CS 6375 Machine Learning Project 2 Report: Evaluation of Tree-Based Classifiers and Their Ensembles

Prepared By:
Benjamin Walmer, bjw220002

Introduction:

To begin this project, I uploaded the 45 datasets into a public github repository and imported them into Python in Pandas dataframes for convenient data analysis (I reused a lot of code from Project 1 for importing the files). After successfully importing all files, I began to conduct experiments with the four different models.

Experiments:

I initially attempted to use the GridSearchCV library for hyperparameter tuning. While it was successful, it was difficult to track the progress of each of my models and also made using the validation set for hyperparameter tuning more difficult. Instead, I created for loops to run a manual grid search for each combination of different hyperparameters. This allowed me to track how slow/fast each of my models were running, and also was easier to understand. For each model, I computed the validation accuracy for each combination of hyperparameters. I then retrained the model with the best validation accuracy on the full dataset (training + validation set), and then computed its test accuracy. I repeated this process for all fifteen datasets, and for each of the four different model types.

Hyperparameter Tuning:

For the decision tree model, I tested max_depth values from 3-10, along with all possible criterion and splitter values (criterion being either entropy, gini, or log loss, and splitter being either best or random). As shown in Figure 1, a wide variety of hyperparameters were chosen for the different sized datasets, with varying degrees of accuracy on the test dataset.

```
[c=300, d=100] Best Params: {'max_depth': 4, 'criterion': 'gini', 'splitter': 'best'}, Test Accuracy: 0.6850, F1 Score: 0.7123
[c=300, d=1000] Best Params: {'max_depth': 5, 'criterion': 'gini', 'splitter': 'best'}, Test Accuracy: 0.6725, F1 Score: 0.7095
[c=300, d=5000] Best Params: {'max_depth': 8, 'criterion': 'gini', 'splitter': 'random'}, Test Accuracy: 0.7796, F1 Score: 0.7999
[c=500, d=100] Best Params: {'max_depth': 6, 'criterion': 'entropy', 'splitter': 'best'}, Test Accuracy: 0.6850, F1 Score: 0.6897
[c=500, d=1000] Best Params: {'max_depth': 6, 'criterion': 'entropy', 'splitter': 'best'}, Test Accuracy: 0.7080, F1 Score: 0.7203
[c=500, d=5000] Best Params: {'max_depth': 8, 'criterion': 'entropy', 'splitter': 'best'}, Test Accuracy: 0.7812, F1 Score: 0.8020
[c=1000, d=100] Best Params: {'max_depth': 4, 'criterion': 'gini', 'splitter': 'random'}, Test Accuracy: 0.7300, F1 Score: 0.7477
[c=1000, d=1000] Best Params: {'max_depth': 7, 'criterion': 'entropy', 'splitter': 'random'}, Test Accuracy: 0.8185, F1 Score: 0.8309
[c=1000, d=5000] Best Params: {'max_depth': 10, 'criterion': 'entropy', 'splitter': 'best'}, Test Accuracy: 0.8535, F1 Score: 0.8597
[c=1500, d=100] Best Params: {'max_depth': 3, 'criterion': 'gini', 'splitter': 'random'}, Test Accuracy: 0.8550, F1 Score: 0.8585
[c=1500, d=1000] Best Params: {'max_depth': 6, 'criterion': 'entropy', 'splitter': 'random'}, Test Accuracy: 0.9195, F1 Score: 0.9220
[c=1500, d=5000] Best Params: {'max_depth': 10, 'criterion': 'entropy', 'splitter': 'random'}, Test Accuracy: 0.9547, F1 Score: 0.9551
[c=1800, d=100] Best Params: {'max_depth': 3, 'criterion': 'gini', 'splitter': 'best'}, Test Accuracy: 0.9050, F1 Score: 0.9116
[c=1800, d=1000] Best Params: {'max_depth': 8, 'criterion': 'entropy', 'splitter': 'best'}, Test Accuracy: 0.9765, F1 Score: 0.9768
[c=1800, d=5000] Best Params: {'max_depth': 9, 'criterion': 'entropy', 'splitter': 'best'}, Test Accuracy: 0.9839, F1 Score: 0.9840
```

Figure 1: Hyperparameter Tuning for Decision Tree Classifier

For the bagging classifier, I tested two hyperparameters, n_estimators and max_samples, with estimators of either 100, 150, or 200, and max_samples of 0.5, 0.75, or 1. I tested less combinations of hyperparameters with bagging than the decision tree model because bagging took much longer to run. As shown in Figure 2, different hyperparameters were chosen for each dataset, and most datasets had a pretty high test accuracy with bagging (it outperformed the decision tree model across all datasets).

```
[c=300, d=100] Best Params: {'n_estimators': 150, 'max_samples': 0.75}, Test Accuracy: 0.7700, F1 Score: 0.7700
[c=300, d=1000] Best Params: {'n_estimators': 200, 'max_samples': 0.75}, Test Accuracy: 0.8975, F1 Score: 0.9000
[c=300, d=5000] Best Params: {'n_estimators': 200, 'max_samples': 0.5}, Test Accuracy: 0.9208, F1 Score: 0.9253
[c=500, d=100] Best Params: {'n_estimators': 150, 'max_samples': 0.5}, Test Accuracy: 0.8650, F1 Score: 0.8670
[c=500, d=1000] Best Params: {'n_estimators': 200, 'max_samples': 0.5}, Test Accuracy: 0.9055, F1 Score: 0.9049
[c=500, d=5000] Best Params: {'n_estimators': 200, 'max_samples': 0.5}, Test Accuracy: 0.9427, F1 Score: 0.9433
[c=1000, d=100] Best Params: {'n_estimators': 100, 'max_samples': 0.5}, Test Accuracy: 0.9100, F1 Score: 0.9100
[c=1000, d=1000] Best Params: {'n_estimators': 200, 'max_samples': 0.5}, Test Accuracy: 0.9505, F1 Score: 0.9512
[c=1000, d=5000] Best Params: {'n_estimators': 150, 'max_samples': 0.75}, Test Accuracy: 0.9659, F1 Score: 0.9660
[c=1500, d=100] Best Params: {'n_estimators': 150, 'max_samples': 0.75}, Test Accuracy: 0.9900, F1 Score: 0.9901
[c=1500, d=1000] Best Params: {'n_estimators': 150, 'max_samples': 0.75}, Test Accuracy: 0.9840, F1 Score: 0.9840
[c=1500, d=5000] Best Params: {'n_estimators': 200, 'max_samples': 0.5}, Test Accuracy: 0.9907, F1 Score: 0.9907
[c=1800, d=100] Best Params: {'n_estimators': 100, 'max_samples': 0.75}, Test Accuracy: 0.9850, F1 Score: 0.9849
[c=1800, d=1000] Best Params: {'n_estimators': 200, 'max_samples': 0.75}, Test Accuracy: 0.9930, F1 Score: 0.9930
[c=1800, d=5000] Best Params: {'n_estimators': 200, 'max_samples': 0.75}, Test Accuracy: 0.9977, F1 Score: 0.9977
```

Figure 2: Hyperparameter Tuning for Bagging Classifier

For the random forest classifier, I tested different values for max_depth, n_estimators, and max_features (max_depth = 5, 7, or 9, n_estimators = 100, 150, or 200, and max_features = sqrt or log2). As shown in Figure 3, the random forest classifier ran quickly and produced very impressive results across all datasets, even perfectly understanding the relationship in some datasets where we reached a test accuracy of 1.0.

```
[c=300, d=100] Best Params: {'max_depth': 5, 'n_estimators': 200, 'max_features': 'sqrt'}, Test Accuracy: 0.8450, F1 Score: 0.8488
[c=300, d=1000] Best Params: {'max_depth': 7, 'n_estimators': 200, 'max_features': 'sqrt'}, Test Accuracy: 0.8860, F1 Score: 0.8875
[c=300, d=5000] Best Params: {'max_depth': 9, 'n_estimators': 200, 'max_features': 'log2'}, Test Accuracy: 0.9219, F1 Score: 0.9238
[c=500, d=100] Best Params: {'max_depth': 5, 'n_estimators': 200, 'max_features': 'log2'}, Test Accuracy: 0.8800, F1 Score: 0.8800
[c=500, d=1000] Best Params: {'max_depth': 5, 'n_estimators': 200, 'max_features': 'log2'}, Test Accuracy: 0.9460, F1 Score: 0.9465
[c=500, d=5000] Best Params: {'max_depth': 7, 'n_estimators': 200, 'max_features': 'log2'}, Test Accuracy: 0.9572, F1 Score: 0.9576
[c=1000, d=100] Best Params: {'max_depth': 5, 'n_estimators': 200, 'max_features': 'log2'}, Test Accuracy: 0.9850, F1 Score: 0.9852
[c=1000, d=1000] Best Params: {'max_depth': 7, 'n_estimators': 200, 'max_features': 'log2'}, Test Accuracy: 0.9960, F1 Score: 0.9960
[c=1000, d=5000] Best Params: {'max_depth': 9, 'n_estimators': 200, 'max_features': 'log2'}, Test Accuracy: 0.9972, F1 Score: 0.9972
[c=1500, d=100] Best Params: {'max_depth': 5, 'n_estimators': 100, 'max_features': 'sqrt'}, Test Accuracy: 1.0000, F1 Score: 1.0000
[c=1500, d=1000] Best Params: {'max_depth': 5, 'n_estimators': 100, 'max_features': 'log2'}, Test Accuracy: 1.0000, F1 Score: 1.0000
[c=1500, d=5000] Best Params: {'max_depth': 9, 'n_estimators': 100, 'max_features': 'log2'}, Test Accuracy: 0.9999, F1 Score: 0.9999
[c=1800, d=100] Best Params: {'max_depth': 5, 'n_estimators': 100, 'max_features': 'sqrt'}, Test Accuracy: 1.0000, F1 Score: 1.0000
[c=1800, d=1000] Best Params: {'max_depth': 5, 'n_estimators': 100, 'max_features': 'sqrt'}, Test Accuracy: 1.0000, F1 Score: 1.0000
[c=1800, d=5000] Best Params: {'max_depth': 5, 'n_estimators': 150, 'max_features': 'log2'}, Test Accuracy: 1.0000, F1 Score: 1.0000
```

Figure 3: Hyperparameter Tuning for Random Forest Classifier

For the gradient boosting classifier, I tested learning rates of 0.01, 0.05, and 0.1, n_estimators of 100, 150, and 200, and max_features of log2 or sqrt. As shown in Figure 4, we see the gradient boosting classifier achieved very impressive results across all datasets, even reaching a perfect test accuracy of 1 in some cases. However, it ran in significantly more time than the random forest classifier, taking over an hour to run.

```
[c=300, d=100] Best Params: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 200}, Test Accuracy: 0.8050, F1 Score: 0.8020
[c=300, d=1000] Best Params: {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 200}, Test Accuracy: 0.9945, F1 Score: 0.9945
[c=300, d=5000] Best Params: {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 200}, Test Accuracy: 0.9989, F1 Score: 0.9989
[c=500, d=100] Best Params: {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 200}, Test Accuracy: 0.8700, F1 Score: 0.8762
[c=500, d=1000] Best Params: {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 200}, Test Accuracy: 0.9960, F1 Score: 0.9960
[c=500, d=5000] Best Params: {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 200}, Test Accuracy: 0.9993, F1 Score: 0.9993
[c=1000, d=100] Best Params: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 150}, Test Accuracy: 0.9750, F1 Score: 0.9754
[c=1000, d=1000] Best Params: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200}, Test Accuracy: 0.9960, F1 Score: 0.9960
[c=1000, d=5000] Best Params: {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 200}, Test Accuracy: 0.9999, F1 Score: 0.9999
[c=1500, d=100] Best Params: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 150}, Test Accuracy: 1.0000, F1 Score: 1.0000
[c=1500, d=1000] Best Params: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200}, Test Accuracy: 1.0000, F1 Score: 1.0000
[c=1500, d=5000] Best Params: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 150}, Test Accuracy: 1.0000, F1 Score: 1.0000
[c=1800, d=100] Best Params: {'learning_rate': 0.05, 'max_depth': 3, 'n_estimators': 100}, Test Accuracy: 0.9900, F1 Score: 0.9900
[c=1800, d=1000] Best Params: {'learning_rate': 0.05, 'max_depth': 3, 'n_estimators': 100}, Test Accuracy: 0.9980, F1 Score: 0.9980
[c=1800, d=5000] Best Params: {'learning_rate': 0.05, 'max_depth': 4, 'n_estimators': 150}, Test Accuracy: 0.9999, F1 Score: 0.9999
```

Figure 4: Hyperparameter Tuning for Gradient Boosting Classifier

Comparative Analysis:

Dataset	DecisionTree	Bagging	RandomForest	GradientBoosting
c_300_d_100	0.6850	0.7700	0.8450	0.8050
c_300_d_1000	0.6725	0.8975	0.8860	0.9945
c_300_d_5000	0.7796	0.9208	0.9219	0.9989
c_500_d_100	0.6850	0.8650	0.8800	0.8700
c_500_d_1000	0.7080	0.9055	0.9460	0.9960
c_500_d_5000	0.7812	0.9427	0.9572	0.9993
c_1000_d_100	0.7300	0.9100	0.9850	0.9750
c_1000_d_1000	0.8185	0.9505	0.9960	0.9960
c_1000_d_5000	0.8535	0.9659	0.9972	0.9999
c_1500_d_100	0.8550	0.9900	1.0000	1.0000
c_1500_d_1000	0.9195	0.9840	1.0000	1.0000
c_1500_d_5000	0.9547	0.9907	0.9999	1.0000
c_1800_d_100	0.9050	0.9850	1.0000	0.9900
c_1800_d_1000	0.9765	0.9930	1.0000	0.9980
c_1800_d_5000	0.9839	0.9977	1.0000	0.9999

Dataset	DecisionTree	Bagging	RandomForest	GradientBoosting
c_300_d_100	0.712329	0.770000	0.848780	0.802030
c_300_d_1000	0.709534	0.900049	0.887463	0.994530
c_300_d_5000	0.799855	0.925297	0.923812	0.998901
c_500_d_100	0.689655	0.866995	0.880000	0.876190
c_500_d_1000	0.720307	0.904882	0.946482	0.996016
c_500_d_5000	0.802027	0.943262	0.957641	0.999300
c_1000_d_100	0.747664	0.910000	0.985222	0.975369
c_1000_d_1000	0.830927	0.951207	0.996000	0.996012
c_1000_d_5000	0.859661	0.965951	0.997203	0.999900
c_1500_d_100	0.858537	0.990099	1.000000	1.000000
c_1500_d_1000	0.922034	0.983952	1.000000	1.000000
c_1500_d_5000	0.955135	0.990695	0.999900	1.000000
c_1800_d_100	0.911628	0.984925	1.000000	0.990000
c_1800_d_1000	0.976767	0.992986	1.000000	0.998004
c_1800_d_5000	0.983979	0.997697	1.000000	0.999900

Figure 5: Accuracy (left) and F1 Score (right) across all Datasets and Models

From Figure 5, we see that overall randomforest and gradientboosting have the best overall generalization accuracy/f1 score. Across all datasets they achieve test accuracies and f1-scores of over 0.8, and in the majority of datasets they reach an accuracy of over 99%, which means they are almost perfectly able to model the relationships present in the data. While decision trees and bagging do perform well, randomforest performs better likely due to its random selection of features. This leads to less correlated trees which can capture a wider variety of patterns and variance in the data then bagging or decision trees can. Similarly for gradient boosting, its impressive performance can be attributed to its sequential nature where it learns from previous trees which gives it an advantage over decision trees. For these datasets, randomforest and gradientboosting were able to best model the patterns present in the data, leading to their higher test accuracies and f1-scores.

For datasets with less training data (i.e., with d_100), we generally see lower test accuracies/f1-scores (the lowest test accuracies across all datasets are for c_300_d_100 and c_500_d_100). As training data size increases to 1000, and then 5000, we see much higher test accuracies/f1-scores achieved. This was true across all classifiers, as we can conclude that simply having more data to train on leads to higher test accuracies for each classifier. Similarly for the number of features (clauses), we see that as the number of clauses increases the test accuracy/f1-score improves for all classifiers. We reach the similar conclusion that simply having more features leads to gaining more knowledge about the data and thus leads to better predictive modeling on the test dataset (and higher accuracies/f1-scores).

MNIST Dataset:

I began by importing the MNIST dataset from the openML website. (note: Hopefully there is no issue running the code, but on my end sometimes the openML website connection took a few tries before it successfully imported the dataset). I split the data into the recommended training and test sizes (60k training, 10k test). Additionally, I used train_test_split to randomly split the training data into 12k validation samples for hyperparameter tuning (so now training contains 48k samples). Similar to the previous example, I opted to use a manual for loop instead of GridSearchCV in order to better control my code and understand how it was running. I did the same process as before of finding the optimal hyperparameters using the validation set, then retraining on the full dataset (training + validation) to then get the final accuracy on the test dataset.

Hyperparameter Tuning:

For the decision tree model, I tried out many different hyperparameters for max_depth, criterion, and splitter. Looking at Figure 6, we see that at a max_depth of 12 is when we reach the highest validation accuracy of 0.8772, and that accuracy appears to decrease on either side of 12. Using the optimal hyperparameters we achieved a test accuracy of 0.8833, which is pretty good for a decision tree. Additionally since our test accuracy is higher than our validation accuracy it is unlikely overfitting has occurred as our model generalizes pretty well to the test data.

```
Max Depth: 11, Criterion: gini, Splitter: best, Validation Accuracy: 0.8630
Max Depth: 11, Criterion: gini, Splitter: random, Validation Accuracy: 0.8529
Max Depth: 11, Criterion: entropy, Splitter: best, Validation Accuracy: 0.8707
Max Depth: 11, Criterion: entropy, Splitter: random, Validation Accuracy: 0.8555
Max Depth: 11, Criterion: log_loss, Splitter: best, Validation Accuracy: 0.8707
Max Depth: 11, Criterion: log_loss, Splitter: random, Validation Accuracy: 0.8555
Max Depth: 12, Criterion: gini, Splitter: best, Validation Accuracy: 0.8684
Max Depth: 12, Criterion: gini, Splitter: random, Validation Accuracy: 0.8590
Max Depth: 12, Criterion: entropy, Splitter: best, Validation Accuracy: 0.8772
Max Depth: 12, Criterion: entropy, Splitter: random, Validation Accuracy: 0.8668
Max Depth: 12, Criterion: log_loss, Splitter: best, Validation Accuracy: 0.8772
Max Depth: 12, Criterion: log_loss, Splitter: random, Validation Accuracy: 0.8668
Max Depth: 13, Criterion: gini, Splitter: best, Validation Accuracy: 0.8710
Max Depth: 13, Criterion: gini, Splitter: random, Validation Accuracy: 0.8626
Max Depth: 13, Criterion: entropy, Splitter: best, Validation Accuracy: 0.8756
Max Depth: 13, Criterion: entropy, Splitter: random, Validation Accuracy: 0.8703
Max Depth: 13, Criterion: log_loss, Splitter: best, Validation Accuracy: 0.8756
Max Depth: 13, Criterion: log_loss, Splitter: random, Validation Accuracy: 0.8703
Max Depth: 14, Criterion: gini, Splitter: best, Validation Accuracy: 0.8722
Max Depth: 14, Criterion: gini, Splitter: random, Validation Accuracy: 0.8689
Max Depth: 14, Criterion: entropy, Splitter: best, Validation Accuracy: 0.8744
Max Depth: 14, Criterion: entropy, Splitter: random, Validation Accuracy: 0.8751
Max Depth: 14, Criterion: log_loss, Splitter: best, Validation Accuracy: 0.8744
Max Depth: 14, Criterion: log_loss, Splitter: random, Validation Accuracy: 0.8751
Max Depth: 15, Criterion: gini, Splitter: best, Validation Accuracy: 0.8725
Max Depth: 15, Criterion: gini, Splitter: random, Validation Accuracy: 0.8661
Max Depth: 15, Criterion: entropy, Splitter: best, Validation Accuracy: 0.8765
Max Depth: 15, Criterion: entropy, Splitter: random, Validation Accuracy: 0.8734
Max Depth: 15, Criterion: log_loss, Splitter: best, Validation Accuracy: 0.8765
Max Depth: 15, Criterion: log_loss, Splitter: random, Validation Accuracy: 0.8734
```

```
Best Params: (12, 'entropy', 'best'), Test Accuracy: 0.8833
```

Figure 6: MNIST Dataset: Hyperparameter Tuning for Decision Tree Classifier

For the next three models, I made the decision to hold n_estimators = 100 constant to control runtime. I initially decided to test different values of n_estimators, but because of the vast size of the training data in this dataset it was unfeasible to vary n_estimators, as the runtimes approached 3-4 hours for larger numbers of estimators. For the bagging classifier, I took max_samples as the hyperparameter and tested values of 0.4, 0.5, and 0.6. This varies the size of the random subsets of data used to train each base estimator. Higher values of max_samples were tested but not included in my final code due to high runtimes, but they had lower validation accuracies than the values shown here. From this hyperparameter tuning I found the optimal max_samples value of 0.5, which gave a high test accuracy of 0.9571 (see Figure 7).

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   2 out of   2 | elapsed:  7.7min finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   2 out of   2 | elapsed:   17.4s finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
N Estimators: 100, Max Samples: 0.4, Validation Accuracy: 0.9517
[Parallel(n_jobs=2)]: Done   2 out of   2 | elapsed:  9.9min finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   2 out of   2 | elapsed:   19.7s finished
N Estimators: 100, Max Samples: 0.5, Validation Accuracy: 0.9518
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   2 out of   2 | elapsed: 11.3min finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
N Estimators: 100, Max Samples: 0.6, Validation Accuracy: 0.9509
[Parallel(n_jobs=2)]: Done   2 out of   2 | elapsed:   18.7s finished
```

```
Best Params: (100, 0.5), Test Accuracy: 0.9571
```

Figure 7: MNIST Dataset: Hyperparameter Tuning for Bagging Classifier

For the random forest classifier, holding n_estimators = 100 constant, I took max_depth and max_features as hyperparameters. Testing many values for max_depth, not all of which are shown in this code, I found an optimal max_depth of 22 and max_features of sqrt, which gave a test accuracy of 0.9695.

```
Max Depth: 18, N Estimators: 100, Max Features: sqrt, Validation Accuracy: 0.9643
Max Depth: 18, N Estimators: 100, Max Features: log2, Validation Accuracy: 0.9618
Max Depth: 19, N Estimators: 100, Max Features: sqrt, Validation Accuracy: 0.9653
Max Depth: 19, N Estimators: 100, Max Features: log2, Validation Accuracy: 0.9624
Max Depth: 20, N Estimators: 100, Max Features: sqrt, Validation Accuracy: 0.9657
Max Depth: 20, N Estimators: 100, Max Features: log2, Validation Accuracy: 0.9634
Max Depth: 21, N Estimators: 100, Max Features: sqrt, Validation Accuracy: 0.9663
Max Depth: 21, N Estimators: 100, Max Features: log2, Validation Accuracy: 0.9628
Max Depth: 22, N Estimators: 100, Max Features: sqrt, Validation Accuracy: 0.9665
Max Depth: 22, N Estimators: 100, Max Features: log2, Validation Accuracy: 0.9634
Max Depth: 23, N Estimators: 100, Max Features: sqrt, Validation Accuracy: 0.9663
Max Depth: 23, N Estimators: 100, Max Features: log2, Validation Accuracy: 0.9623
Max Depth: 24, N Estimators: 100, Max Features: sqrt, Validation Accuracy: 0.9662
Max Depth: 24, N Estimators: 100, Max Features: log2, Validation Accuracy: 0.9646
```

```
Best Params: (22, 100, 'sqrt'), Test Accuracy: 0.9695
```

Figure 8: MNIST Dataset: Hyperparameter Tuning for RandomForest Classifier

For the gradientboosting classifier, due to extremely long runtimes, in addition to holding n_estimators = 100 constant I also took learning rate = 0.1 as a constant. From the previous models 0.1 was a very good learning rate that worked well for larger datasets so I figured it would also work well here. I took max depth as a hyperparameter, and found an optimal max_depth of 4 (other values were tested that are not shown in the final code due to runtime restrictions). With n_estimators = 100, learning rate = 0.1, and max_depth = 4, the gradient boosting classifier achieved a test accuracy of 0.9614 (see Figure 9). Note the hyperparameter tuning was not shown for this step because the output was too long, but the model using max_depth = 4 achieved a validation accuracy of 0.9565 which was higher than the model using max_depth = 3 which achieved a validation accuracy of 0.9441.

```
Best Params: (0.1, 100, 4), Test Accuracy: 0.9614
```

Figure 9: MNIST Dataset: Hyperparameter Tuning for GradientBoosting Classifier

MNIST Dataset Comparative Analysis:

DecisionTree	Bagging	RandomForest	GradientBoosting
0.8833	0.9571	0.9695	0.9614

Figure 10: MNIST Dataset: Test Accuracies Across Models

Overall we see that random forest achieves the highest accuracy on the MNIST dataset. While bagging, gradientboosting, and randomforest all had similar/comparable accuracies, randomforest also runs in a significantly shorter amount of time, with models running in a few minutes versus bagging and gradient boosting which take multiple hours to run. Randomforest may have performed slightly better than bagging (as explained in the previous analysis), because randomforest decorrelates trees compared to bagging which can have many correlated trees. This allows the randomforest model to capture more of the patterns present in the data than bagging. Additionally, while gradient boosting did have a comparable accuracy, it may be too complex for this dataset and potentially overfit to noise, while randomforest can provide a diverse group of decorrelated trees which shouldn't be as prone to overfitting. Overall, random forest slightly outperforms gradientboosting and bagging slightly in terms of accuracy, and additionally provides significant advantages in terms of its runtime.