Musical Note Classification: Accuracy vs Runtime

Casey Justus, Tara Khanna, Qiaochu Xiong, Michael Zhou

ABSTRACT: Our paper is about helping people learn music by classifying musical notes into five types of notes: whole, half, quarter, eighth, and sixteenth notes. We classified musical notes using four algorithms: logistic regression, histogram gradient boosting trees, fully-connected neural networks, and convolutional neural networks. We applied each of these four algorithms to both 28 x 28 images and 64 x 64 images. When choosing parameters, we used both grid search and randomized search to perform cross validation for the first three algorithms, while we manually tuned parameters for convolutional neural networks. Histogram gradient boosting trees with parameters chosen by grid search CV achieves the highest test accuracy for 28 x 28 images, while the convolutional neural network with manually tuned parameters achieves the highest test accuracy for 64 x 64 images; however, they both take significantly longer to perform CV and train on compared to other methods. Logistic regression has the fastest runtime of all the algorithms for both 28 x 28 and 64 x 64 images.

Introduction:

Are you a beginner in music and still need to learn how to read the different notes? A struggling piano student alone at home, needing some automated assistance? Symbolic music representation entails parsing the structure of music, including individual notes, different events and cadences. It has been prevalent in a variety of music applications including interactive music tutorials, music theory courses, ear training, music production, and play training **[1]**. While symbolic music representation spans a wide range of tasks, we specifically focus on musical note classification - detecting visually whether each note is a whole, half, quarter, eighth, or sixteenth note.

We used two different datasets of 5000 musical notes - one for 28 x 28 images (small), the other for 64 x 64 images (large) - each consisting of whole, half, quarter, eighth, and sixteenth notes, 1000 per category **[2]**. Based on the grayscale pixel values (0-255) of the images, we wanted to classify each musical note into each category, and measure how efficient it was to train the examples and do model selection using parallel workers (for efficiency), along with how the size of the images can affect this. Our algorithms include logistic regression, histogram gradient boosting trees, fully-connected neural networks (FCNNs) and convolutional neural networks (CNNs).

Then using our different analysis we looked into how the runtime was affected by the method as well as parameters used. We noted trends in mean fit time based on several parameters and saw the trade-off between test accuracy and training time as well as CV error and time to find the optimal parameters.

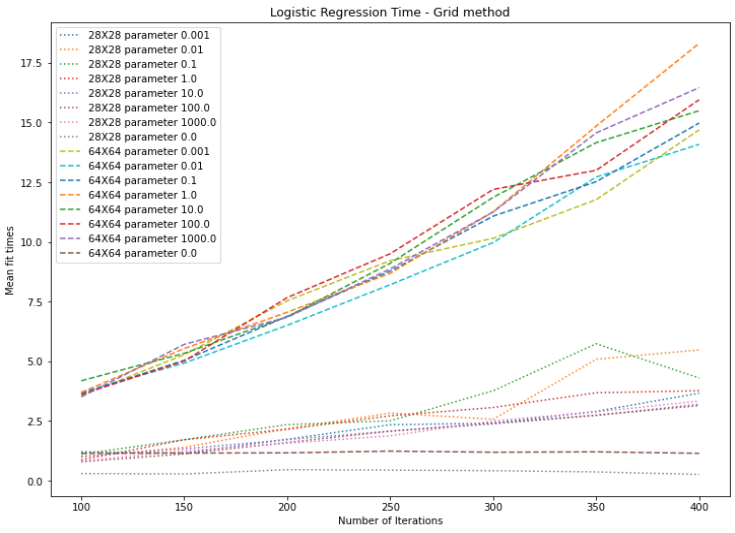
Logistic Regression:

We first used a baseline for classifying the musical notes - logistic regression. We wanted to find the performance of logistic regression in terms of the test accuracy, training time, along with the time it took for performing cross-validation (CV), on both the small (28 x 28) and large (64 x 64) image datasets. We also analyzed the mean fit times for each hyperparameter configuration tried in CV to analyze trends within grid search and see how they map out in the randomized search.

Using the logistic regression model via scikit-learn, we obtain a 70-30 training-test split on both the small and large datasets **[3]**. We then implemented a 5-fold, full grid search CV over the regularization parameter C (1^m where m is an integer from -3 to 3 inclusive, along with 0) and the maximum number of iterations (100, 150, 200, …, 400), all using l2 penalty loss and parallel workers. For small images, we found that the best hyperparameters (C = 0.001, max iterations = 350) gave the lowest CV error, taking 156.389 seconds to obtain these configurations; after 2.622 seconds of training, we obtained a 0.862 test set accuracy. For large images, it took 590.079 seconds to find the best set of hyperparameters (C = 0.01, max iterations = 350), and a 0.848 test accuracy after 7.534 seconds to train on them. As we can see, we can get a respectable baseline test accuracy with logistic regression without too much computational cost.

We repeated the above process for randomized search CV, setting the random seed to 1 and running 10 iterations (using the same grid distribution as in grid search). For the small dataset, it took us 23.378 seconds to find the optimal hyperparameters (C = 0.001, max iterations = 200), 0.733 seconds to train on them, obtaining a 0.8517 test accuracy. For the large dataset, it took us 123.622 seconds to find the optimal hyperparameters (happened to be the same as for the small dataset), 3.968 seconds to train on them, obtaining 0.8426 test accuracy.

We also wanted to see how the overall mean fit time changes with respect to parameters as well as number of iterations. It is important to note that we ran both grid search and randomized search CV in parallel threads (using all available workers on the machine), so the sum of all the mean fit times do not add up to the overall CV search time. To understand the overall progression and trends of mean fit time versus the number of iterations we can graph the difference between the two giving us these two graphs to analyze trends. For both the 28 x 28 and the 64 x 64 we can see a clear linear trend between number of iterations and



mean runtime (other than the 0.0 parameter) which makes sense since we are linearly making more iterations. Additionally the separate parameters, especially in the 64 x 64, have no distinct difference in what parameter is used but as the number of iteration goes up, a large disparity between different parameters seems to arise. Additionally, particularly with the 28 x 28 in higher number of iterations, the data seems to be more noisy since disparities in smaller amounts of time cause bigger ripples. Diving into this further we see a seemingly cubic relationship between the overall trends of the 28 x 28 and the 64 x 64. Additionally, while looking at data points coming from the randomized search, the total time is obviously shorter due to only choosing a subset of points but it overall follows the trends of the given data. However due to the large disparity in overall training versus the little gain in CV error comparatively, the randomized search is a better option for finding the best parameters with logistic regression.

Histogram-based Gradient Boosting Trees:

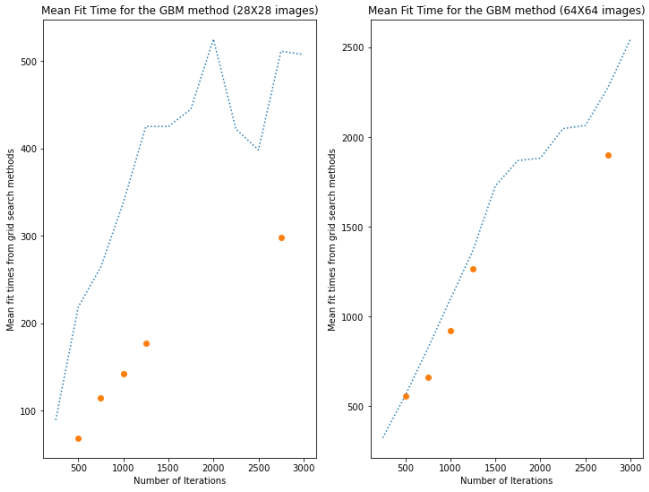
After logistic regression we used histogram-based gradient boosting. Extreme gradient boosting, or XGBoost, is a highly efficient and scalable implementation of boosting algorithms that has been responsible for winning many competitions on Kaggle. One of the most common techniques of XGBoost is the use of histograms, which can drastically speed up the process of training despite slightly more sophisticated training processes **[4, 5]**.

Following a very similar procedure to logistic regression, we found the test accuracy of gradient boosting trees, training time, along with the time it took for performing CV - on both the small (28 x 28) and large (64 x 64) image datasets - and also analyzed the mean fit times for each hyperparameter configuration tried in CV. Using the histogram GBM model from scikit-learn, we implemented a 5-fold grid search CV over the # of trees B from 250 to 3000 (intervals of 250), fixing the learning rate at 0.01 and maximum depth at 4. On the small images dataset, after 6105.601 seconds of grid searching, we found the optimal B at 2500; after 223.285 seconds of training for this B, we obtained a test accuracy of 0.9617. The large images dataset took 24686.569 seconds of grid searching, we found the optimal B at

2250; after 994.838 seconds of training for this B, we obtained a test accuracy of 0.938. Although the test accuracies for both datasets are better than logistic regression, we had to tradeoff a lot more CV and training time due to the more sophisticated procedures of GBMs. Histogram GBMs are one of the most efficient implementations of GBMs.

We wanted to see how randomized search CV affects the overall test performance. After choosing 5 random configurations (with seed 1) from the same grid as in grid search, the test accuracies for both the small and large datasets turned out very similar (0.961 and 0.939 respectively), while taking only 1160.949 seconds and 7646.928 seconds of search for the small and large datasets respectively. Since we are only searching over 5 of the 12 hyperparameter candidates, we expect this drastic decrease in CV search time. The training times were 223.285 and 994.838 seconds for the small and large datasets respectively. B = 2750 was selected as the optimal hyperparameter for both datasets.

Once again, all iterations of both grid search and randomized search CV are run in parallel. Similar to what we saw with logistic regression, we note a quadratic difference between the large and small data with a linear trend between number of iterations and mean fit time of the GBM method with respect to number of iterations. Yet GBM produced large disparities between runtimes for grid and randomized search, particularly in for 28 x 28 images. Noting the graphs on the right, though the randomized points lie near the trendline for 64 x 64 images, the random points taken for 28 x 28 images have a much shorter runtime than their grid search counterparts. Although the time spent was significantly shorter for the 28 x 28 search, it landed upon the same optimal CV value and test accuracy, better than what we found for logistic regression. Additionally, for 64 x 64 images, while the mean fit time was only marginally smaller than of the grid search for the



optimal value, the algorithm found a point with a very similar CV and test accuracy that were in less than a .1% of one another.

Fully-Connected Neural Network:

One way we tried classifying the musical notes was by implementing a fully connected neural network (FCNN). An FCNN is composed of a series of fully connected layers that connect every node in one layer to every node in the other layer **[6]**. A fully connected layer is a function from R^m to R^n, where m and n are the number of nodes in their respective layers **[6]**. This means that each output dimension depends on each input dimension, and many different functions, called activation functions, can be applied between the layers. A major advantage of this network is that there are no special assumptions that need to be made about the input, making them broadly applicable **[6, 7]**.

We implemented scikit-learn's neural network library using the default 'adam' solver, as it works well on relatively large datasets in terms of training time and CV score. In order to choose the best parameters for the neural network, we performed a full grid search over different parameters such as the following: whether or not there is early stopping (True, False), hidden layer sizes (100, 200, 300, 400, 500), activation functions (relu, tanh, logistic), learning rates (constant, invscaling, adaptive), and learning rate initializations (0.0001, 0.001, 0.01, 0.1). When we ran the neural network using the 28x28 image data, the grid search found the following optimized parameters: the activation function was logistic, no early stopping, hidden layer size of 500, learning rate is constant, and the learning rate initialization is 0.0001. This produced training and test accuracies of 0.8787 and 0.8331, with CV search and training times of 2651.10 and 20.78 seconds respectively. When we ran the neural network using 64x64 images, a different set of optimized parameters were selected, which are as follows: relu activation with early stopping, hidden layer size 400, adaptive learning rate, and learning rate initialization 0.0001. For this set of parameters, the bigger dataset took 8822.688 seconds for grid search CV. The training took 25.48 seconds. This produced a training accuracy of 0.803 and a test accuracy of 0.722.

The timing with respect to the FCNN varied based on the parameters used. Splitting the data by activation functions, learning rates, learning rate initialization, and ability to stop early showed interesting overall trends. Similar to all the other tests, with increased hidden layer sizes, the overall time increases approximately linearly but unlike the logistic regression, the graphs were slightly noisier, more similar to the 28 x 28 GBM graphs. One key trend with the data pertained to the parameter of early stopping, where with early stopping the mean fit time seemed to be less with data lines of initial learning rate initializations being closer together. Another key difference between time is that tanh had shorter mean fit times on average than both relu and logistic activation functions for both all learning rates and for both options of early stopping. While constant, invscaling, and adaptive learning rates all had different approaches with different CV errors, training accuracy, and testing accuracy, there were no distinct differences between their runtimes.

After implementing this neural network with the adam solver, we wanted to explore how the LBFGS solver would perform on our data set, as it converges faster and performs better for small data sets. Specifically, we wanted to see how much does using the LBFGS solver affect the final test accuracy. We changed the grid search CV to use this solver, and after running for 11 hours on the 28x28 images and observing that it is barely halfway finished, we concluded that the LBFGS is not feasible for our 5000-example large dataset. Therefore, we did not try to use the LBFGS solver for the 64x64 images. Additionally, in an attempt to decrease the amount of time it took to run the network, we tried running a randomized grid search CV over the same parameters. The non-randomized grid search took 2650.102 seconds to run on 28x28 images and 8822.688 seconds to run on the 64x64 images. The randomized grid search ended up selecting similar optimized parameters for the 28x28 images, producing a training accuracy of 0.87 and a testing accuracy of 0.8174, and taking 240.871 seconds to run. However, it did not work nearly as well for the 64x64 images, as it selected optimized parameters that produced a training accuracy of 0.672 and testing accuracy of 0.652, though it significantly decreased the runtime as it only took 583.669 seconds to run.

Convolutional Neural Network:

CNN, Convolutional Neural Network, is particularly useful when working with image data, so CNN is very suitable for our image classification project. It contains one or many pairs of convolutional and pooling layers **[8]**. After convolutional and pooling layers, a flatten layer follows, and then a dense layer, and finally an activation layer. Since we wanted to classify a new image into one of the five classes: 'Whole', 'Half', 'Quarter', 'Eight', and 'Sixteenth', we need the dense layer to have output dimension 5. Since we are working on a multi-class classification task, for the activation layer, we use the activation function 'softmax', which converts the output of the dense layer into probabilities of an image belonging to each of the five classes **[9]**. Finally, when we compiled our models, we used the 'categorical_crossentropy' loss function **[11]**; for metrics, we used 'accuracy', which measures how likely predictions we get are the same as the true label **[12]**.

When preprocessing data, we divided our dataset into 80% train-validation, and 20% testing. We further divided the 80% train-validation data into 80% train and 20% validation. We removed any training instances that contain the following values: NAN, infinity, and negative infinity. Due to the required input format of CNN, we reformatted datasets into the form of (number of instances, 28, 28, 1) for 28x28 images, and (number of instances, 64, 64, 1) for 64x64 images. The last dimension, which represents color channels, is 1 for both 28x28 images and 64x64 images because our images are grey **[10]**. We also changed the representation of class labels from strings into categorical variables, since the categorical_crossentropy loss function requires class labels to be represented by one-hot encoding.

When training model for 28x28 images, we manually tuned different combinations of parameters, for instance: number of convolutional/pooling pairs (1,2), dimensions of output for convolutional layers (16, 32, 40), activation functions (relu, sigmoid, tanh), and optimizer (adadelta, adam). For 28 x 28 images, we found that CNN with 2 convolutional/pooling pairs behaves consistently better compared to CNN with 1 convolutional/pooling pair. For CNN models with 2 convolutional/pooling pairs, models with output size (32, 40) for the first and second convolutional layers, respectively, have better performance than models with output size (16, 32) for the first and second convolutional layers,

respectively. Also, for activation functions, both ‘tanh’ and ‘relu’ are able to achieve accuracy more than 0.9, and ‘tanh’ behaves better than ‘relu’. However, ‘sigmoid’ has a testing accuracy of 0.7670. For optimizers, ‘adadelta’ is able to achieve accuracy more than 0.9, while performance of ‘adam’ is so bad that the model accuracy doesn’t even reach 0.5. As a result, the best model for 28 x 28 images has two convolutional/pooling pairs, with ‘tanh’ as activation function for both the first and the second convolutional layer, with output dimensions (32, 40), respectively, one flatten layer, one dense layer with parameter 5, and one activation layer with activation function ‘softmax’. When compiling the model, loss is ‘categorical_crossentropy’ and optimizer is ‘adadelta’. This model yielded 0.9636, 0.9350, 0.9340 training, CV, and test accuracies. Training time is 1772.190 seconds. We manually turned the same set of parameters for 64 x 64 images using the same procedure described before, and the same set of values for parameters are chosen. For 64 x 64 images, the training, CV, and test accuracies were 0.9866, 0.9588, and 0.9600 respectively. Training time is 15400.268 seconds.

Performance Results for 28 x 28 Images (above) and 64 x 64 Images (below):

Algorithm	CV Search Method	Test Accuracy	Cross-validation Search Time (seconds)	Training Time (seconds)
Logistic Regression	Grid	0.862	156.38888597488403	2.6218600273132324
Logistic Regression	Randomized	0.8517142857142858	23.3782320022583	0.7330210208892822
Histogram GBM	Grid	**0.9617142857142857**	6105.600783824921	223.2854940891266
Histogram GBM	Randomized	0.9611428571428572	1160.9488430023193	135.37225699424744
Fully-Connected NN	Grid	0.8331428571428572	2650.101747274399	20.77657198905945
Fully-Connected NN	Randomized	0.8174285714285714	240.87050485610962	15.206079959869385
CNN	N/A (Manually tuned)	0.9340000152587891	N/A (Manually tuned)	1772.1896510124207

Algorithm	CV Search Method	Test Accuracy	Cross-validation Search Time (seconds)	Training Time (seconds)
Logistic Regression	Grid	0.8477142857142858	590.0785450935364	7.534313201904297
Logistic Regression	Randomized	0.8425714285714285	123.62241625785828	3.9675979614257812
Histogram GBM	Grid	0.9382857142857143	24686.5692858696	994.8379521369934
Histogram GBM	Randomized	0.9388571428571428	7646.927664041519	708.2642109394073
Fully-Connected NN	Grid	0.7222857142857143	8822.687611818314	25.47961688041687
Fully-Connected NN	Randomized	0.652	583.6694300174713	15.283200979232788
CNN	N/A (Manually tuned)	**0.9599999785423279**	N/A (Manually tuned)	15400.268100738525

Conclusion:

Looking at the tables given above, Histogram Gradient Boosting Trees give the best test accuracy for 28 x 28, followed by Convolutional Neural Networks. As for the large data set, CNN performs better than Histogram GBM in test error. While these two algorithms produce better results, they are computationally far more expensive than that of logistic regression and fully connected neural networks.

Looking at runtime of our algorithms both in cross-validation search time and training time in conjunction with the test accuracy, we see that logistic regression appears the most consistent between the data sets with around a 0.85 test accuray and a far shorter runtime than the other functions. However overall, Historical GBM has the best overall performance across both results. In both data sets, the test accuracy ranks above 0.9 unlike some of its quicker algorithms and for the trade off in being less accurate, the training time is a small fraction of that of the CNN.

Bibliography:

1. Symbolic Music Representation: https://mpeg.chiariglione.org/standards/mpeg-4/symbolic-music-representation#:~:text=A%20symbolic%20representation%20of%20music,synchronized%20with%20other%20media%20types
2. Data source for Musical Note Classification: https://www.kaggle.com/kishanj/music-notes-datasets
3. Grid Search with Logistic Regression: https://www.kaggle.com/enespolat/grid-search-with-logistic-regression
4. Slides 7-8, Lecture 15, ORIE 4740 (SP 2021): https://canvas.cornell.edu/courses/24583/files/3488621?module_item_id=929557
5. Histogram-Based Gradient Boosting Ensembles in Python: https://machinelearningmastery.com/histogram-based-gradient-boosting-ensembles/#:~:text=Grad

ient%20Boosting%20Ensemble-,Histogram%20Gradient%20Boosting%20With%20XGBoost,for%20the%20continuous%20input%20variables

6. Neural Network explanation:
   https://medium.com/swlh/fully-connected-vs-convolutional-neural-networks-813ca7bc6ee5
7. Fully Connected Deep Neural Networks:
   https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html
8. Convolutional Neural Network (CNN):
   https://www.tensorflow.org/tutorials/images/classification
9. Softmax activation function explanation:
   https://machinelearningmastery.com/softmax-activation-function-with-python/#:~:text=The%20softmax%20function%20will%20output,%5B0%2C%201%2C%200%5D
10. Convolutional Neural Network explanation:
    https://www.youtube.com/watch?v=n2MxgXtSMBw
11. Keras Losses:
    https://keras.io/api/losses/
12. Keras Metrics:
    https://keras.io/api/metrics/