CS 6375: Machine Learning Project 2

Results on synthesized data

1. DecisionTreeClassifier

DATASET	BEST HYPER PARAMETER SETTING (Criterion, Splitter, Max Depth, Min_samples_split, Min_samples_leaf, Max features)	ACCURACY	F1-Score
c300_d100	(gini, random, 2, 2, 1, log2)	0.48	0.504
c300_d1000	(gini, best, 6, 50, 1, None)	0.686	.706
c300_d5000	(entropy, random, 10, 50, 1, None)	0.779	0.786
c500_d100	(entropy, best, 6, 2, 5, None)	0.65	0.667
c500_d1000	(entropy, random, 10, 50, 1, None)	0.715	0.696
c500_d5000	(entropy, random, 10, 50, 1, None)	0.784	0.786
c1000_d100	(gini, random, 6, 2, 2, None)	0.715	0.713
c1000_d5000	(entropy, best, 10, 2, 2, None)	0.858	0.863
c1000_d1000	(entropy, random, 10, 2, 2, None)	0.797	0.806

c1500_d100	(entropy, best, 6, 2, 5, None)	0.885	0.887
c1500_d1000	(entropy, best, 6, 2, 2, None)	0.92	0.922
c1500_d5000	(entropy, best, 10, 2, 2, None)	0.955	0.956
c1800_d100	(gini, best, 6, 2, 5, None)	0.94	0.939
c1800_d1000	(entropy, best, 10, 2, 1, None)	0.972	0.973
c1800_d5000	(gini, best, 10, 2, 2, None)	0.986	0.986

2. DecisionTreeClassifier with BaggingClassifier

Best Parameters used for DecisionTreeClassifier are the ones mentioned in previous table

DATASET	**BEST HYPER PARAMETER SETTING (no of estimators, max_samples, max features)**	**ACCURACY**	**F1-Score**
c300_d100	(100, 1.0, 0.5)	0.755	0.753
c300_d1000	(100, 1.0, 1.0)	0.858	0.863
c500_d5000	(50, 1.0, 0.5)	0.949	0.949
c500_d100	(100, 0.5, 0.5)	0.875	0.8792
c500_d1000	(100, 1.0, 0.5)	0.914	0.914
c300_d5000	(100, 1.0, 0.5)	0.940	0.942

c1000_d100	(100, 1.0, 0.5)	0.95	0.95
c1000_d1000	(100, 0.5, 0.5)	0.984	0.984
c1000_d5000	(100, 0.5, 0.5)	0.989	0.989
c1500_d100	(50, 0.5, 0.5)	0.99	0.99
c1500_d1000	(100, 0.5, 0.5)	0.997	0.997
c1500_d5000	(100, 0.5, 0.5)	0.999	0.999
c1800_d100	(50, 0.5, 0.5)	0.995	0.995
c1800_d1000	(50, 0.5, 0.5)	0.999	0.999
c1800_d5000	(50, 1.0, 0.5)	0.999	0.999

3. RandomForestClassifier

DATASET	**BEST HYPER PARAMETER SETTING (Criterion, Max Depth, Min_samples_split, Min_samples_leaf, Max features, no. of estimators)**	**ACCURACY**	**F1-Score**
c300_d100	(gini, 6, 2, 5, None, 100)	0.82	0.827
c300_d1000	(gini, 10, 2, 5, None, 100)	0.8845	0.887
c300_d5000	(gini, 10, 10, 1, None, 100)	0.903	0.910
c500_d100	(gini, 10, 2, 1, None, 50)	0.82	0.830
c500_d1000	(gini, 10, 2, 1, None, 100)	0.889	0.888

c500_d5000	(gini, 10, 2, 2, None, 100)	0.932	0.934
c1000_d100	(gini, 6, 2, 2, None, 100)	0.89	0.889
c1000_d1000	(gini, 10, 10, 2, None, 100)	0.948	0.948
c1000_d5000	(gini, 10, 2, 2, None, 100)	0.951	0.952
c1500_d100	(gini, 6, 10, 1, None, 100)	0.98	0.98
c1500_d1000	(gini, 10, 2, 2, None, 100)	0.9835	0.9835
c1500_d5000	(entropy, 10, 10, 1, None, 100)	0.9879	0.9879
c1800_d100	(gini, 6, 2, 1, None, 50)	0.97	0.970
c1800_d1000	(gini, 10, 2, 1, None, 50)	0.9925	0.9925
c1800_d5000	(gini, 10, 10, 2, None, 100)	0.996	0.996

4. **GradientBoostingClassifier**

DATASET	**BEST HYPER PARAMETER SETTING (Loss, Learning Rate, Criterion, Max Depth, Min_samples_split, Min_samples_leaf, Max Features, no. of estimators, subsample)**	**ACCURACY**	**F1-Score**
c300_d100	(exponential, 0.1, friedman_mse, 6, 10, 2, None, 100, 0.5)	0.79	0.798

c300_d1000	(log_loss, 0.1, friedman_mse, 6, 2, 5, None, 100, 1.0)	0.9845	0.985
c300_d5000	(log_loss, 0.1, friedman_mse, 6, 2, 1, None, 100, 1.0)	0.9984	0.9984
c500_d100	(log_loss, 0.1, friedman_mse, 6, 10, 2, None, 100, 0.5)	0.885	0.883
c500_d1000	(log_loss, 0.1, friedman_mse, 6, 2, 5, None, 100, 0.5)	0.9875	0.9876
c500_d5000	(log_loss, 0.1, friedman_mse, 6, 2, 1, None, 100, 1.0)	0.9985	0.9985
c1000_d100	(exponential, 0.1, Friedman_mse, 10, 2, 2, None, 100, 0.5)	0.975	0.975
c1000_d1000	(exponential, 0.1, Friedman_mse, 10, 10, 1, None, 100, 0.5)	0.991	0.991
c1000_d5000	(exponential, 0.1, Friedman_mse, 6, 2, 1, None, 100, 0.5)	0.999	0.999
c1500_d100	(exponential, 0.1, squared_error, 6, 2, 2, None, 100, 0.5)	1.0	1.0
c1500_d1000	(exponential, 0.1, friedman_mse, 2, 2, 1, None, 100, 0.5)	0.997	0.997
c1500_d5000	(log_loss, 0.1, friedman_mse, 6, 2, 1, None, 100, 0.5)	1.0	1.0
c1800_d100	(exponential, 0.1,	0.995	0.995

	friedman_mse, 2, 2, 1, None, 50, 0.5)		
c1800_d1000	(exponential, 0.1, friedman_mse, 2, 2, 1, None, 100, 0.5)	1.0	1.0
c1800_d5000	(exponential, 0.1, friedman_mse, 2, 2, 1, None, 100, 0.5)	0.9998	0.9998

Questions

Q1. Which classifier (among the four) yields the best overall generalization accuracy/F1 score? Based on your ML knowledge, why do you think the "classifier" achieved the highest overall accuracy/F1 score.

Answer: Overall, **GradientBoostingClassifier** gave the best accuracy/F1 Score.
The data produced from the random CNF formulae is complex but structured data. The way GradientBoostingClassifier works is it trains gradient trees sequentially to fix the error produced by previous trees, which can lead to better generalization of data with complex relations. The weak-learners in it can learn the patterns which are hard to predict. It also tries to find the sweet-spot in Variance vs Bias, by reducing both bias and variance, the weak learners have high bias and low variance, at each iteration it reduces the bias but tries to maintain the low variance.

The DecisionTreeClassifier is just a single tree trained on the data, that is why it can't generalize well and is prone to overfitting.

Bagging helps immensely with the generalization, as can be verified by the results and it has almost comparable results to the GradientBoosting. The way it works is it learns trees from samples and uses all the results to predict the data, again it can generalize complex structured data well, the small samples of data helps in learning the patterns that are hard to predict. GradientBoostingClassifier is still better as it has the ability to predict the hardest patterns and has other features like regularization.

The reason it performed better than RandomForestClassifier is that the RandomForestClassifier works by adding randomness to trees and predicting the average from them, it doesn't focus on difficult to predict points, like the GradientBoostingClassifier does.

Q2. What is the impact of increasing the amount of training data on the accuracy/F1 scores of each of the four classifiers?

Answer: Accuracy/F1 scores for all models except GradientBoostingClassifier on a dataset with higher number of clauses (>= 1500 increases with increase in data points. GradientBoostingClassifier already has perfect or near perfect accuracies/F1 Scores on dataset with 1500/1800 clauses, that is why it is not increasing. It could be that the model couldn't establish a very complex pattern for a very few data points. The difference is merely .01-0.3%, so it is wrong to say that more data would not help it in even more complex models.

Q3. What is the impact of increasing the number of features on the accuracy/F1 scores of each of the four classifiers?

Answer: The accuracy and F1 scores improve with the increase in number of features, if the features are relevant. Otherwise, except RandomForestClassifier, all the models are prone to overfitting the irrelevant features and the scores might decrease in that case.

Results on MNIST Dataset

MODEL	ACCURACY
DecisionTreeClassifier	.8498
DTClassifier with Bagging	.9086
RandomForestClassifier	.9389
GradientBoostingClassifier	.9764

Again, GradientBoostingClassifier gives the best accuracy on the dataset. The reason is the same, the data consists of handwritten digits. A handwritten digit is a complex data, we can recognize a handwritten digit by seeing from the eyes but might not be able to tell the features. The features could be like how many circle-like shapes are in the top and bottom half of the image, how many triangles like, how many 'U' like shapes etc. Just as mentioned previously, By design

GradientBoostingClassifier is able to establish these complex patterns better by using weak classifiers and iteratively learns over them. It also tries to find the sweet-spot in Variance vs Bias, by reducing both bias and variance, the weak learners have high bias and low variance, at each iteration it reduces the bias but tries to maintain the low variance.

Third-Party libraries

a. **Scikit-learn**: It was used for the Classifiers. Source: https://scikit-learn.org/stable/
b. **Scipy**: It was used for random and uniform functions. Source: https://scipy.org/
C. **Pandas:** Used to load data
https://pandas.pydata.org/