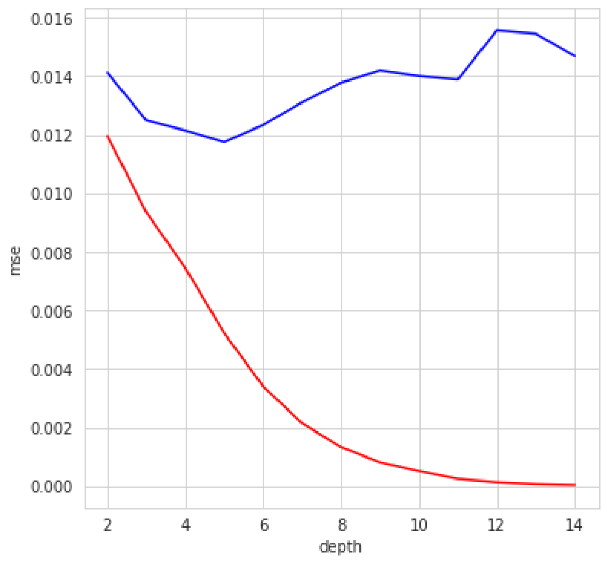
Pattern Recognition and Machine Learning
Report for : Lab 3

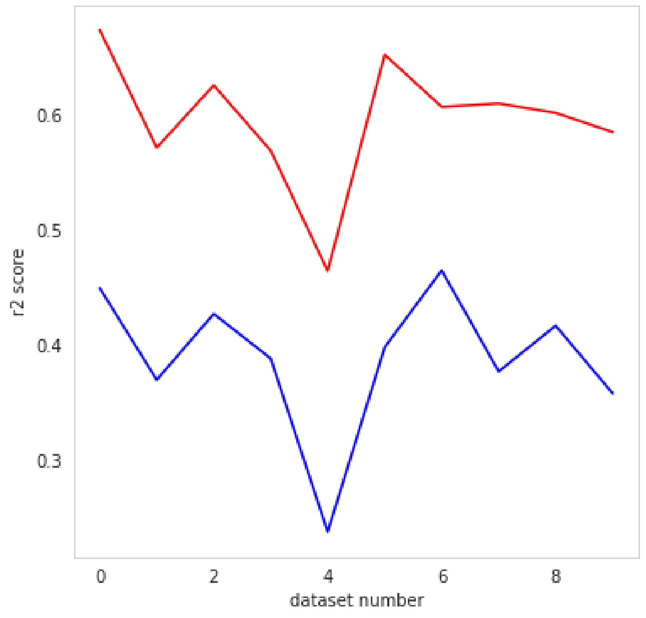
Name : Khushi Parikh
Roll No : B20EE029

Question 1:

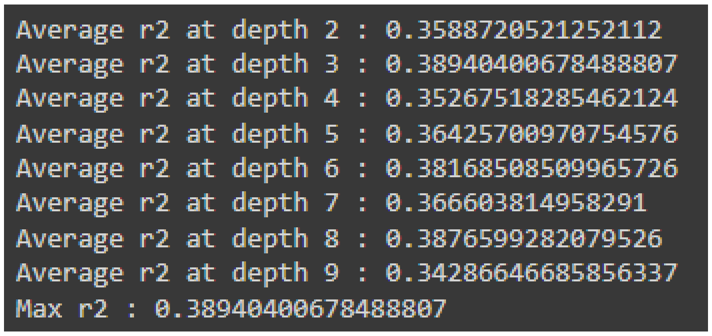
- **(1.1)** After importing the necessary libraries, we imported the data("Housing.csv"). We looked at the features of the data - data types, shape, scaling using MinMaxScaler, head of pandas table and dropped the NaN values.
- I used a correlation matrix in order to understand how much the parameters were related to each other. Price, the target variable depended mostly on **area, bathrooms and air conditioning.**
- The data was split into training and testing sets. Using the decision tree regressor, I trained the tree on the training set and then predicted the testing data. The mean squared error was 0.0204 and r2 score was 0.34.
- **(1.2)** Cross validation was implemented from scratch by choosing random indexes and then dropping it from the original dataset so that it is **not used again.** Using this I was able to make 5 folds. One of these folds was used for testing whereas the other four were used for training. Training and testing the data was **repeated for different depths** from 2 to 14. The outcomes for training data were also predicted.
- **(1.3)** Red line is on the training data, blue is on the test data. Minimum value for test is at max_depth=3, but the error on testing data is high. A depth of about 6 seems optimal.



- (1.4 + 1.5) Bagging is used to create various datasets. Taking the number of samples in each dataset to be around 4/5th of the original dataset, we create 10 such datasets. This is done by choosing random indices in the range of the length of the original dataset. **Repetition is allowed.** We fit each such dataset in the regressor decision tree and predict the values for training and testing datasets.
- (1.6 + 1.7) We check the average r2 score obtained for each of the 10 datasets. We then take the average of these. This is done for both training and testing datasets. Red line is on the training data, blue is on the test data.

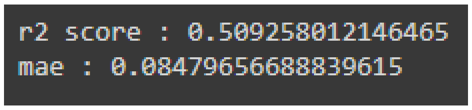


- (1.8) We check the r2 score at different depths. We can see that we have a maximum r2 score at max_depth=6 and the score decreases on either side.



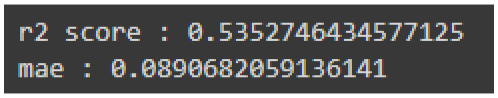
```
Average r2 at depth 2 : 0.3588720521252112
Average r2 at depth 3 : 0.38940400678488807
Average r2 at depth 4 : 0.35267518285462124
Average r2 at depth 5 : 0.36425700970754576
Average r2 at depth 6 : 0.38168508509965726
Average r2 at depth 7 : 0.366603814958291
Average r2 at depth 8 : 0.3876599282079526
Average r2 at depth 9 : 0.34286646685856337
Max r2 : 0.38940400678488807
```

- (1.9) We have imported a random forest regressor here. As a parameter we have passed n_estimators to be 10, which means that 10 trees will be made. We fit the training data on it and predict on the testing set. To evaluate it, we use the previous functions of r2 score and mean absolute error made from scratch.



```
r2 score : 0.509258012146465
mae : 0.08479656688839615
```

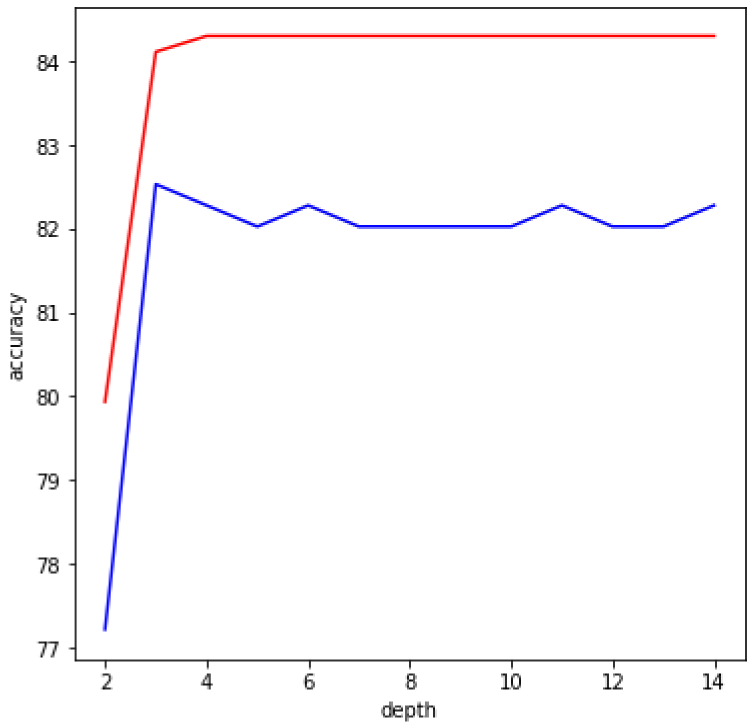
- (1.10) Here, we use the Adaboost Regressor. Similar to the above process, we fit the training data and predict the testing data.



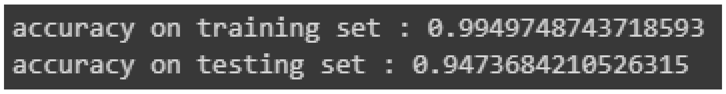
```
r2 score : 0.5352746434577125
mae : 0.0890682059136141
```

<u>Question 2:</u>

- **(2.1)** After importing the necessary libraries, we imported the data(“Breast_cancer_data.csv”). The necessary pre-processing was done. Using the classifier made in the previous lab, I have trained the classifier on the training data and then predicted the outcomes of the testing set. This gave an accuracy of about 80 percent.
- **(2.2)** Using **5 fold cross validation**, I have divided the training data into 5 parts and used 4 of them for training, and 1 of them for validation. The outer loop was used to vary the maximum depth parameter of the regressor. For each max depth, the average accuracy on each of the five folds was calculated. The best of these values is the best max_depth.
- **(2.3)** Red line is on the training data, blue is on the test data. The best max_depth occurs at around **max_depth=3**. After that the accuracy on the training data is maximum and on the testing data decreases, showing overfitting. Before it, the accuracy on both the training data and testing data is low.

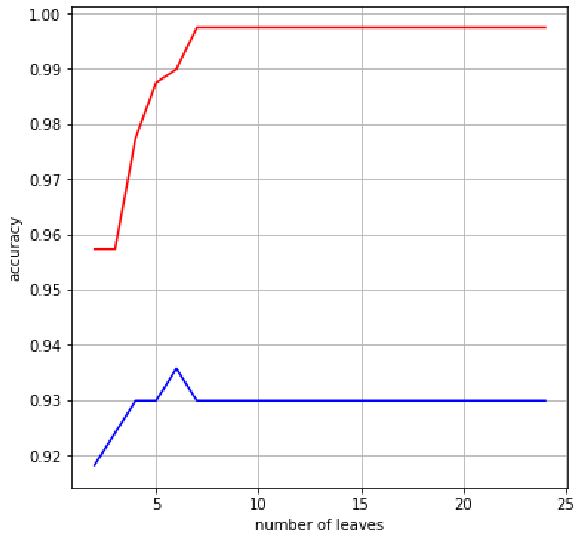


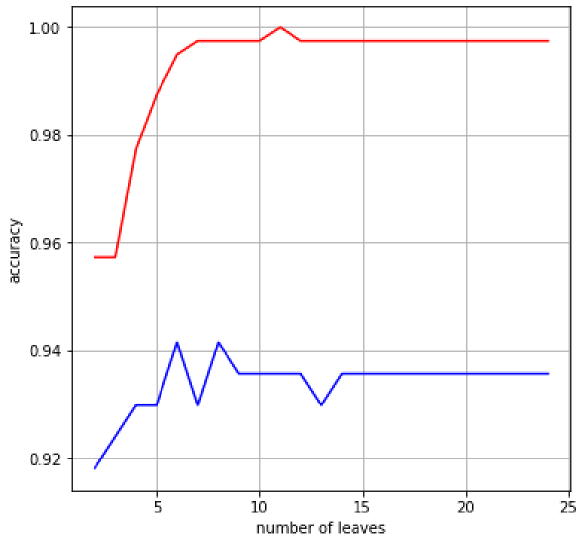
- **(2.4)** XGBoost or extreme gradient boost was installed and implemented using given parameters.
- **(2.5)** Checking the accuracy on both the training and testing set, we get



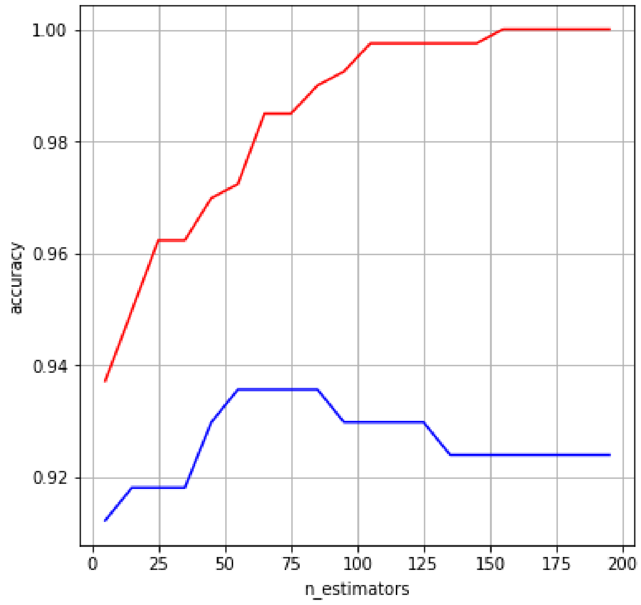
```
accuracy on training set : 0.9949748743718593
accuracy on testing set : 0.9473684210526315
```

- **(2.6)** LightGBM is installed and max_depth parameter is kept as 3. The number of leaves is varied from 2 to 25 and predictions are made on the training and testing data.
- **(2.7)** Red line is on the training data, blue is on the test data. The first graph is at max depth 4 and second one is at max depth 5. The first one overfits at around 6 leaves and the second one overfits at around 9. The number of leaves in a tree is 2^n. The accuracy is low at the start of both graphs due to underfitting. Later the accuracy on the training data increases and reaches a maximum and at that time the accuracy on the testing data is low. This shows overfitting.

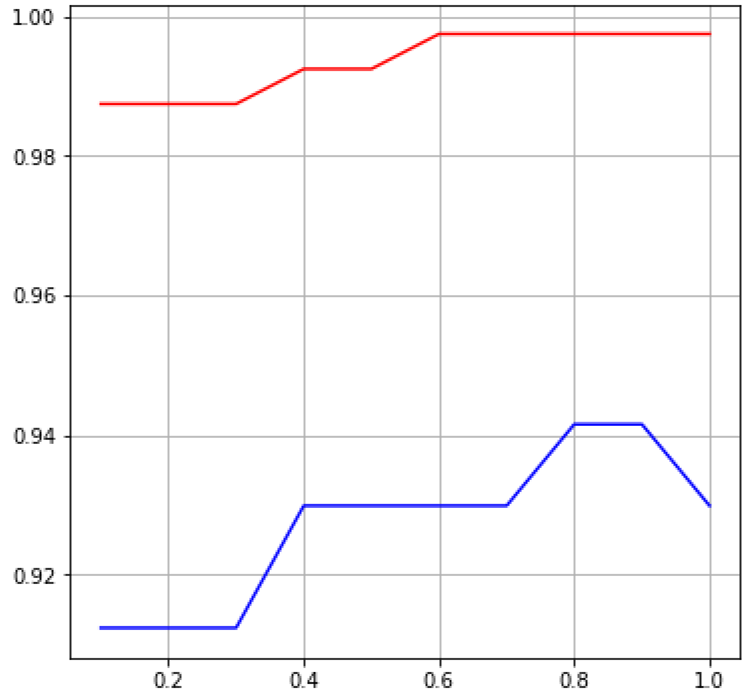




- **(2.8)** Comparing n_estimators from 5 to 200 in intervals of 10, we get the following graph. After 85, the model starts overfitting as the accuracy of training is maximum and that of testing is very low.



  Changing feature fraction from 0.2 to 1, we see the below graph. After 0.9, the model starts overfitting as the accuracy on testing decreases.



  To reduce overfitting, we can tune the following parameters -
  - Use small number of num_leaves (changes maximum number of leaves at each node split)
  - Use larger training data
  - Setting a max_depth to prevent tree from splitting after a certain depth