COM S 474/574 Spring 2020
Introduction to Machine Learning

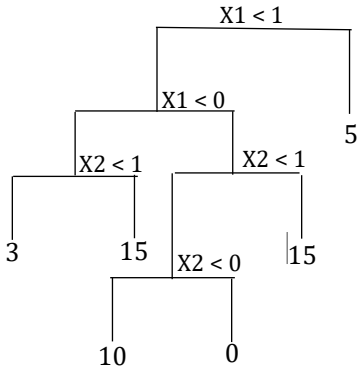
Homework 4

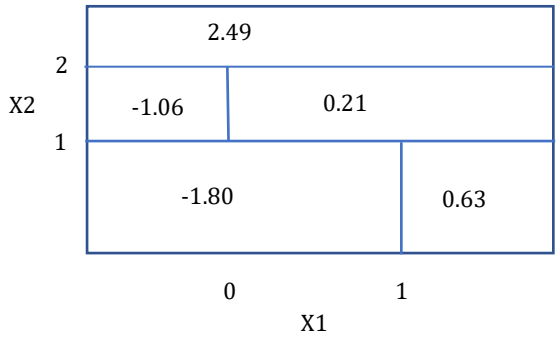
1 Directions:

- **Due: Thursday April 2, 2020 at 10pm.** Late submissions will be accepted for 24 hours after that time, with a 15% penalty.
- Upload the homework to Canvas as a pdf file. Answers to problems 1-3 can be handwritten, but writing must be neat and the scan should be high-quality image. Other responses should be typed or computer generated.
- Any non-administrative questions must be asked in office hours or (if a brief response is sufficient) Piazza.

2 Problems

Problem 1. [10 points] Ch. 8 #4 "This question relates ..."





Problem 2. [5 points] Ch. 8 #5 "Suppose we produce ..."

Majority vote approach:

Red classification. (6/10) probabilities said red was most likely

Average probability:

Average probability = 0.45. This would give a green classification.

Problem 3. [15 points] Suppose we construct a decision tree (shown below) for the data set in the following table, using the Gini index to select which features to split on.

Fill in the missing values, including the branches (what feature is split on, such as X_{male}), the Gini index at each node, the number of samples at each node with $Y = 0$ and $Y = 1$, and the predictions at the leaf nodes.

Show your calculations. Also report the training accuracy.

Y=1=Yes X=1=Yes

Y=0=No X=0=No

Gini= (4/6)(2/6))*2 = 0.4444
2 $Y = 0$, 4 $Y = 1$

X Chest Pain =0 | X Chest Pain =1

Gini= (1/3)(2/3)+(2/3)(1/3) = 0.4444
2 $Y = 0$, 1 $Y = 1$

Y_b = No

Gini= (3/3)(0/3)+(0/3)(3/3) = 0
0 $Y = 0$, 3 $Y = 1$

Y_b = Yes

(3/6)*0.4444 + (3/6)*0 = 0.2222

Training accuracy for Yb = Yes is 3/3*100 = 100%
Training accuracy for Yb = No is 2/3*100 = 66.67%

Gini= (4/6)(2/6))*2 = 0.4444
2 $Y = 0$, 4 $Y = 1$

X Male =0 | X Male =1

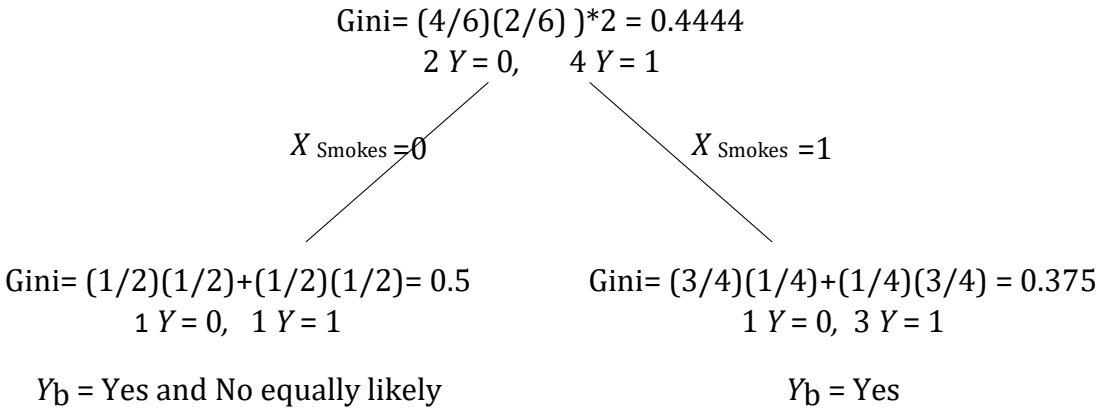
Gini= (2/2)(0/2)+(0/2)(2/2) = 0
0 $Y = 0$, 2 $Y = 1$

Gini= (2/4)(2/4)+(2/4)(2/4) = 0.5
2 $Y = 0$, 2 $Y = 1$

Y_b = Yes

Y_b = Yes and No equally likely

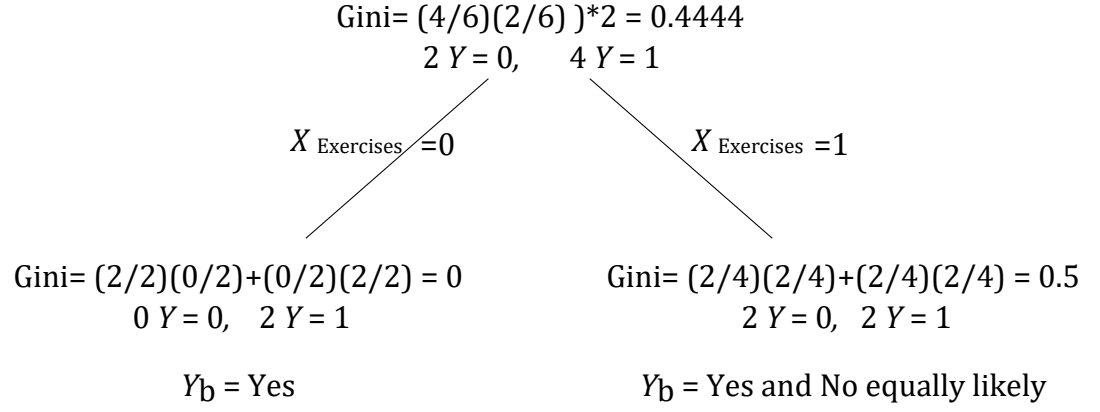
(2/6)*0+(4/6)*0.5 = 0.3333



Y_b = Yes and No equally likely

Y_b = Yes

(2/6)*0.5+(4/6)*0.375 = 0.4167



Y_b = Yes

Y_b = Yes and No equally likely

(2/6)*0+(4/6)*0.5 = 0.3333

PATIENT ID	CHEST PAIN?	MALE?	SMOKES?	EXERCISES?	HEART ATTACK?
1.	yes	yes	no	yes	yes
2.	yes	yes	yes	no	yes

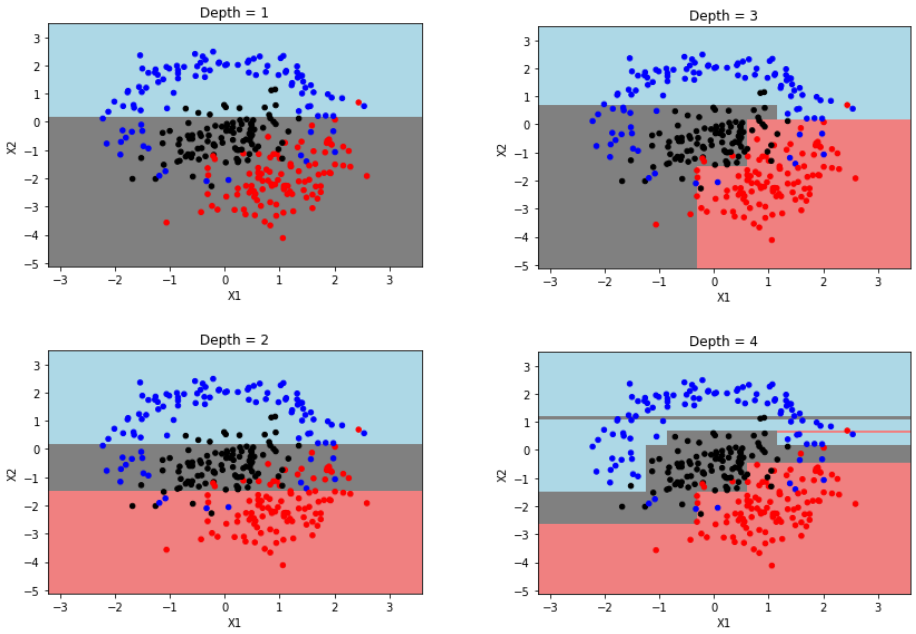
3.	no	no	yes	no	yes
4.	no	yes	no	yes	no
5.	yes	no	yes	yes	yes
6.	no	yes	yes	yes	no

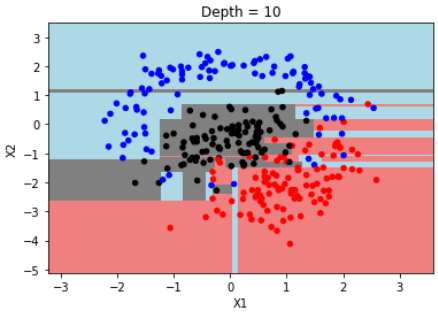
Compared to the other features, the weighed average of the child nodes for chest pain gives the lowest Gini index of 0.2222. Therefore chest pain would give the best split for predicting heart attack.

Problem 4. [40 points] This problem uses the same data sets as homework 3. You should be able to re-use code from homework 3, with only a few modifications. We will compare boosting, bagging, and a single decision tree. If you are using Python, you can call from sklearn.tree import DecisionTreeClassifier from sklearn.ensemble import RandomForestClassifier from sklearn.ensemble import GradientBoostingClassifier

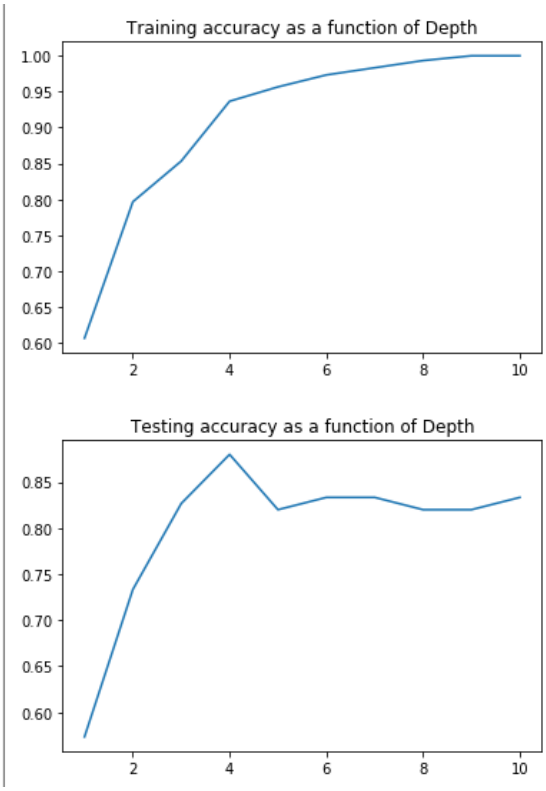
A. Make separate scatter plots of the training (only training, not testing) data like in homework 3, with the decision tree classification function plotted using colors. Make plots for max_depths of 1, 2, 3, 4, and 10. In Python, you can call

clf=DecisionTreeClassifier(criterion='gini',splitter='best',max_depth= depth)





B. Make plots of testing and training accuracies as a function of the depth, with the depth ranging from 1 to 10. Report the best depth.



The best testing accuracy was 0.88 at depth 4.

Remark: We are setting a parameter for the maximum depth, but the trees may stop growing before reaching that limit. Your plots are based on the maximum depth parameter.

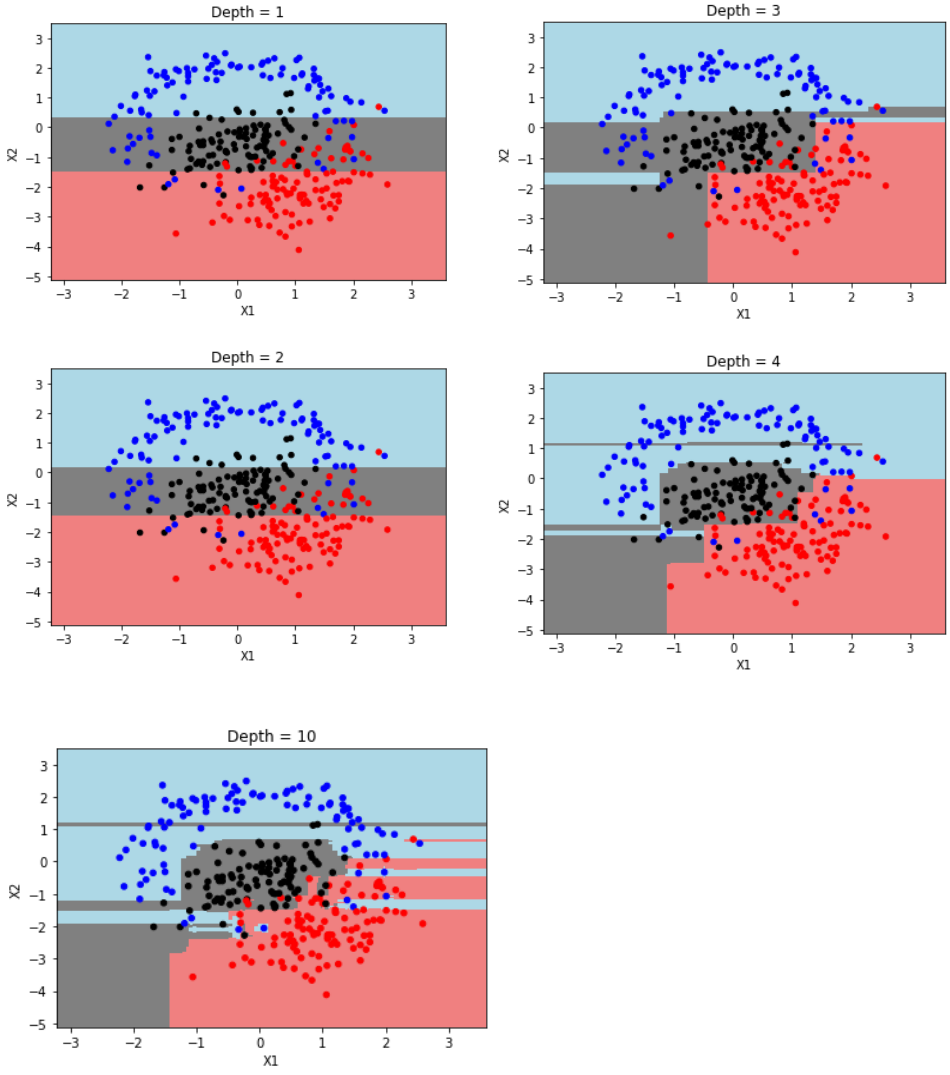
C. Next we will do bagging, where we randomly sample from the original data, fit a tree to that data, repeat, and then combine the predictions of all the trees. In Python, you can call

```
clf=RandomForestClassifier(bootstrap=True,n_estimators=num_trees,max_
    features=None,criterion='gini',max_depth=depth)
```

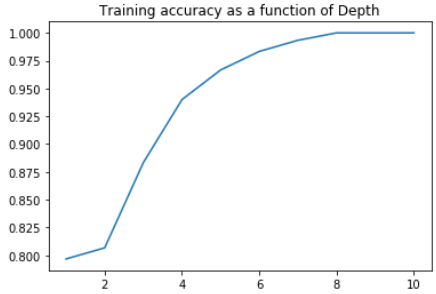
Make separate scatter plots of the training data (only training, not testing) like in homework 3, with the decision tree classification function (fit using training data) plotted using colors. Make plots for max_depths of 1, 2, 3, 4, and 10. For each depth, use the best num_trees value for that particular depth, from the set

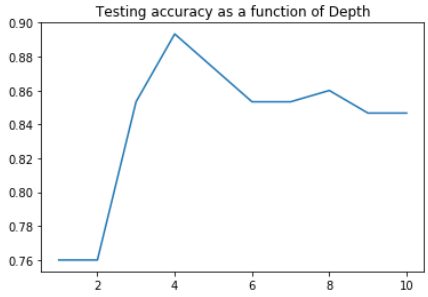
numtreerange=[1,5,10,25,50,100,200].

Remark: even though the function is called RandomForestClassifier, setting max_features=None results in allowing each tree to use any of the features (all two of them), which is just bagging.



D. Also make plots of testing and training accuracies as a function of the depth (using the best num_trees value for that depth), with the depth ranging from 1 to 10. Report the best depth and corresponding number of trees.





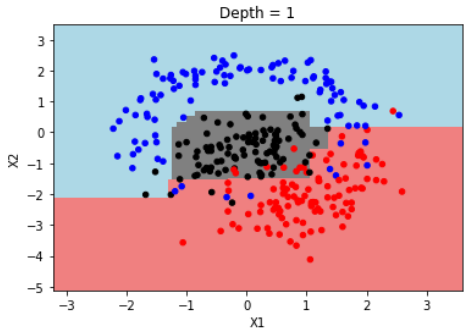
The best testing accuracy was 0.8933 at depth 4. This corresponded to 1 tree

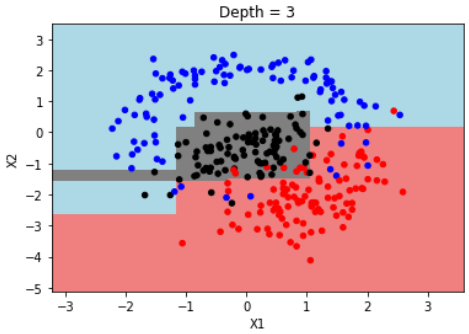
E. Next we will do boosting, where we adaptively build trees to learn from the previous trees' mistakes. In Python, you can call

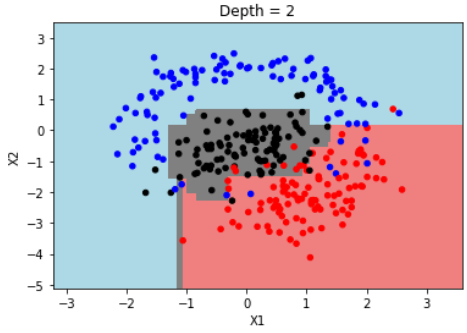
```
clf=GradientBoostingClassifier(learning_rate=rate,n_estimators=num_
trees,max_depth=depth)
```

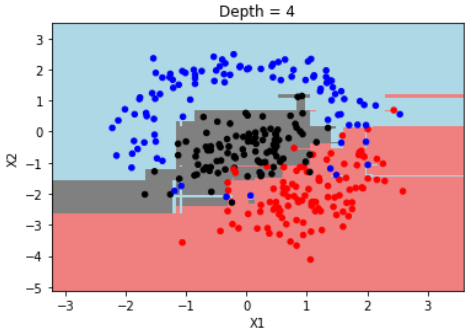
Make separate scatter plots of the training data (only training, not testing) like in HW 3, with the decision tree classification function (fit using training data) plotted using colors. Make plots for max_depths of 1, 2, 3, 4, and 10. For each depth, use the best number of trees and the best learning rate for that particular depth, from the sets

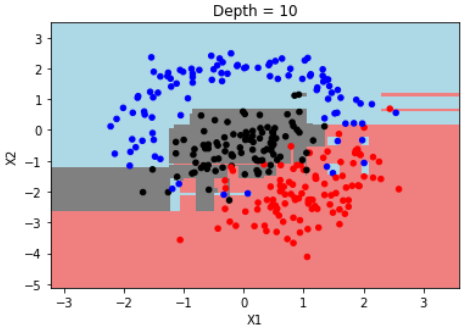
```
numtreerange=[1,5,10,25,50,100,200].
learnraterange=np.logspace(-3,0,15,base=10)
```



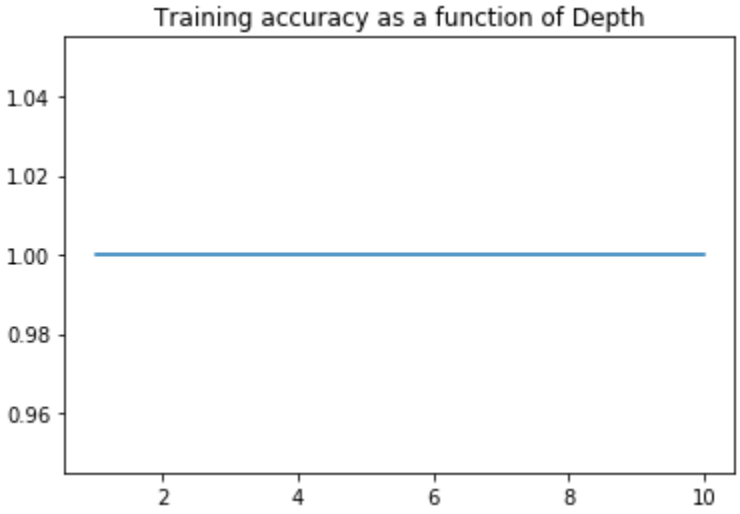


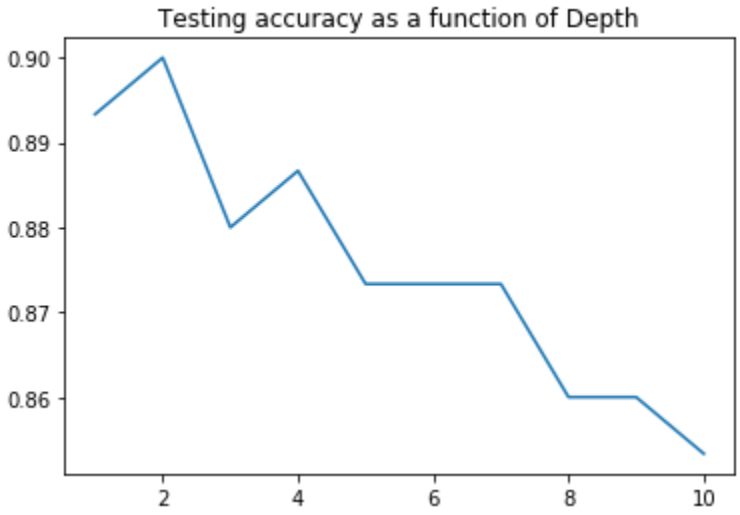






F. Also make plots of testing and training accuracies as a function of the depth (use the best number of trees and the best learning rate for that particular depth), with the depth ranging from 1 to 10. Report the best depth and num_trees value.





The best testing accuracy was 0.90 at depth 2. This corresponded to 200 trees

G. In about 4-6 sentences, comment on how the decision boundaries differ between single trees, bagged trees, and boosted trees, how the depth affects (or not) the performance, and how the regions and performance compare to methods from homework 3.

The single tree gave the worst testing accuracy while boosting gave the best. In the single tree and bagging methods, the testing data was better fit as depth increased up to a depth of 4, then the testing accuracy would decease and flatline(not change much) as depth continued to increase. In the bosting method, overfitting occurred after a depth of 2 causing the testing accuracy to fall. The decision boundaries in the boosting and bagging methods are better than that in the single decision tree method. These methods and the

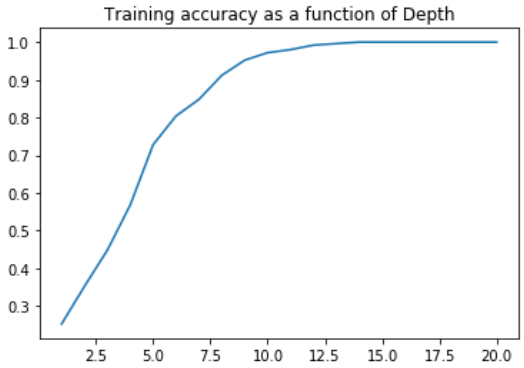
ones in HW3 all fit the data better as the models become more complex up to a certain point.

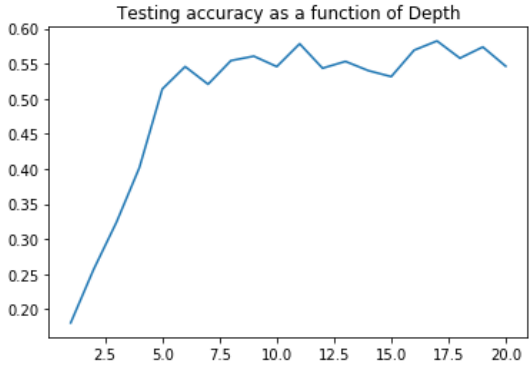
Problem 5. [30 points] The previous problem allowed us to visualize the prediction function since we only had two features. Now let's explore what happens on a data set with more features. You should be able to use most of the same code as in the previous problem.

Download the files digits-train.csv and digits-test.csv. These come from a digit recognition data-set from https://archive.ics.uci.edu/ml/datasets/Multiple+Features. The first column in both files is the digit ('0', '1', ..., '9') that we want to predict from the other features.

Note, for this data set, we will use most of the samples for testing. (This will allow us to see a bigger contrast between the methods and have an accurate sense of how well the methods will do on new data.)

A. Make plots of testing and training accuracies of a single decision tree, as a function of the depth, with the depth ranging from 1 to 20. Report the best depth.





Best depth was 17 at a testing accuracy of 0.5823

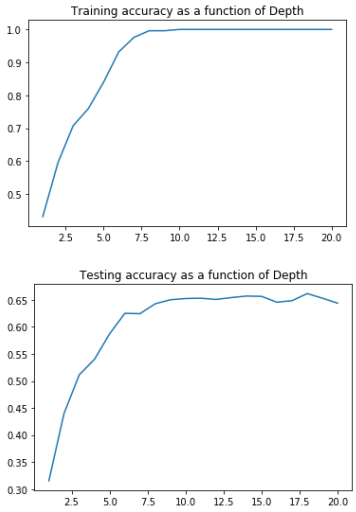
B. Make plots of testing and training accuracies of bagging as a function of the depth (using the best num_trees value for that depth), with the depth ranging from 1 to 20. Report the best depth and corresponding number of trees.

Like in the previous problem, use

clf=RandomForestClassifier(bootstrap=True,n_estimators=num_trees,max_
features=None,criterion='gini',max_depth=depth)

For each depth, use the best number of trees for that particular depth, from the set

numtreerange=[1,5,10,25,50,100,200].



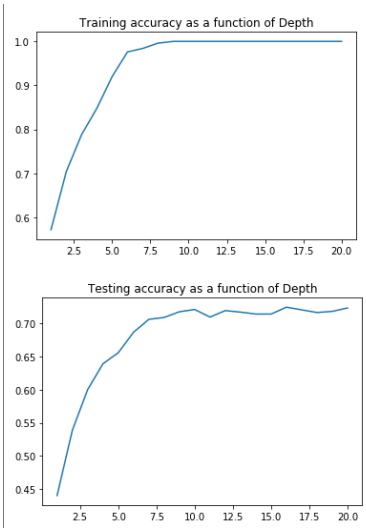
The best testing accuracy was 0.6617 at depth 18. This corresponded to 50 trees

C. Next we will do random forests. Make plots of testing and training accuracies of random forests as a function of the depth (using the best num_trees value for that depth), with the depth ranging from 1 to 20. Report the best depth and corresponding number of trees. Use

clf=RandomForestClassifier(bootstrap=True,n_estimators=num_trees,max_
features='sqrt',criterion='gini',max_depth=depth)

The argument max_features='sqrt' results in a random, small subset of features being considered for splits in each tree. For each depth, use the best number of trees for that particular depth, from the sets

numtreerange=[1,5,10,25,50,100,200].



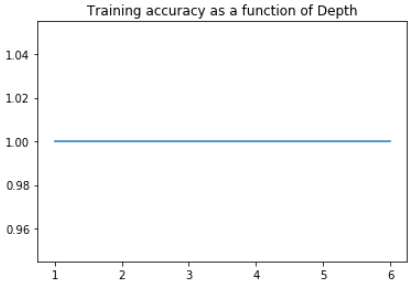
The best testing accuracy was 0.724 at depth 16. This corresponded to 200 trees

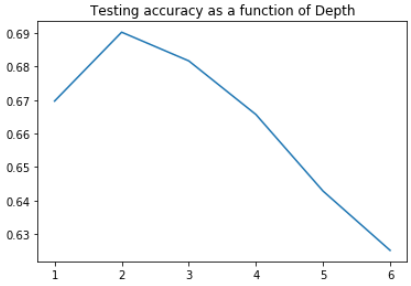
D. Make plots of testing and training accuracies for boosting as a function of the depth (use the best number of trees and the best learning rate for that particular depth). Report the best depth and num_trees value. Use

clf=GradientBoostingClassifier(learning_rate=rate,n_estimators=num_ trees,max_depth=depth)

For each depth, use the best number of trees and the best learning rate for that particular depth, from the sets

depthrange=range(1,6)
numtreerange=[50,100,150]. learnraterange=np.logspace(-2,0,10,base=10)





The best testing accuracy was 0.6903 at depth 2. This corresponded to 10 trees

E. In about 3-5 sentences, comment on how the performance of the different methods and how they varied as the maximum depth changed. (Although you do not have to hand it in, if you re-plot the training accuracy curves of all methods together, and re-plot the testing accuracy curves of all methods together, this may help your analysis.) Comment also on if there were any noticeable differences in how long each took to complete.

The Boosting took the longest to complete while the others were much faster. In the boosting testing accuracy plot, overfitting is evident after depth 2. Out of all these plots, random forests gave the best testing accuracy of 0.724 the lowest was the single decision tree which gave a testing accuracy of 0. 5823. In the single decision tree, bagging and random forest, the training and testing accuracies flatlined at a certain depth and did not change much as depth increased. In the Boosting plot, the accuracy in the training data flatlined right a way and did not change as depth increased, while the testing data accuracy increased up to a depth of 2, then decreased dramatically as depth increased.