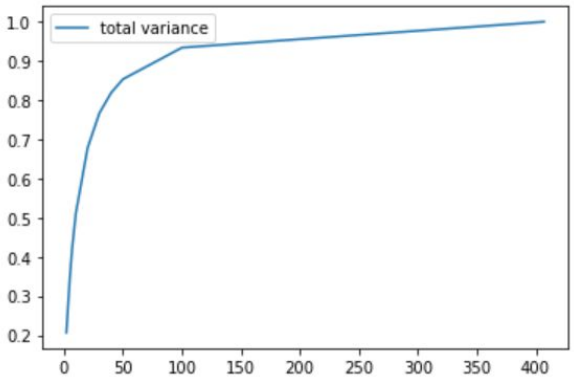
I first tested different classifiers with default sklearn parameters. I first performed a 70:30 train-test split on the provided trainset. The models were KNN with 1-5 neighbors, Logistic Regression, Decision Tree, Multi-layer Perceptron, and SVM with kernel tricks. When all of them were tested, they got test accuracies below 70%, except for Decision Tree with ~80%.

Next, I performed pre-processing of the data. First, I normalized by mean subtraction and standard-deviation division of the train-set. I learned that KNN and SVM benefited, but other models not as much. Secondly, I performed dimension reduction using PCA. I plotted the # of components vs variance plot and found that 100 components would explain 93%+ variance. Two sets of PCA dataset were created with 100 and 200 components. When SVM and KNN were tested with the new train set, their test accuracy improved but none was close to 90%.



# of components vs variance plot.

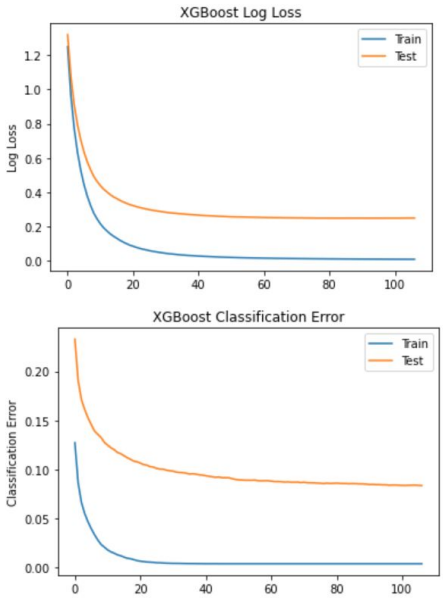
Since DT has the best performance, I went to test tree-based ensemble methods like RandomForest, AdaBoost, XgBoost, and Hist-Gradient Boosting, and hopefully to get better predictions. With default parameters, XgBoost has best test accuracy at above 90%, while others are at 75%-89%. So, I have decided to focus on tweaking XgBoost.

I learned about using Grid search, which tests all combinations of different parameter values to find the best combination. The main parameters that I tested are learning_rate from 0.03-0.3, max_depth from 7-12, and n_estimators from 500-4000. Other parameters like alpha and gamma were experimented, but they barely improved the accuracy. I learned that grid search is needed because the best value of every parameter when tested individually may not provide the best model when combined.

The gridsearch was initially performed with the 70:30 train-test split. I learned that in this task using a larger train set produces a better predicting model, as test accuracy in Kaggle using the entire train set is higher than using 70% train set during experimentation. As a result, I then used 80:20 train-test split to get a similar result to using the entire train set.

Performing Grid Search is most challenging because there are tons of combinations and each model took ~4 hours to train. I need to cleverly reduce the range of parameter values for less combinations. For example, I learned that the best model has depth of 8-10. I also learned that n_estimators and learning_rate are best adjusted by increasing one and lowering the other. Another method I tried was early-stopping, that will stop training when consecutive iterations have constant log loss or accuracy to

prevent overfitting. However, it is not effective as I noticed the test accuracy would continue to decrease after being stopped.



n_estimators vs error or log loss plot from early stopping.

At this point I felt a bottleneck and am stuck at 93-94% test accuracy. This is because each model trains for a long time and improvement is not guaranteed. I struggled to increase the test accuracy by barely 0.1%.

Closer to the end of the project, I considered trying CNN in PyTorch. However, since I was getting a poorer performance in NN-like Multilayer Perceptron, I feel that it is better to use the time for data augmentation to get a larger train set. I augment the training data to twice the amount, by adding the same train set but with shifted image by shifting one row, which allows the same character to be kept.

After performing a small grid search of XgBoost with an augmented train set, I was able to reach 96% test accuracy using my own train-test set. However, when using the entire dataset to have predictions submitted to Kaggle, I was only able to improve to 94.2% test accuracy. I was not able to get better parameters to use because I did not have enough time to perform a larger grid search and each training runs for ~10h.

The final classifier that I used is an XgBoost with max_depth of 9, n_estimators of 2000, and learning_rate of 0.2, using an augmented training set.