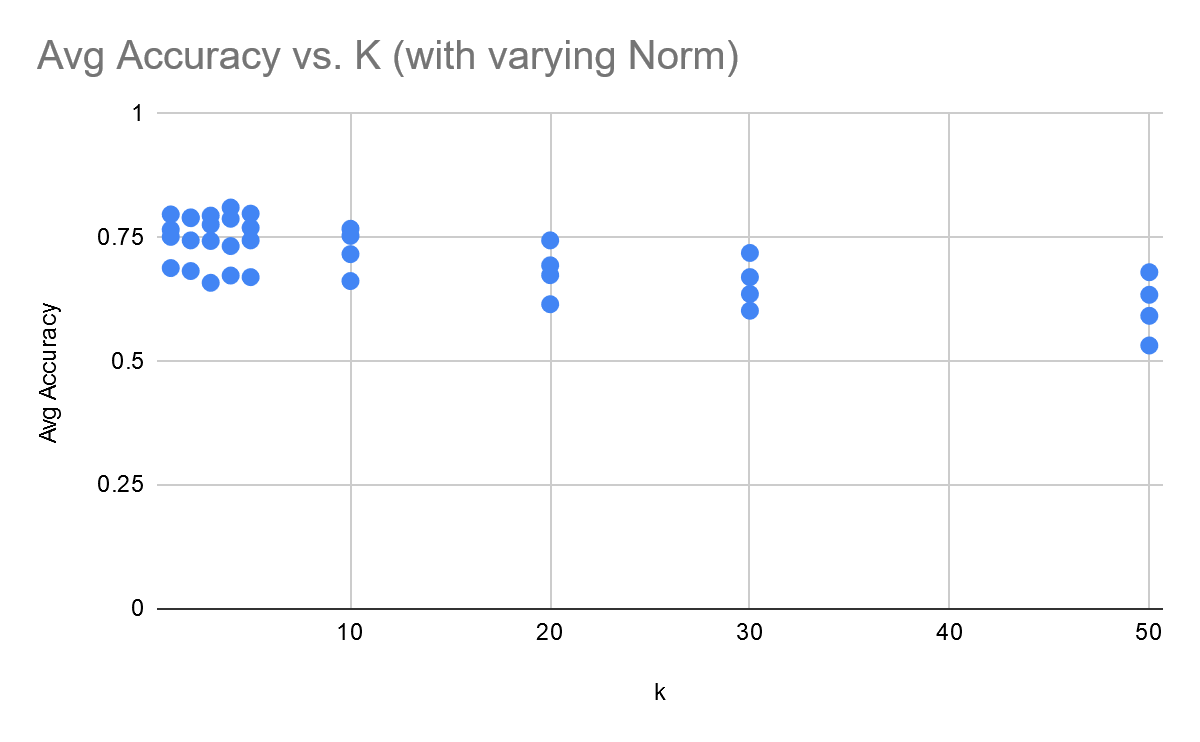
Despite more features, the accuracy scores actually went down by a good amount. With this many features, the high dimensionality may be starting to play a role in the accuracy of KNN. The team decided that for KNN, at least, the best features to use are the MFCC coefficients and the percussive and harmonic average.

As one final analysis, the team removed all data that was not manually verified and kept only the MFCC coefficients and the harmonic/percussive average. The results below are for various norms and values of k, with an average accuracy calculated over 10 random test/train splits of 20/80 percent respectively (100 percent of the data being used):

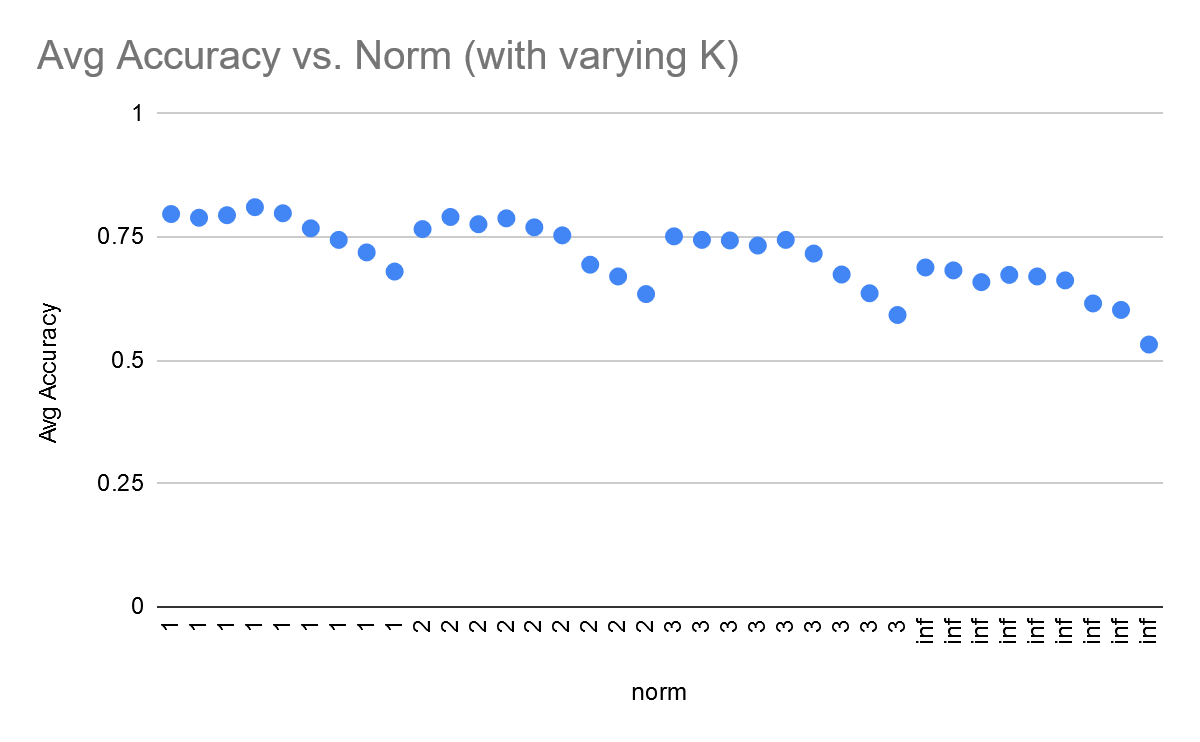
**MFCC Coefficients + Harmonic/Percussive Average (Manually verified data only)**

|  |  |  |
| --- | --- | --- |
| **K** | **Norm** | **Average Accuracy** |
| 1 | 1 | 0.7968253968 |
| 2 | 1 | 0.7892857143 |
| 3 | 1 | 0.7944444444 |
| 4 | 1 | 0.8107142857 |
| 5 | 1 | 0.7984126984 |
| 10 | 1 | 0.7678571429 |
| 20 | 1 | 0.7444444444 |
| 30 | 1 | 0.719047619 |
| 50 | 1 | 0.6801587302 |
| 1 | 2 | 0.7662698413 |
| 2 | 2 | 0.7908730159 |
| 3 | 2 | 0.7761904762 |
| 4 | 2 | 0.7880952381 |
| 5 | 2 | 0.7698412698 |
| 10 | 2 | 0.7535714286 |
| 20 | 2 | 0.694047619 |
| 30 | 2 | 0.6702380952 |
| 50 | 2 | 0.6345238095 |
| 1 | 3 | 0.7515873016 |
| 2 | 3 | 0.7444444444 |
| 3 | 3 | 0.7432539683 |
| 4 | 3 | 0.7329365079 |
| 5 | 3 | 0.7444444444 |
| 10 | 3 | 0.7166666667 |
| 20 | 3 | 0.6742063492 |
| 30 | 3 | 0.6361111111 |
| 50 | 3 | 0.5920634921 |
| 1 | inf | 0.6884920635 |
| 2 | inf | 0.6825396825 |
| 3 | inf | 0.6587301587 |
| 4 | inf | 0.6734126984 |
| 5 | inf | 0.6702380952 |
| 10 | inf | 0.6623015873 |
| 20 | inf | 0.6154761905 |
| 30 | inf | 0.6023809524 |
| 50 | inf | 0.5321428571 |

**Figure 26: Accuracy Scores for values of K and Norm for MFCC Coefficients + Harmonic/Percussive Average (Manually verified data only)**



**Figure 27: Accuracy Scores versus k for MFCC Coefficients + Harmonic/Percussive Average (Manually verified data only)**

****

**Figure 28: Accuracy Scores versus Norm for MFCC Coefficients + Harmonic/Percussive Average (Manually verified data only)**

With these parameters, the max accuracy reached over 80 percent (max value: 81 percent with k=4 norm=1), a massive increase compared to the original 30 percent accuracy scores. The trends are very similar to that of MFCC + Percussive/Harmonic average, although removing the automatically tagged data seemed to increase the accuracy by around 10 percentage points. One downside of only using the manually verified data, however, is that there are many fewer data points to train on and compare against, which could affect the overall generalization of the model to other datasets.

The team submitted KNN with norm=1, k=1, using the MFCC and harmonic/percussive average features, and 100 percent of the training data, and received a public leaderboard score of **72.427%**, only about 0.53% from the predicted accuracy score (using the 80/20 train/test split with the same parameters).

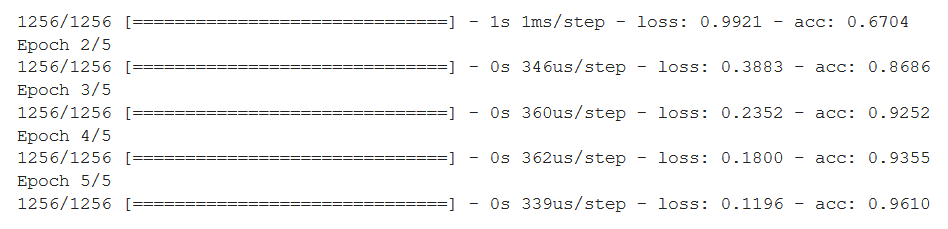
The team also submitted KNN with norm=1, k=4, using the MFCC and harmonic/percussive average features, only the manually verified portion of the training data, and received a public leaderboard score of **75.720%,** about‭ 5.4%‬ from the predicted accuracy score (using the 80/20 train/test split with the same parameters). This higher margin of error in the predicted accuracy could be due to having less data to train on, which could impact the model when facing a much larger data set.

**Random Forests**

For the Random Forests model, one of the challenges that presented itself early on was that it was not as good at classifying the data as the team had hoped. The first attempt yielded only about **25%** accuracy with the test data in the Kaggle competition, which was by far the worst model that was trained. There are many parameters that can be set to manage how decisions in the random forest are made, and the team attempted many times to optimize these parameters to the data. The team was able to boost the accuracy by 2% by increasing the number of decision trees from 100 to 1,000. There were additional opportunities to do more parameter tuning, but many resources that the team found seemed like they would lead to overfitting so the team focused the time on other models.

**Neural Network**

The team made sure to adjust the hyperparameters of epoch, batch size, and the number of layers to arrive at the best model. For efficiency, having a low batch size number allowed for less memory and faster training. Since the network is trained on fewer samples, the overall procedure requires less computer memory. Secondly, using small batches allows for increased speed of training by updating the weights after each propagation. As for deciding on the number of epochs for the best performing model (5), the team prevented the model from overfitting by choosing to pass through the training data a limited number of times.



**Figure 29: Training Accuracy Output From Neural Network**

For deciding on which layers to implement, the team decided that using three hidden layers consisting of 128 nodes that used the Relu activation functions and one output layer that used ten nodes an the softmax activation.

We evaluated these the best number of each of these by splitting our given training data into train and test data, and determined whether a model was too overfit by comparing the accuracies of the test and train data results with each other. If the accuracy of the training data was much higher than that of the test data, it indicated that we were training too much on our data which resulted in overfitting. After doing this, we then kept the best parameters and inputted all of the training data into the model. This greatly boosted the performance of our model, as we were able to improve from a 77% to **88.9%** model accuracy by changing these parameters.

**Conclusion**

Overall, the team would consider the project a success. The team was able to examine and analyze the proper features of the given audio samples that would best separate different instruments, preprocess the data such that noise was removed and the important features extracted, and evaluate various models to determine the ones with the best performance. In the end, the team trained a model that could classify the public test data with almost 90% accuracy, maintained position in the private leaderboard (overfitting avoided), and created other models with comparatively high accuracy as well.

**Links to Code**

<https://drive.google.com/open?id=1gm_KfEpy4B_U2DQ4MFNmYvZAPhrRNsHm>

* Anirudh’s Workspace

<https://drive.google.com/open?id=1WQNnIRJR-BfUcPRbOxjr-QZA1M_U0vvL>

* Rene’s KNN

<https://drive.google.com/open?id=1Qi7GdaIjL9YRXDu3j1g-dJSefNuHWOO6>

* Zach’s Sandbox

<https://drive.google.com/open?id=1lWplwPYH1MLhDwKRs5iXbh5NflJDpR-L>

* Josh’s notebook

<https://drive.google.com/open?id=1UEKhi9v7tqvoSFumbT-o-QPLJ9Ds5GbN>

* Sacred Neural Network