Pattern Recognition and Machine Learning

Lab - 5 Assignment

Bagging and Boosting

Akriti Gupta
B21AI005

Question 01:

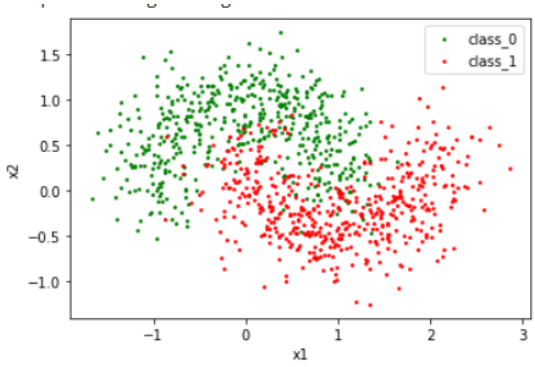
Q1

(a)

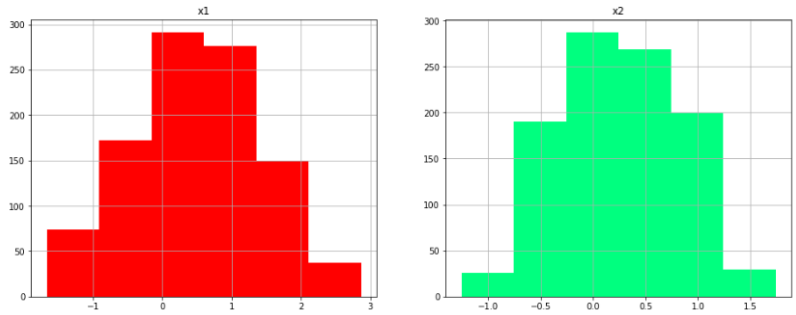
We first import all the required libraries then by using the make moon's function of sklearn, create a dataframe df.

Precprocessing:

We preprocess the dataset by first hecking if there are any null values. As we see no null values so we check the dataset further by seeing if there is noise in the dataset i.e if any of the values are not float. We find that all the values are of datatype float so we proceed further and visulaize the dataset using scatter plot in which green colored datapoints indicate points belonging to class 0 and red ones indicate points belonging to class 1.



Next we plot histograms to see the distributions of attributes x1 and x2.



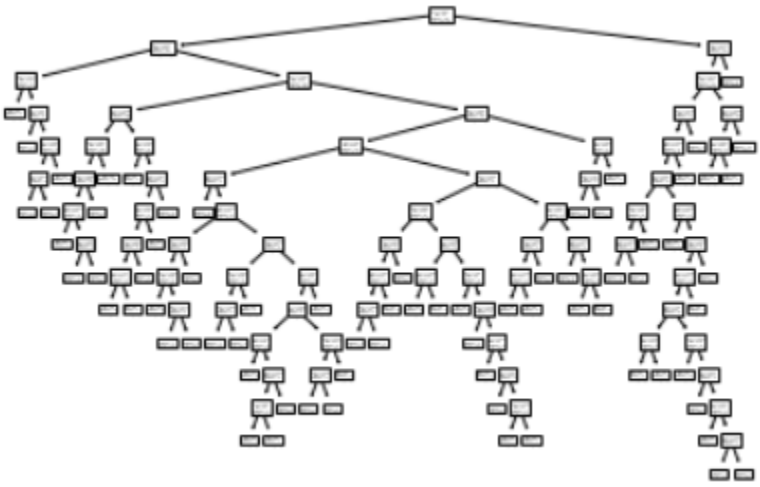
Now, we see the description of the attributes and scale them using minmaxscaler.

Splitting:

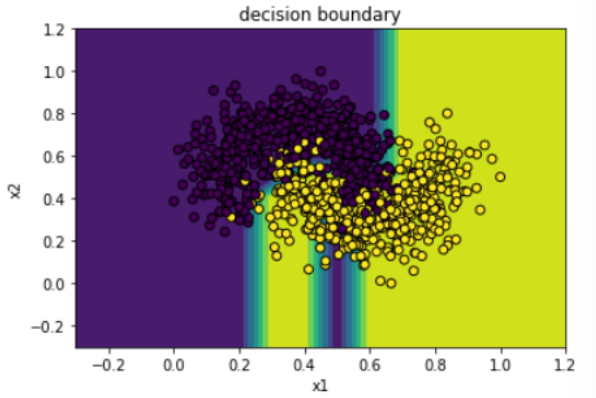
We now split the dataset into training and testing in the ratio 70:30.

(b)

Now we train a decision tree using sklearn library and fit training data into the classifier. We obtain the below tree:

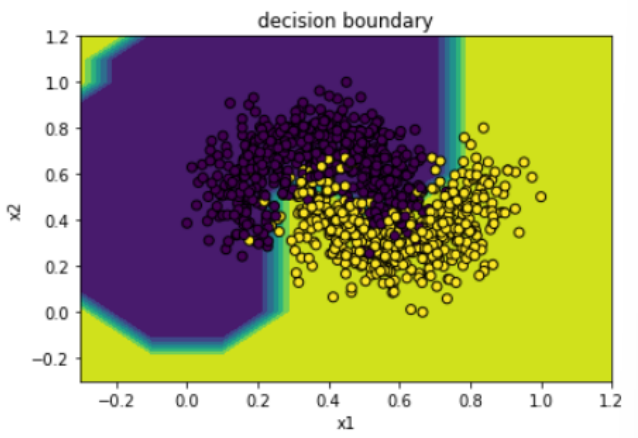


The decision boundary obtained is as follows:



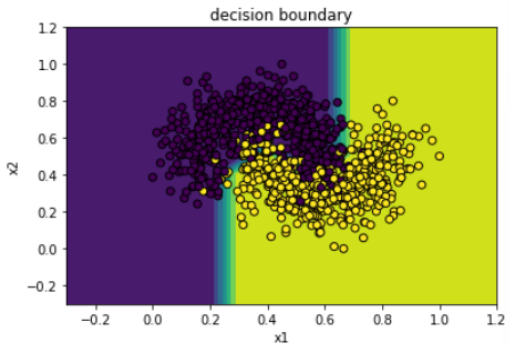
We get the accuracy score for
training data= 100%
and testing data = 89.66 % respectively.
Now we find the best value for max_depth using hyperparamet tuning. We define a function which checks accuracy of the model on testing data for different values of max_depth and get 7 as the optimal value.
max_depth=7
best_accuracy=90.66%

(c)
For this, we train a bagging classifier on the dataset using sklearn library and take base_estimator as SVC(). The decision boundary obtained is as follows:



We get the accuracy score for training and testing data as 91.14 and 90.33 respectively.

(d)
For this, we train a Random forest classifier on the dataset using sklearn library. The decision boundary obtained is as follows:



We get the accuracy score for training and testing data as 100 and 90.66 respectively.

By comparing the models, we can see that overfitting is happening in the desion tree classifier model as the accuracy for the training set is 100% whereas the accuracy on testing data is lowest. The testing accuracy for the baggingClassifier and the randomForestClassifier is better than the decision tree model. Hence, we can say that overfitting is less in these as compared to the decision tree one.Also, decision boundary is more even and polished for random forest than for bagging. This makes us conclude that increased randomness reduces overfitting and hence the testing score is also better.

(e)

Next, we vary the number of estimators for bagging classifier from 5 to 25 and for random forest from 70 to 200. We generate the decision boundaries and accuracy scores for each of these cases and obtain the below information from these:

For bagging , the best accuracy i.e 91% is obtained when no. of estimators=15 and for random forest the best accuracy obtained is 92% for no. of estimators=150.

From here we can observe that when no. of estimators gets close to its optimal value than there is very less change in accuracy and the decision boundaries also don't change much. For the optimal values, the decision boundary becomes even and polished as no outlying lines are observed in the boundary plot.

Q2.

We implement the bagging algorithm from scratch by defining a class named bagging. In this class we have the following methods/functions:

- Constructor: The use of this constructor is to initialize the dataframe, testing dataset and n_estimators which will be given by the user as parameters when using the bagging algorithm.
- Create_df: this create the random samples with replacement datasets which will be used in the method classify().
- Classify: used to train a decision tree model using a random sampled dataset.
- Set_classify: used to train n_estimators to create an ensemble of decision trees.
- Predict_set: used to predict the value of target for a test dataset i.e x_test for each classifier and return a list consisting y_test lists for each classifier.

- Predict_y: Here we take the x_test and return y_predicted for it. This is done by making use of the Predict_set method and then count the majority votes to get the correct target class of x_test value. Then we append it into a list called y_predicted and return it.
- Performance: gives us the accuracy score of each classifier and also plots the decision boundary for them.
- Avg_performance: gives us the accuracy score of the bagging algorithm.

Now, we implement this on our dataset by taking n_estimators=10. We fit the training data into it and get the accuracy score for testing dataset as 90%. The individual scores we get are 89.33%,89.66%, 89% etc which are less than the bagging algorithm score. Hence, we can clearly state that bagging algorithm is better than individual trees as it takes the majority votes of all the trees.

Question 02:

Q1.
We train an adaboost model from sklearn and then store the training and testing accuracies in 2 different variables.

Q2.
We train an xgboost model from sklearn with hyper-parameter subsample as 0.7 and then store the training and testing accuracies in another 2 variables.

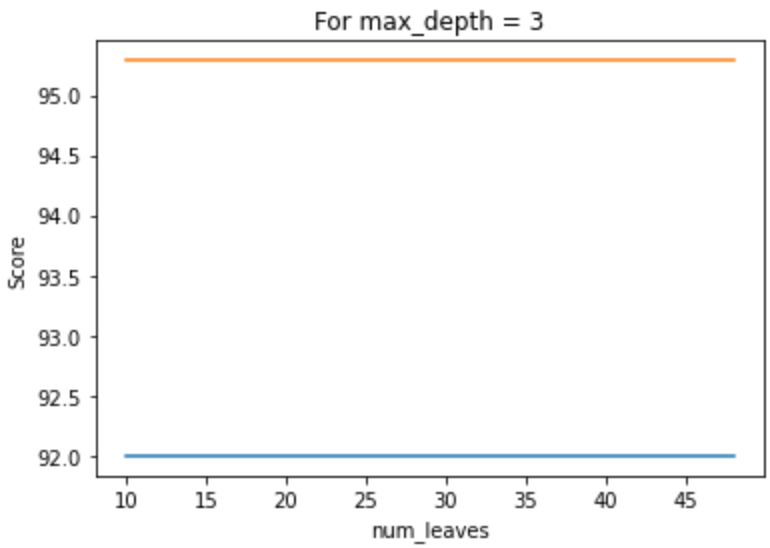
Q3.
Now we display the training and testing accuracies for both the models trained and get the following results:

For adaboost model:
Training accuracy is 0.944
Testing accuracy is 0.89

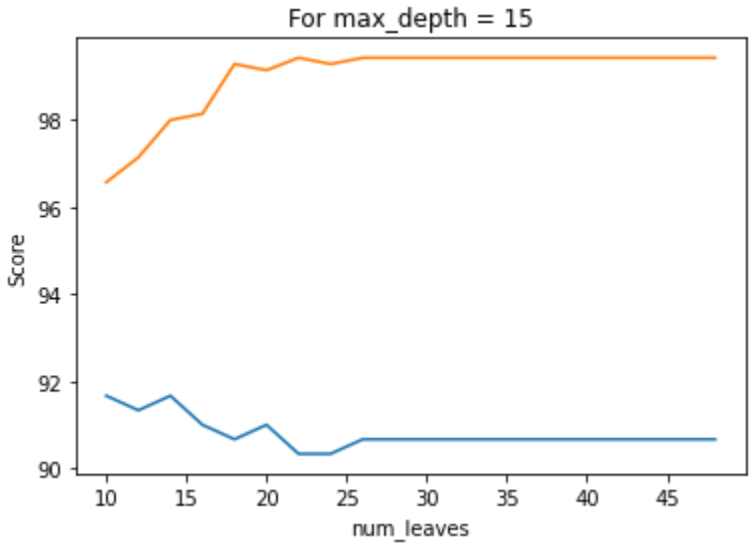
For xgboost model:
Training accuracy is 0.994
Testing accuracy is 0.9033

Q4.
Now we train lightgbm model from sklearn and using the default hyper-parameterised model, get the testing accuracy as 0.8967
Now we tune the hyper-parameter num_leaves considering values of num_leaves from 2 to 39 and get the best testing accuracy when num_leaves=3 and the best testing accuracy comes out to be 0.92.

Q5.
To analyze the relationship between num_leaves and max_depth we first keep max_depth as constant and vary the num_leaves to get the score and plot both training and testing score and print the accuracies for each combination as well. We observe that as the max_depth increases our algorithm has more freedom of expanding further ie. Get deeper, as it gets deeper upon increasing num_leaves the model obtains a good fit till some limit of value of num_leaves after which it starts overfitting. We observe that for lower value of max_depth the accuracy doesn't change on changing the num_leaves as that num_leaves can't be achieved at that depth and same num_leaves and max_depth is being trained everytime so the score remains constant. Now as we increase the max_depth, the we can now have more num_leaves, this freedom allows us to have a good fit. Overfitting as observed from the plots starts at: max_depth=15 num_leaves=26 as training score is very high and stays constant whereas testing score keep on decreasing

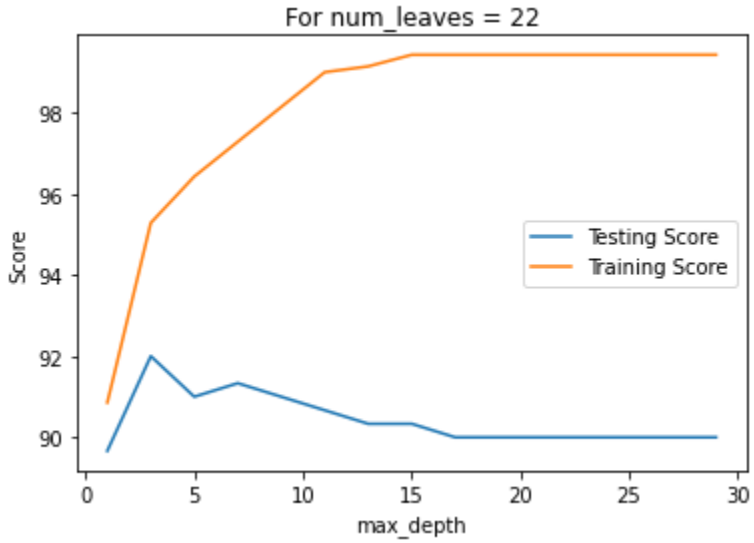


We can clearly see for lower value of max_depth the score doesn't change for whatever value of num_leaves we take



We can see that for max_depth=15 and num_leaves=26 the training accuracy from that num_leaves onwards goes on increasing but testing accuracy decreases indicating that overfitting started taking place.

Similarly we keep num_leaves as constant, now for smaller value of num_leaves the max_depth can't increase much so the scores doesn't vary, but as num_leaves increases each trees gets more flexibility to expand to limit till which It get's good accuracy on training data. Now, if we keep on increasing num_leaves then a limit comes when overfitting starts to occur and the testing accuracy is reduced.
Overfitting as observed from the plots starts at: max_depth=17 and num_leaves=22 as training score is very high whereas the testing score remain low after that



We can clearly see again that overfitting starts for num_leaves=22 and max_depth=17 using the similar logic

So, combining both results we can say that overfitting starts as max_depth approaches value 16 and num_leaves approaches value 24.

Q6.

The hyper-parameters which can be tuned are:

1)max_depth: Increasing max_depth to a certain level(ensuring overfitting doesn't occur) we can increase the testing accuracy

2)num_leaves: Increasing num_leaves to a certain level we can ensure that testing accuracy increases.

3)learning_rate: For smaller learning rate, we can have more accuracy

4)reguralisation through reg_alpha and reg_lambda can help to avoid overfitting

5)n_estimators: More value of n_estimator may help normalize the outliers and make decision boundary less sensitive to outliers and thus give better accuracy.

The Best score comes out to be : 92.66666666666666

Best max_depth = 3

Best num_leaves = 4

Best learning_rate = 0.05

Best reg_alpha = 0.1

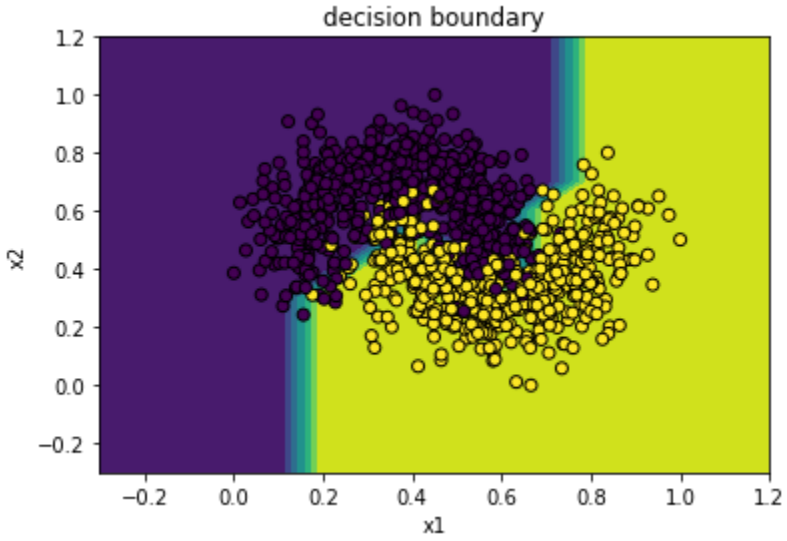
Best reg_lambda = 0.1

Best n_estimators = 100

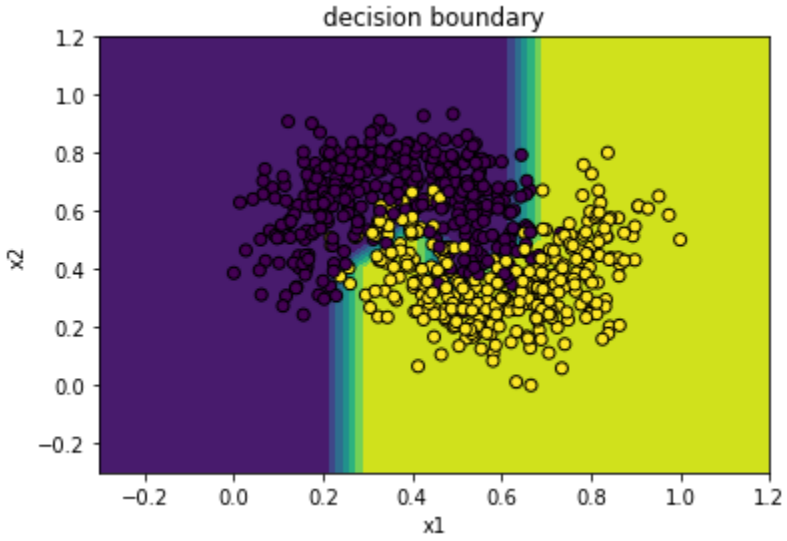
Q7.

Now, plotting the decision boundary for all the 3 models,

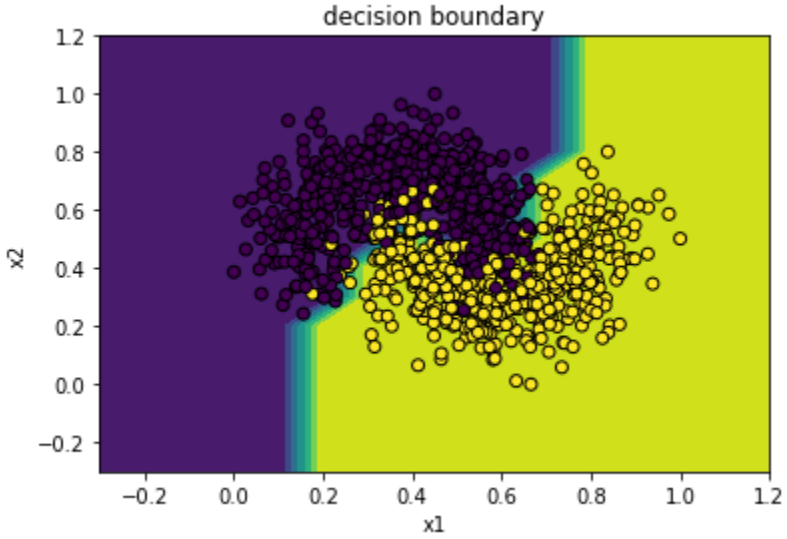
For adaboost:



For xgboost:



For LightGBM:



Observing the performance metrics(testing accuracy):
Adaboost performs the worst as accuracy is 0.89, followed by this is the lightbgm default hyper-parameterised model which has testing accuracy of 0.8967 and the best performance is by the xgboost model if default hyper-parameterised model's testing accuracy is taken into account.
So according to testing accuracy:
adaboost<lightgbm<xgboost

Now, observing the decision boundary, we can see that the adaboost model tries fitting many outliers which results in it adding minute details to the decision boundary and thus we can say adaboost doesn't perform that well on training data(0.944 accuracy) as well as testing data(0.89 accuracy)
On the other hand, xgboost has quite a descent performance on both the training data(0.994 accuracy) and testing data(0.9033 accuracy).
Observing the decision boundary of adaboost, we can get to see that the decision boundary again tries covering minute details of each training sample resulting in lower testing accuracy.

Question 03:

Firstly, we train gaussian naive bayes model and get the training and testing accuracies according to the default hyper-parameterised model as 0.85 and 0.86 respectively.
Now, we vary the hyper-parameter, var_smoothing and get the maximum accuracy as 0.86 for var_smoothing value of 0.0432

Now, we choose the 3 best performing models as xgboost,adaboost and bagging classifier and then, train that along with naive bayes model with the best hyper-parameter and then with all these 4 models build our voting classifier model. Now, we train the voting classifier with training data and get the accuracies as follow:
Training accuracy is 0.927
Testing accuracy is 0.9033

Now the individual models had the following training and testing accuracies:
1)Naive Bayes: Training=0.85, Testing=0.86
2)Adaboost: Training=0.944, Testing=0.89
3)LightGBM: Testing=0.8967
4)XGBoost: Training=0.994, Testing=0.9033

Clearly we can see that the testing accuracy is the best in case of voting algorithm and also it has the lowest variance showing that it has minimised the overfitting to some great extent using all these individual models.
But it has comparatively lower bias as compared to the individual models.