Name: Rhugaved Narmade
Net ID: rrn210004
CS6375 Assignment 3

Google Collab Link:
https://colab.research.google.com/drive/1pnCKjhX2KI1tm3mkvbY3kC6nnb72D1mN?usp=sharing

1. Tree Classifiers

Code Explanation:

1. Read the CSV files in train, test, and tune sets
2. We have 4 functions to perform DecisionTreeClassifier, BaggingClassifier, RandomForestClassifier, and GradientBoostingClassifier
3. In each function, we have lists of parameters we want to tune on and we train on every combination of those parameters and check their accuracy on the tune set
4. Next, we merge train and tune sets and train the best parameter setting.
5. Finally, we test the complete trained model on the test set and report the Accuracy and F1 Score

1. DecisionTreeClassifier:
    a. We tune on the following parameters:
        i. Criterion
        ii. Splitter
        iii. Max_depth
        iv. Max_features

    Best parameter tuning:

	criterion	splitter	max_depth	max_features
c1000_d5000	gini	random	7	auto
c1000_d1000	entropy	best	7	auto
c1000_d100	gini	random	5	auto
c1500_d5000	gini	best	15	auto
c1500_d1000	entropy	best	7	auto
c1500_d100	gini	best	5	auto
c1800_d5000	entropy	random	15	auto
c1800_d1000	entropy	random	15	log2
c1800_d100	gini	random	7	log2

c300_d5000	gini	best	7	auto
c300_d1000	gini	best	5	auto
c300_d100	gini	random	2	log2
c500_d5000	entropy	best	7	auto
c500_d1000	entropy	best	7	auto
c500_d100	gini	best	5	log2

2. BaggingClassifier:
    a. We tune on the following parameters:
        i. Criterion
        ii. Splitter
        iii. Max_depth
        iv. Max_feaures
        v. N_estimators

    Best parameter tuning:

	criterion	splitter	max_depth	max_features	n_estimators
c1000_d5000	gini	best	15	log2	500
c1000_d1000	gini	best	15	log2	500
c1000_d100	gini	best	10	log2	500
c1500_d5000	gini	best	2	log2	500
c1500_d1000	gini	best	2	log2	500
c1500_d100	gini	best	2	auto	100
c1800_d5000	gini	best	2	log2	100
c1800_d1000	gini	best	2	auto	500
c1800_d100	gini	best	2	auto	50
c300_d5000	entropy	best	10	log2	500
c300_d1000	entropy	best	5	log2	500
c300_d100	gini	random	7	log2	500
c500_d5000	entropy	best	10	log2	500
c500_d1000	entropy	random	7	log2	500

c500_d100	gini	best	2	log2	500

3. RandomForestClassifier:
    a. We tune on the following parameters:
        i. Criterion
        ii. Max_depth
        iii. Max_feaures
        iv. N_estimators

Best parameter tuning:

	criterion	n_estimators	max_depth	max_features
c1000_d5000	entropy	500	20	log2
c1000_d1000	gini	500	15	log2
c1000_d100	gini	500	5	log2
c1500_d5000	entropy	100	10	log2
c1500_d1000	gini	50	5	log2
c1500_d100	gini	50	2	auto
c1800_d5000	gini	50	5	log2
c1800_d1000	gini	50	2	auto
c1800_d100	gini	10	5	auto
c300_d5000	entropy	500	20	auto
c300_d1000	entropy	500	10	auto
c300_d100	entropy	500	10	auto
c500_d5000	gini	500	20	log2
c500_d1000	gini	500	5	log2
c500_d100	gini	500	2	log2

4. GradientBoostingClassifier:
    a. We tune on the following parameters:
        i. Loss
        ii. Learning rate
        iii. N_estimators
        iv. Criterion

Best parameter tuning:

	loss	lr	n_estimators	criterion
c1000_d5000	deviance	0.1	500	friedman_mse
c1000_d1000	exponential	0.1	500	friedman_mse
c1000_d100	deviance	0.1	500	friedman_mse
c1500_d5000	deviance	0.1	500	friedman_mse
c1500_d1000	deviance	0.1	500	friedman_mse
c1500_d100	deviance	0.1	500	friedman_mse
c1800_d5000	deviance	0.1	500	friedman_mse
c1800_d1000	deviance	0.1	100	squared_error
c1800_d100	deviance	0.1	500	friedman_mse
c300_d5000	deviance	0.1	500	friedman_mse
c300_d1000	deviance	0.1	500	friedman_mse
c300_d100	deviance	0.1	500	friedman_mse
c500_d5000	deviance	0.1	500	friedman_mse
c500_d1000	deviance	0.1	500	friedman_mse
c500_d100	exponential	0.1	500	friedman_mse

Accuracy and F1 Score of the best parameters for each dataset on each model on the test set trained using Train+Tune sets:

c1000_d5000
DT>>>) Accuracy -> 0.7939 &&& F1 Score -> 0.7943319030036923
BAGG>>>) Accuracy -> 0.9988 &&& F1 Score -> 0.998800719568259
RF>>>) Accuracy -> 0.999 &&& F1 Score -> 0.999
GB>>>) Accuracy -> 1.0 &&& F1 Score -> 1.0

c1000_d1000
DT>>>) Accuracy -> 0.78 &&& F1 Score -> 0.7920604914933836
BAGG>>>) Accuracy -> 0.998 &&& F1 Score -> 0.998001998001998
RF>>>) Accuracy -> 0.998 &&& F1 Score -> 0.998001998001998
GB>>>) Accuracy -> 0.9985 &&& F1 Score -> 0.9985022466300548

c1000_d100
DT>>>) Accuracy -> 0.68 &&& F1 Score -> 0.6701030927835052
BAGG>>>) Accuracy -> 1.0 &&& F1 Score -> 1.0
RF>>>) Accuracy -> 1.0 &&& F1 Score -> 1.0
GB>>>) Accuracy -> 0.98 &&& F1 Score -> 0.98

c1500_d5000
DT>>>) Accuracy -> 0.9185 &&& F1 Score -> 0.9191066997518611
BAGG>>>) Accuracy -> 0.9995 &&& F1 Score -> 0.9994999499949995
RF>>>) Accuracy -> 0.9999 &&& F1 Score -> 0.9998999899989999
GB>>>) Accuracy -> 1.0 &&& F1 Score -> 1.0

c1500_d1000
DT>>>) Accuracy -> 0.884 &&& F1 Score -> 0.8872691933916423
BAGG>>>) Accuracy -> 1.0 &&& F1 Score -> 1.0
RF>>>) Accuracy -> 1.0 &&& F1 Score -> 1.0
GB>>>) Accuracy -> 1.0 &&& F1 Score -> 1.0

c1500_d100
DT>>>) Accuracy -> 0.825 &&& F1 Score -> 0.8325358851674641
BAGG>>>) Accuracy -> 1.0 &&& F1 Score -> 1.0
RF>>>) Accuracy -> 1.0 &&& F1 Score -> 1.0
GB>>>) Accuracy -> 1.0 &&& F1 Score -> 1.0

c1800_d5000
DT>>>) Accuracy -> 0.9671 &&& F1 Score -> 0.9673384294649061
BAGG>>>) Accuracy -> 1.0 &&& F1 Score -> 1.0
RF>>>) Accuracy -> 1.0 &&& F1 Score -> 1.0
GB>>>) Accuracy -> 1.0 &&& F1 Score -> 1.0

c1800_d1000
DT>>>) Accuracy -> 0.929 &&& F1 Score -> 0.9298418972332017
BAGG>>>) Accuracy -> 0.9995 &&& F1 Score -> 0.9995002498750626
RF>>>) Accuracy -> 0.999 &&& F1 Score -> 0.9990009990009989
GB>>>) Accuracy -> 1.0 &&& F1 Score -> 1.0

c1800_d100
DT>>>) Accuracy -> 0.885 &&& F1 Score -> 0.8844221105527638
BAGG>>>) Accuracy -> 1.0 &&& F1 Score -> 1.0
RF>>>) Accuracy -> 1.0 &&& F1 Score -> 1.0
GB>>>) Accuracy -> 0.995 &&& F1 Score -> 0.9950248756218906

c300_d5000
DT>>>) Accuracy -> 0.6505 &&& F1 Score -> 0.6629376024688977
BAGG>>>) Accuracy -> 0.9375 &&& F1 Score -> 0.938958882703389
RF>>>) Accuracy -> 0.9335 &&& F1 Score -> 0.9346822512523327
GB>>>) Accuracy -> 0.9991 &&& F1 Score -> 0.9991008092716556

c300_d1000
DT>>>) Accuracy -> 0.6075 &&& F1 Score -> 0.611578426521524
BAGG>>>) Accuracy -> 0.9005 &&& F1 Score -> 0.9018253576714357
RF>>>) Accuracy -> 0.9015 &&& F1 Score -> 0.9032891507118311
GB>>>) Accuracy -> 0.996 &&& F1 Score -> 0.9960159362549801

c300_d100
DT>>>) Accuracy -> 0.48 &&& F1 Score -> 0.5047619047619047
BAGG>>>) Accuracy -> 0.855 &&& F1 Score -> 0.8585365853658538
RF>>>) Accuracy -> 0.855 &&& F1 Score -> 0.8585365853658538
GB>>>) Accuracy -> 0.84 &&& F1 Score -> 0.8431372549019608

c500_d5000
DT>>>) Accuracy -> 0.6943 &&& F1 Score -> 0.7081622911694511
BAGG>>>) Accuracy -> 0.9732 &&& F1 Score -> 0.973559589581689
RF>>>) Accuracy -> 0.9726 &&& F1 Score -> 0.9728390166534497
GB>>>) Accuracy -> 0.9996 &&& F1 Score -> 0.9996001599360256

c500_d1000
DT>>>) Accuracy -> 0.639 &&& F1 Score -> 0.6535508637236084
BAGG>>>) Accuracy -> 0.968 &&& F1 Score -> 0.9683794466403162
RF>>>) Accuracy -> 0.962 &&& F1 Score -> 0.9622641509433962
GB>>>) Accuracy -> 0.9975 &&& F1 Score -> 0.9975062344139651

c500_d100

DT>>>) Accuracy -> 0.625 &&& F1 Score -> 0.6543778801843316
BAGG>>>) Accuracy -> 0.93 &&& F1 Score -> 0.9285714285714285
RF>>>) Accuracy -> 0.905 &&& F1 Score -> 0.9025641025641025
GB>>>) Accuracy -> 0.895 &&& F1 Score -> 0.8975609756097561

Answer:

1. Overall Gradient Boosting performed the best with some exceptions. Boosting does so well because it's an ensemble method, so it combines several different classifers which learn different features from the data. GB improves trees sequentially by using the previous tree. It tries to reduce the variance by increasing the weight for incorrectly classified examples. It then does things like gradient descent i.e, it moves in the toward directions that get the model closer to the target value. Compared to random forests, it uses shallow trees, so it generalizes better. And generally, ensembles are better, so it performs better than Decision trees and boosting does better than bagging too on stable data.
2. Increasing the data generally increases the accuracy and F1 scores with some exceptions
3. Increasing the features(i.e clauses) too increases the accuracy and F1 scores

Extra Credit (MNIST):

Running all MNIST on all 4 classifiers gives the following accuracies:

DT>>>) Accuracy -> 0.5325

BAGG>>>) Accuracy -> 0.6875

RF>>>) Accuracy -> 0.6515

GB>>>) Accuracy -> 0.912

Thus, GradientBoostingClassifier is the best classifier as it yields the best accuracy although it is considerably slow compared to others

Boosting does so well because it's an ensemble method, so it combines several different classifiers which learn different features from the data. GB improves trees sequentially by using the previous tree. It tries to reduce the variance by increasing the weight for incorrectly classified examples. It then does things like gradient descent i.e, it moves toward directions that get the model closer to the target value. Compared to random forests, it uses shallow trees, so it generalizes better. And generally, ensembles are better, so it performs better than Decision trees and boosting does better than bagging too on stable data.

2. K-means clustering on images

Code Explanation:

1. Read the image
2. In the driver function, I replicate the java code given with the homework with some changes
3. Used NumPy for making use of vectorized operations
4. Flatten out the image keeping the RGB colors' distinct values. Eg. [[r,gb], [r,g,b]....]
5. In kmeans function, for every k value, we first randomly initailize starting positions
6. Next, using vectorized operations to find the distance of every pixel from the means, we change the clusters for each pixel and then recalculate the means.
7. Return a list of assignments of pixels to clusters and the final k means
8. Remake the image properly
9. This was run three times for every value of k for each image

Answers to questions:

1. PFA attached single images for each k and each input image
2. Compression Rations for k values for each image repeated 3 times:

{'Koala.jpg': {2: [4.273865757338573, 4.273865757338573, 4.261852260198457],
5: [3.976143070287556, 3.9531143208638992, 3.98861389931806],
10: [4.5480443836095175, 4.542963863693222, 4.553189380200711],
15: [4.757567449001974, 4.776075308279506, 4.69393263560346],
20: [4.806267350317922, 4.607623993154928, 4.677035040431267]},
'Penguins.jpg': {2: [8.209080451278588, 8.209080451278588, 8.209080451278588],
5: [7.154150379397563, 7.025878421100171, 7.107410453216374],
10: [6.543301787592008, 6.474081534133471, 6.470903872551059],
15: [6.478341259463466, 6.716823253082795, 6.724895171400164],
20: [6.762371331200445, 6.716591254490191, 6.7120704830609395]}}

Mean and Variance of compression ratios:

Image = Koala.jpg, k = 2: MEAN = 4.269861258291868 and VARIANCE = 3.207202523012911e-05
Image = Koala.jpg, k = 5: MEAN = 3.972623763489839 and VARIANCE = 0.00021622943857208332
Image = Koala.jpg, k = 10: MEAN = 4.5480658758344825 and VARIANCE = 1.7427095598686913e-05
Image = Koala.jpg, k = 15: MEAN = 4.7425251309616465 and VARIANCE = 0.0012377054450738668
Image = Koala.jpg, k = 20: MEAN = 4.696975461301373 and VARIANCE = 0.006775340749735931

Image = Penguins.jpg, k = 2: MEAN = 8.209080451278588 and VARIANCE = 0.0
Image = Penguins.jpg, k = 5: MEAN = 7.095813084571369 and VARIANCE = 0.002809532027318694
Image = Penguins.jpg, k = 10: MEAN = 6.4960957314255126 and VARIANCE = 0.0011158887915858884
Image = Penguins.jpg, k = 15: MEAN = 6.640019894648809 and VARIANCE = 0.013080849848584574
Image = Penguins.jpg, k = 20: MEAN = 6.730344356250524 and VARIANCE = 0.000516269791273982

3. Yes, there is a tradeoff between image quality and degree of compression although the degree of compression doesn't change much for a better-quality image.

   In the case of the Koala Image:

   The degree of compression doesn't decrease with an increase in K, i.e it roughly remains similar around 4-5. That means, for a higher quality image, we get good compression. The good value of K for this would be 20, as the degree of compression is 4.69 and the quality of the image is the best

   In the case of the Penguins Image:

   The degree of compression decreases with an increase in K, but the decrease is not significant (from 8.2 to 6.7). For higher quality images, i.e with higher K, the compression is less, but not significantly less compared to smaller values of K. Depending on user requirements, K can be either as small as 2 or as large as 20. But I feel the best results of both worlds are achieved with K=20, although K=5 is good too

   The reason for the improper trend in the degree of compression and K values may be due to the JPG format used.