Musical Instrument Classification from Recorded Audio

Georgia Tech ECE-6254

Spencer Gass
Huntsville, AL
sgass6@gatech.edu

Sam Tanner
Oklahoma City, OK
stanner6@gatech.edu

Tim Coulter
Syracuse, NY
tcoulter6@gatech.edu

Eli Johnson
Chattanooga, TN
ejohnson340@gatech.edu

Project Summary:

In recent years, advances in image recognition and natural language processing have enabled many applications that were previously impossible. Less focus has been put on applying machine learning techniques to recorded music and as a result there is still a lot to explore. Our project involved evaluating the effectiveness of a number of different classification methods on their ability to determine the type of instrument playing a single note in an audio recording.

In order to do this the Nsynth[1] data set was used, which includes over 300,000 recordings of musical instruments playing all the notes in their range. Having hundreds of thousands of recordings that each contained 4 seconds of 16000 sample per second audio was both a great opportunity and a great challenge. The large quantity of data allowed for the use of deep neural networks but the high dimensionality and large quantity of data was computationally burdensome for SVMs, and KNNs.

The linear classifiers achieved the highest test accuracy. Several SVMs were trained with varying results. A linear SVM achieved 98% accuracy on the test set. The polynomial kernel achieved 89% accuracy on the test set, and the RBF kernel achieved an accuracy of 57% on the test set. A random forest was trained, but only achieved an accuracy of 30% on the test set. A linear classifier was also trained using stochastic gradient descent and achieve an accuracy of 96% on the test set. A five layer neural network was trained using stochastic gradient descent and was able to achieve an accuracy of 54% on the test set. The final classifier that was evaluated was a relatively small convolutional neural network derived from VGGnet[2]. The CNN achieved an accuracy of 72% on the test set.

Detailed Description:

While the fundamental frequency of a recorded note can be determined with conventional signal processing methods, instrument detection is still an open research topic and is a natural fit for machine learning algorithms. Early attempts to classify the musical instrument being played

[1] "The NSynth Dataset - Magenta - TensorFlow." 5 Apr. 2017, https://magenta.tensorflow.org/datasets/nsynth. Accessed 21 Apr. 2018.
[2] "Very Deep Convolutional Networks for Large-Scale Image Recognition." 10 Apr. 2015, https://arxiv.org/pdf/1409.1556. Accessed 21 Apr. 2018.

in an audio recording leveraged cepstral coefficients and qualitative features along with k-nearest neighbor and Gaussian mixture model classifiers. These methods resulted in accuracies of 35% when classifying type of instrument and 77% when classifying instrument family (i.e. brass, string, woodwind, etc.).[3] A more recent method used dimensionality reduction methods with k-nearest neighbor and achieved high accuracy but on a small test set of 108 samples.[4] The use of a sparse coding approach to compactly represent the cepstral information yielded an F-score of 0.955 on a slightly larger data set of 273 audio segments.[5]

After reviewing the literature, Eli determined that Mel-frequency cepstral coefficients (MFCC) were the primary feature used in audio classification[6][7]. MLCCs are often cited because of their ability to more closely approximate the response of the human ear than a typical spectral decomposition. Additionally, spectral centroid, spectral bandwidth, spectral slope, and zero-crossing rate were added to the feature vector. These features have been previously shown to be among the most effective spectral measurements for distinguishing between instrument classes[8]. Initially the resolution parameters for these metrics were determined by doing validation runs with a small two layer neural network but this produced a feature vector that was too large to be useful for the other classifiers, so a reduced feature vector was created to train the KNNs and SVMs.

The NSynth data came already divided into 289,305 training samples, 12,678 validation samples, and 4,096 test samples. There are eleven instrument types which are not evenly represented in the data set. Bass was the majority class represented by 68,955 samples while synth lead was the minority class represented by 5,501 samples.

Sam was responsible for training an Radial Basis Function (RBF) kernel SVM. The SVM training algorithm was implemented using Scikit Learn. With such a large data set to train on, the SVM could not be trained outright. For a linear SVM a stochastic gradient descent could be used to accelerate the training. For kernelized SVM, stochastic gradient descent is not easily possible. Instead an approximation of the feature map of an RBF kernel was used. The Scikit Learn module RBFSampler was used for this approximation, which implements a method called Random Kitchen Sinks.[9] Random Kitchen Sinks accelerates learning by mapping the data to a

---

[3] "Musical instrument identification in continuous recordings - Hal." https://hal.archives-ouvertes.fr/hal-01156882. Accessed 22 Mar. 2018.

[4] "automatic musical instrument recognition - CiteSeerX." http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.79.6635&rep=rep1&type=pdf. Accessed 22 Mar. 2018.

[5] "IEEE International Conference on Acoustics, Speech and Signal ...." https://researchr.org/publication/icassp-2014. Accessed 22 Mar. 2018.

[6] "Computational Intelligence and Signal Processing (CISP), 2012 2nd ...." http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6183749&filter%3DAND%28p_IS_Number%3A6189660%29&pageNumber=2. Accessed 22 Mar. 2018.

[7] "Musical instrument identification using MFCC - IEEE Xplore." http://ieeexplore.ieee.org/document/8256990/. Accessed 22 Mar. 2018.

[8] "Role of Features and Classifiers on Accuracy of Identification of Musical Instruments - IEEE Xplore." http://ieeexplore.ieee.org.prx.library.gatech.edu/stamp/stamp.jsp?tp=&arnumber=6189710&isnumber=6189660. Accessed 22 Mar. 2018.

[9] "4.6. Kernel Approximation — scikit-learn 0.19.1 documentation." http://scikit-learn.org/stable/modules/kernel_approximation.html. Accessed 23 Apr. 2018.

randomized low-dimensional feature space, which allows the use of faster linear methods.[10] Essentially, it is a technique that randomly selects features, allowing the algorithm to be much faster, with surprising accuracy. The training data, and later the test data, were standardized before training or testing with the RBF SVM.

The hyper parameters selected for this function include 'C' and 'Gamma,' like a normal RBF SVM, as well as the number of Monte Carlo samplings. 'Gamma' defines the influence of a single training example, and 'C' defines the trade off between simplicity and acceptable error. 'C' and 'Gamma' parameters were selected using a holdout set of 5,000 samples, from the reduced feature training data. A range for each parameter was evaluated using grid search methods, from 10^{-2} to 10^{10} for 'C', and from 10^{-9} to 10^{3} for 'Gamma.' The values with the best performance in the grid search on the holdout set, 26.82 for 'C,' and 0.000139 for 'Gamma,' were used to train on all the samples in the training set. The number of Monte Carlo samplings was selected as 1,000. Higher samples more accuracy, but at a higher time cost. Sampling 1,000 times was as high as could be run in reasonable time, and without using excessive memory.

The classifier was then trained on the reduced feature data set, using all 289,305 samples. The classifier took about 75 minutes to train. Then the classifier was tested against the test set, and achieved an accuracy of 57%.

Sam was also responsible for the Random Tree classifier. This was not part of our original proposal, but it is an interesting different type of classifier. It averages the predictions of several independent base estimators, to reduce the variance and make a classifier more generalizable.[11] The data was not standardized for the Random Tree classifier.

The Random Tree classifier has many more parameters than can be efficiently computed. Therefore six of the parameters were varied independently, while training on a small holdout set. Graphs of each varying parameter vs the score of the fit were generated. The graphs enabled a much reduced set of possible parameters to be evaluated with a grid search.[12] This method reduces the time spent calculating parameters, but the optimum parameters could still be outside the ranges chosen. The parameters selected for the Random Forest were 20 maximum leaf nodes, 5 maximum depth of the tree, 0.1 minimum weight fraction per leaf, 200 sub-estimators to average, 19 minimum number of samples to split and internal node, and a minimum of 9 samples per leaf.

The resulting parameters were used to create the Random Forest classifier tested on the full featured training set, with 289,305 samples. The classifier took about 45 minutes to train. The resulting classifier fit the test data with 30% accuracy. This might have been improved with further tuning of the hyper parameters.

---

[10] "Random Features for Large-Scale Kernel Machines." https://www.robots.ox.ac.uk/~vgg/rg/papers/randomfeatures.pdf. Accessed 23 Apr. 2018.
[11] "Random Features for Large-Scale Kernel Machines." https://www.robots.ox.ac.uk/~vgg/rg/papers/randomfeatures.pdf. Accessed 23 Apr. 2018.
[12] "Random Features for Large-Scale Kernel Machines." https://www.robots.ox.ac.uk/~vgg/rg/papers/randomfeatures.pdf. Accessed 23 Apr. 2018.

Tim worked on the Linear SVM/SGD (stochastic gradient descent) and K-Nearest Neighbors classifiers. Initially, a SGD based algorithm was thought to be the best approach for an SVM type classifier. The RAM requirements were too large to use the entire training set and compute the maximum margin hyperplane using an “offline” method. An SGD based classifier can batch the training data and work in an “online” manner, taking multiple sets of samples.

The SGD from SKLearn offered many loss models that could emulate various classifiers, including a linear SVM (hinge loss model). This made it an ideal candidate for comparing the performance of a linear SVM using a subset of the training data to an SGD classifier that was able to use the whole training set. Some extra caution is needed when using an SGD. In order to get the best performance from an SGD the data needs to be scaled and the classes have to be weighted in order to not produce an bias. Given that all the data cannot be loaded into memory at once, a large batch size was chosen and the data was shuffled so that the scaling for each batch would hopefully approximate scaling across the entire set. The most difficult part of using an online version of the SGD was figuring out the stopping criterion. The offline SGD that fits all the data at once has built in functionality to stop based on loss tolerance. However, there is no way to do this with an online SGD and, therefore, a certain number of iterations was utilized. This made the online SGD more prone to not converging on a solution. In fact, the results were that the online SGD only achieved an accuracy of 45% on the final test set. With more time, a more appropriate way of creating convergence criterion for the batched SGD would have been created.

Given the results of the online SGD classifier, a different path was eventually chosen. A reduced set of 12,000 samples was used to train on. Both a Linear SVM and offline SGD was tested. This had far better results compared to the online SGD. The learning rate for the offline SGD was trained on a validation set and it produced a classifier that had 96% accuracy on the test set. The Linear SVM was able to achieve an accuracy of 98% which was the highest accuracy among all the classifiers we tested. The K-Nearest Neighbors classifier was also tested on this reduced test data set. It performed above average with a score of 70%. Given the dimensionality of the selected features it was interesting that the linear classifiers were so effective. It shows that sophisticated kernels are not always needed, even on data sets that do not appear to be optimal for linear classification.

Eli was responsible for training an SVM with a polynomial kernel. The common python library SKLearn was used for training and scoring. The complexity of this algorithm increases with order N^2, so these libraries do not easily scale to data sets larger than a few 10,000 points. Because of this limitation, a subset of the training data was used along with a reduced dimensionality feature vector compared to that which was used for the neural networks. The most significant reduction in features came from reducing the number of MFCCs from 256 to 20, and doubling the sampling window over which they were calculated. A set of 12,000 points was used to train this classifier.

The first step in training this classifier was to evaluate choices of the parameters 'C' and 'Gamma'. C parameterized the acceptable error and Gamma parameterized each sample's area of influence. A small holdout set of 2,000 samples was used to evaluate varying choices of C and

Gamma with a grid search strategy. The values chosen were 1 and 0.001, respectively. These corresponded to the smallest parameter values that still resulted in the minimum holdout set error.

Using the above parameters, a set of classifiers of varying degree was trained and evaluated on a new holdout set. Significant decreases in holdout error were found for degrees two through seven. Polynomials of degree eight and above saw the holdout error begin to increase rapidly.

Finally, the 7th-order polynomial kernel SVM classifier was tested using the given test set of 4,096 samples. It achieved an accuracy of 89.2% on this test set.

Spencer was responsible for training and evaluating a neural network and convolutional neural network (CNN). Both the neural networks and CNNs were implemented using a python library called Keras which runs on top of Tensorflow. Tensorflow can be configured to run most of its algorithms on a graphics card in order to increase performance. An Nvidia GTX-970 graphics card was used which gave approximately a ten fold decrease in training times as compared to using an Intel I7 to execute the training algorithms.

Neural nets will learn the a priori probability of a class and so it's important to train them using an equal proportions of each class. Both oversampling and undersampling were considered as methods for making the number of samples of each class equal and over sampling was selected[13]. This exposed minority classes to the possibility of over fitting but allowed for all of the training data to be used.

Stochastic gradient descent (SGD) was used to train both the neural net and CNN. Learning rate is the primary SGD parameter and has a major effect on the ability of SGD to reach a minima or converge at all. 10^{-2} is a common starting point and learning rate is often reduced as training goes on. We chose to set the learning rate to 10^{-3} throughout the entire training run in order to simplify the process. This worked well. A categorical cross entropy loss function was used. This is a common loss function for multi-class classifiers. Batch sizes varied between 50 and 200 and were restricted by graphics memory availability.

There are a number of hyper parameters that determine the structure of the neural network and as a result, its ability to learn effectively. Some of the most important hyper parameters are: number of hidden layers, number of neurons in each hidden layer, dropout rate, and activation function. These parameters, as well as SDG learning rate were initially selected by using a subset of the training data as a holdout set and another subset as a training set. Once parameters were selected the model was trained using the entire training set and the validation set was used to determine when to end training. The most effective neural net had five hidden layers that were 1024 neurons wide with a dropout rate of 0.2 between each fully connected layer and ReLU activation functions on all but the output layer which used a softmax activation.

The neural net converged in just 4 epochs which took about one hour. The resulting classifier has an accuracy of 54% on the test data set. This neural net has approximately 38

[13] "A Review of Class Imbalance Problem - Ajith Abraham." http://www.softcomputing.net/jnic2.pdf. Accessed 21 Apr. 2018.

million learned parameters. The majority of which are in the input layer (approximately 32 million) and are the result of the high dimensional input data.

The CNN classifier that we used was based on VGGnet, one of the highest performing CNNs for image classification. However since the complexity of detecting one of eleven instrument types is a much simpler problem than classifying one of a thousand classes of objects that could be present in an image, we opted to reduce the depth and width of the network. The result was a network with only 600 thousand parameters as opposed to the 133 million in the smallest version of VGGnet. This CNN has a classification accuracy of 72% on the test data set.

Discussion:

Comparing the results of the polynomial and linear SVMs, it seems reasonable that the reduced data set of 12000 samples was sufficient to train a simple linear classifier but not quite enough for a more complex polynomial classifier and as a result the 7th degree polynomial SVM overfit the training data slightly. Since the SGD linear classifier and linear SVM both find a linear classifier, it makes sense that they would receive comparable scores. Since the data is not radial in nature it is also reasonable that a RBF SVM would not be best suited for this task.

Going into the project we would have expected the more complex classifiers to outperform the simpler ones and were surprised by the effectiveness of the linear classifiers compared to the neural nets. However if a line can divide the data set with 98% accuracy, any classifier with more degrees of freedom is only going to do a better job over fitting the training set. Perhaps the takeaway is that if you understand the structure of the data then you can extract features in a way that makes the data easily separable by a linear classifier. This can be much more effective that letting a neural net learn which features to extract.

If we were to continue this research it would be interesting to explore how these classifiers performed for more challenging classification tasks such as multiple sound source audio, note/velocity detection, as well as higher level tasks like tonality and rhythm detection.

Conclusion:

Given a large high dimensional data set, linear classification can still be very effective. It is easy for large amounts of high dimensional data to have large amounts of redundant information. Through effective feature selection and undersampling of the data set, that redundancy can be removed and linear classification can be highly effective. More complex classifiers can be useful for classification tasks where this structure does not exist or is not understood but they underperform when the data can be effectively separated with a hyperplane.