Pranith Mullapudi
PXM220087

Project 2 Tables and analysis

Accuracy:

Dataset	DecisionTree	Bagging	RandomForest	GradientBoosting
c300_d100	0.477387	0.623116	0.577889	0.708543
c300_d1000	0.592796	0.679640	0.786393	0.870935
c300_d5000	0.675668	0.795780	0.846185	0.884088
c500_d100	0.638191	0.603015	0.723618	0.723618
c500_d1000	0.701351	0.804902	0.834917	0.963982
c500_d5000	0.733573	0.870787	0.944194	0.974797
c1000_d100	0.763819	0.874372	0.974874	0.899497
c1000_d1000	0.758379	0.886443	0.959480	0.944972
c1000_d5000	0.812281	0.927193	0.972297	0.993899
c1500_d100	0.783920	0.874372	0.954774	0.954774
c1500_d1000	0.883442	0.964982	0.887944	0.987494
c1500_d5000	0.912891	0.951095	0.905291	0.982698
c1800_d100	0.844221	0.844221	0.909548	0.959799
c1800_d1000	0.915458	0.987994	1.000000	0.990495
c1800_d5000	0.861386	0.882288	0.798680	0.915292

F1 Score

Dataset	DecisionTree	Bagging	RandomForest	GradientBoosting
c300_d100	0.446809	0.623116	0.702128	0.736364
c300_d1000	0.604854	0.668391	0.788718	0.872655
c300_d5000	0.693391	0.815537	0.864470	0.890277
c500_d100	0.647059	0.632558	0.755556	0.712042
c500_d1000	0.724504	0.818267	0.857019	0.964179
c500_d5000	0.744779	0.875169	0.945412	0.975015
c1000_d100	0.758974	0.880383	0.974874	0.900990
c1000_d1000	0.772706	0.894271	0.961001	0.946860
c1000_d5000	0.816717	0.928962	0.973015	0.993927
c1500_d100	0.790244	0.886878	0.956522	0.956522
c1500_d1000	0.889416	0.965585	0.899190	0.987642
c1500_d5000	0.916000	0.952529	0.913476	0.982984
c1800_d100	0.855814	0.855814	0.916667	0.960784
c1800_d1000	0.919867	0.988000	1.000000	0.990580
c1800_d5000	0.876888	0.894392	0.832196	0.921871

Comparative Analysis:
The best classifier by performance overall is tough to say, as all three ensemble methods perform similarly well, especially towards the high end of clauses and datapoints. That being said, i'd choose Gradient boosting as while bagging and random forest do beat it out at times, it has the most consistent performance, especially shining at lower clause/datapoint sets. As for why, it makes sense that all three perform similarly well (with a decent margin above a single tree) since Decision Trees are prone to high variability and overfitting. These ensemble methods do a good job of trying to alleviate that weakness. Since boosting helps with bias, which is mainly a problem with smaller sets, it makes sense that Gradient Boost was the best especially at those lower datasets. As for the performance at larger sets, I think part of that is due to my own implementation where I allowed 250 estimators for both randomforest and gradientboosting. Which ended up being fine for randomforest (since it can be parallelized) but almost unreasonably long for gradient boosting.

Increasing the dataset size improves all of the models across the board since more data helps with counteracting overfitting since it's more likely that the training data will be generalizable. As a result, almost every jump in datapoints corresponded to a higher accuracy and F1 score. With the singular exception being the jump from 1000-5000 for the 1800 clause sets which for some reason performed noticeably worse.

Increasing features similarly also usually led to better performance for all the models. This makes sense, since for Decision Trees, having more features means they have more ways to possibly split the set. With more options, they are more likely to have better options to split on than if there were fewer features. As a result, the trees which are constructed can split the data more accurately and lead to better performance

MNIST dataset:

Model Type	Accuracy
DT	0.881500
Bagging DT	0.966500
Random Forest	0.969800
Gradient Boost (5 estimators)	0.864900

The best performance is very close, but goes to Random Forest. Gradient Boost with enough iterations might be close as well but even just 5 iterations takes several minutes thanks to it not being parallelizable.
It makes sense that both bagging and RF do a good job since they are able to throw a lot of trees at the problem without being too slow since all of the trees can be done in parallel. In addition, random forest adds the random selection element on top of just bagging, allowing it to further reduce variance and improve the generalization. Which while not a massive difference for large datasets like this, still makes a small difference as can be seen with random forest consistently performing a few tenths of a percent better (after running 3-5 times each).