# Pattern Recognition & Machine Learning
# Lab-5 Assignment

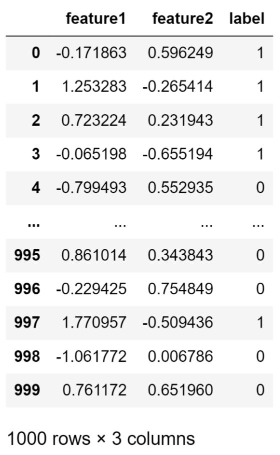
**Name:** Tanish Pagaria
**Roll No.:** B21AI040

## Question-1 [Bagging]

### Task-1

Dataset Generation

A dataset with 1000 samples with `random_state=42` and `noise=0.3` was generated using the `sklearn.datasets.make_moons()` function. The generated dataset was converted into a Pandas dataframe.

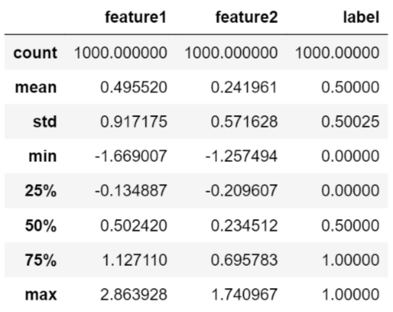


| | feature1 | feature2 | label |
|---|---|---|---|
| **0** | -0.171863 | 0.596249 | 1 |
| **1** | 1.253283 | -0.265414 | 1 |
| **2** | 0.723224 | 0.231943 | 1 |
| **3** | -0.065198 | -0.655194 | 1 |
| **4** | -0.799493 | 0.552935 | 0 |
| **...** | ... | ... | ... |
| **995** | 0.861014 | 0.343843 | 0 |
| **996** | -0.229425 | 0.754849 | 0 |
| **997** | 1.770957 | -0.509436 | 1 |
| **998** | -1.061772 | 0.006786 | 0 |
| **999** | 0.761172 | 0.651960 | 0 |

1000 rows × 3 columns

*Generated dataframe*

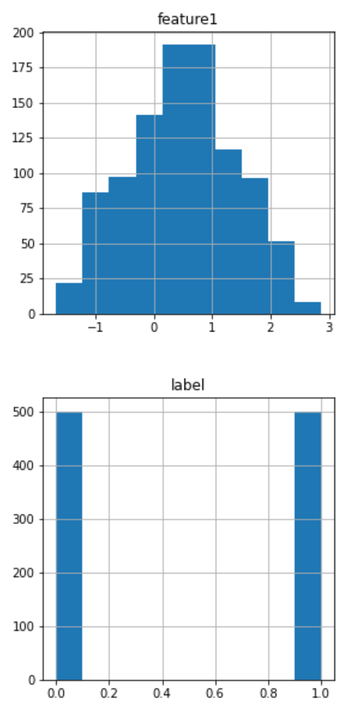
Preprocessing and Exploratory Analysis



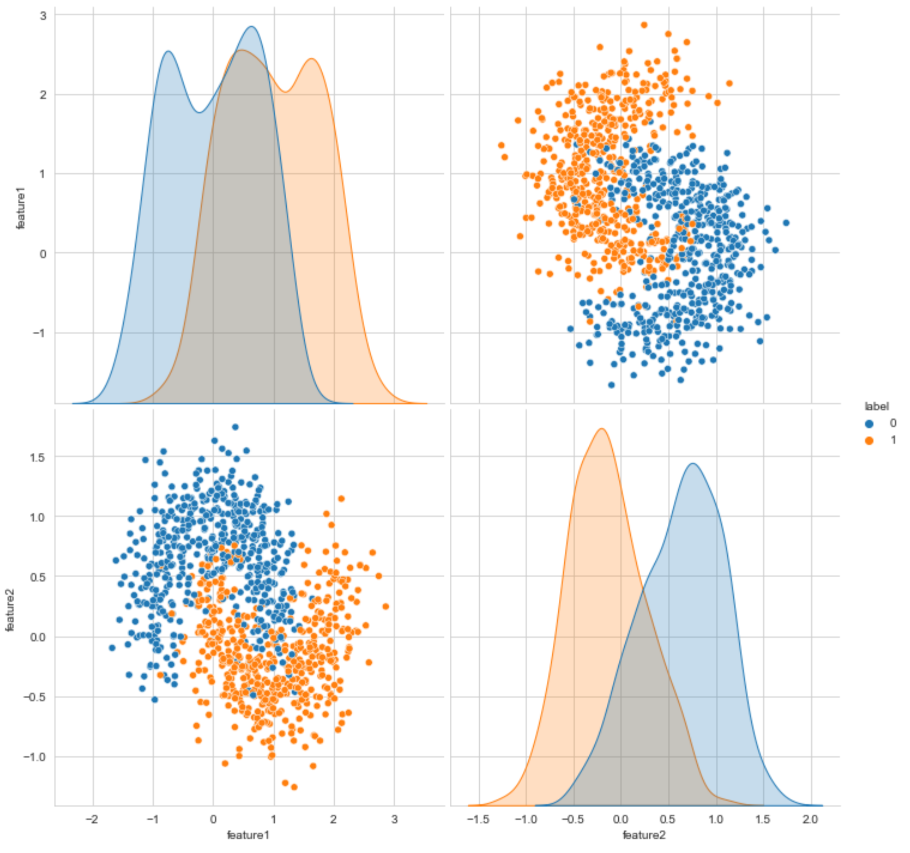
| | feature1 | feature2 | label |
|---|---|---|---|
| **count** | 1000.000000 | 1000.000000 | 1000.00000 |
| **mean** | 0.495520 | 0.241961 | 0.50000 |
| **std** | 0.917175 | 0.571628 | 0.50025 |
| **min** | -1.669007 | -1.257494 | 0.00000 |
| **25%** | -0.134887 | -0.209607 | 0.00000 |
| **50%** | 0.502420 | 0.234512 | 0.50000 |
| **75%** | 1.127110 | 0.695783 | 1.00000 |
| **max** | 2.863928 | 1.740967 | 1.00000 |

In the dataset, both features had similar ranges, means, and standard deviations. Hence, normalization was not required. Also, the dataset did not have any NULL or NaN entries.
There were an equal number of "0" and "1" labels.

Visualization of the dataset



*Histogram plots of the dataset*

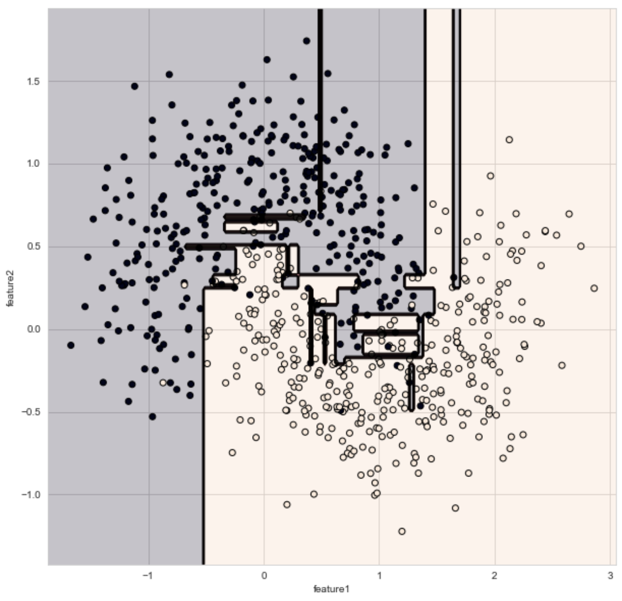


*Dataset Visualization (Features & Labels)*

The dataset was then split into training and testing sets using the `sklearn.model_selection.train_test_split()` function.

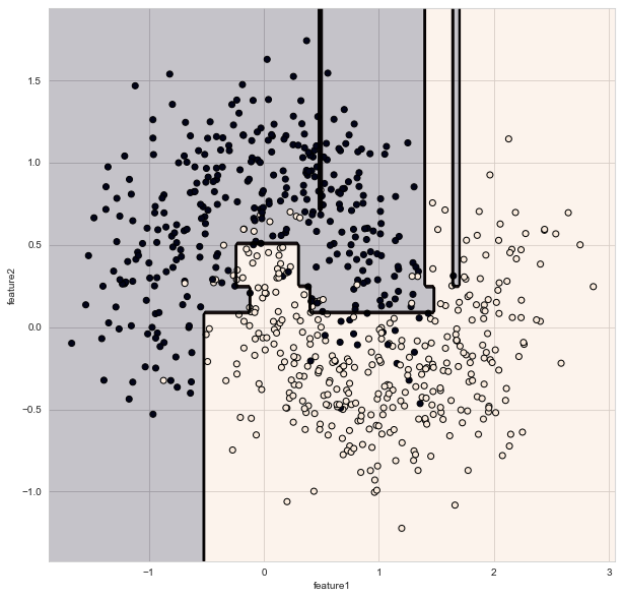
Training a simple DecisionTreeClassifer

The `sklearn.tree.DecisionTreeClassifier()` class was used to implement a simple decision tree classifier. The training set was used to train the classifier. The decision boundary for the classifier was plotted using a self-implemented function.



*Decision Boundary for the DecisionTreeClassifier*

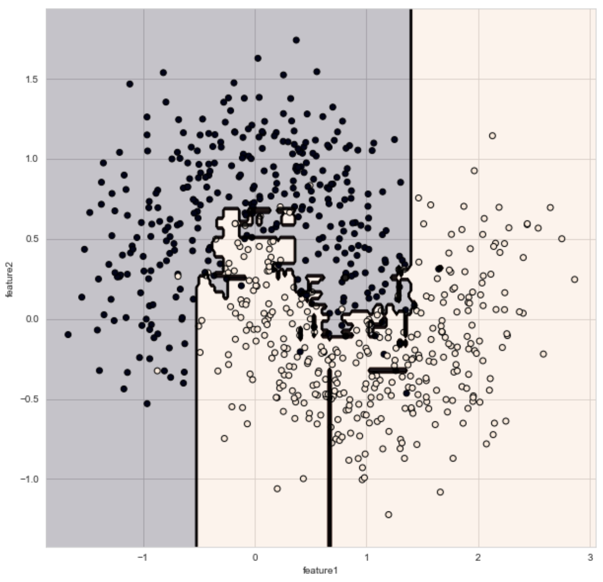
Hyperparameter tuning was performed to find the best value of max_depth for the decision tree classifier. Using a loop, the `max_depth` hyperparameter was varied in order to find the depth at which the model gave the best accuracy. The best accuracy was obtained at `max_depth=6`. The corresponding decision boundary is shown below.



*Decision Boundary for the DecisionTreeClassifier at max_depth=6*

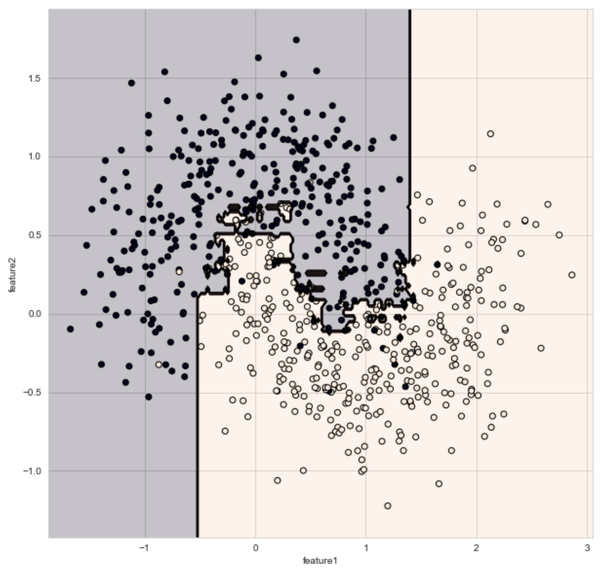
<u>Training a BaggingClassifier</u>

The `sklearn.ensemble.BaggingClassifier()` class was used to implement the Bagging classifier. The training set was used to train the classifier. The corresponding decision boundary is shown below.



*Decision Boundary for the BaggingClassifier*

<u>Training a RandomForestClassifier</u>
The `sklearn.ensemble.RandomForestClassifier()` class was used to implement the Random Forest classifier. The training set was used to train the classifier. The corresponding decision boundary is shown below.



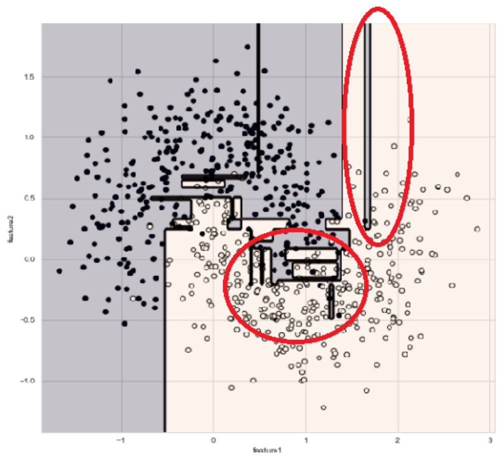
*Decision Boundary for the RandomForestClassifier*

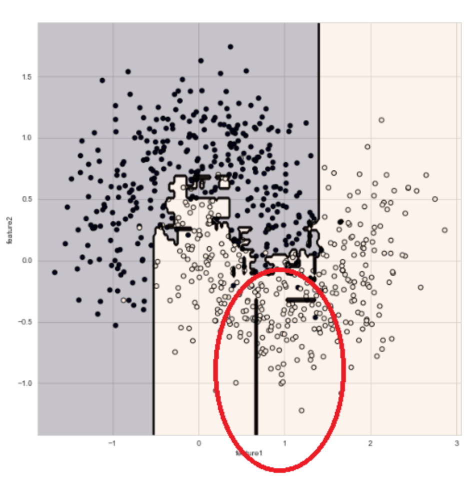
<u>Comparison among the three models</u>
The accuracy scores obtained for the classifiers are as follows:-

- **DecisionTreeClassifier:** 0.8866666666666667
- **BaggingClassifier:** 0.9066666666666666
- **RandomForestClassifer:** 0.9133333333333333

From the above accuracy scores, we can easily say that the performance of the models was in the order:-
***Performance:*** *RandomForest > Bagging > DecisionTree*

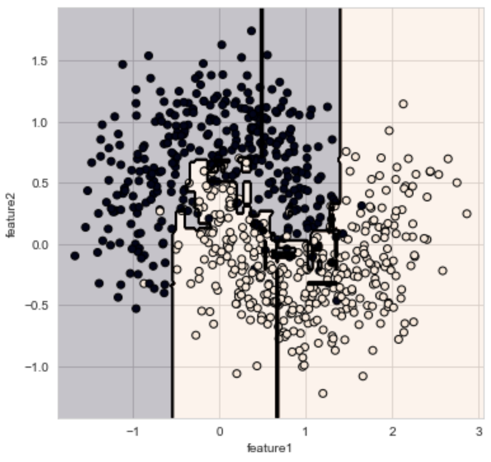
From the decision boundaries plotted for the three models, we observe that in the case of the DecisionTree and Bagging classifiers, the models are trying to overfit the data, as visible in the figures shown below.

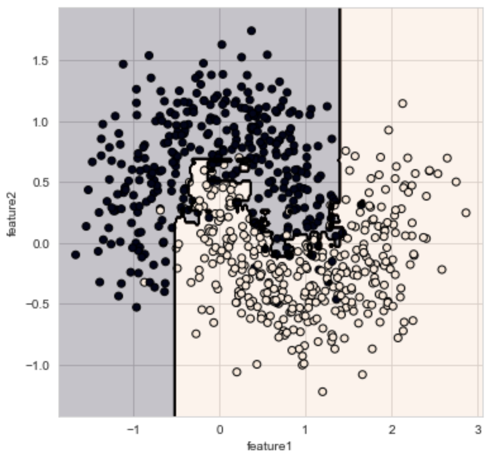


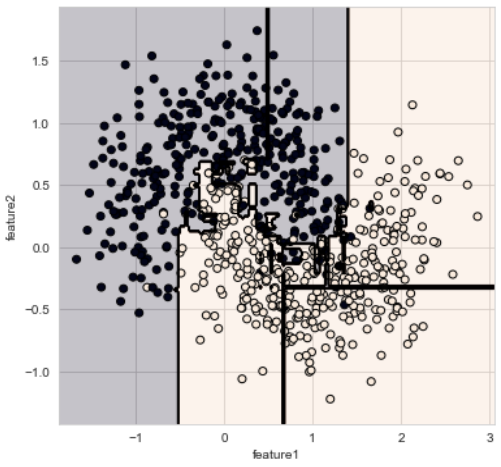


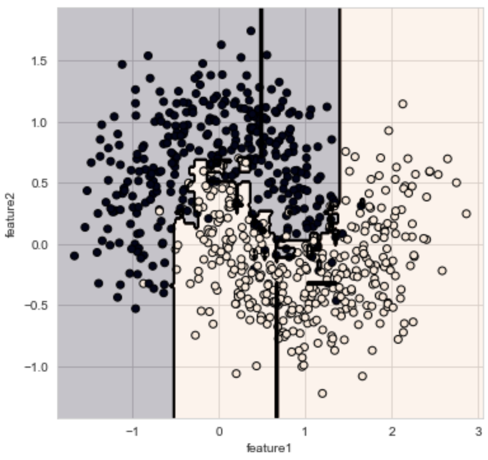
However, the decision boundary in the RandomForest Classifier shows that the model is well-generalized, more accurate, and also not overfitting.

<u>Varying the number of estimators for the BaggingClassifier & RandomForestClassifier</u>
By using a loop and changing the number of estimators, we observe the following decision boundaries for the Bagging classifier.

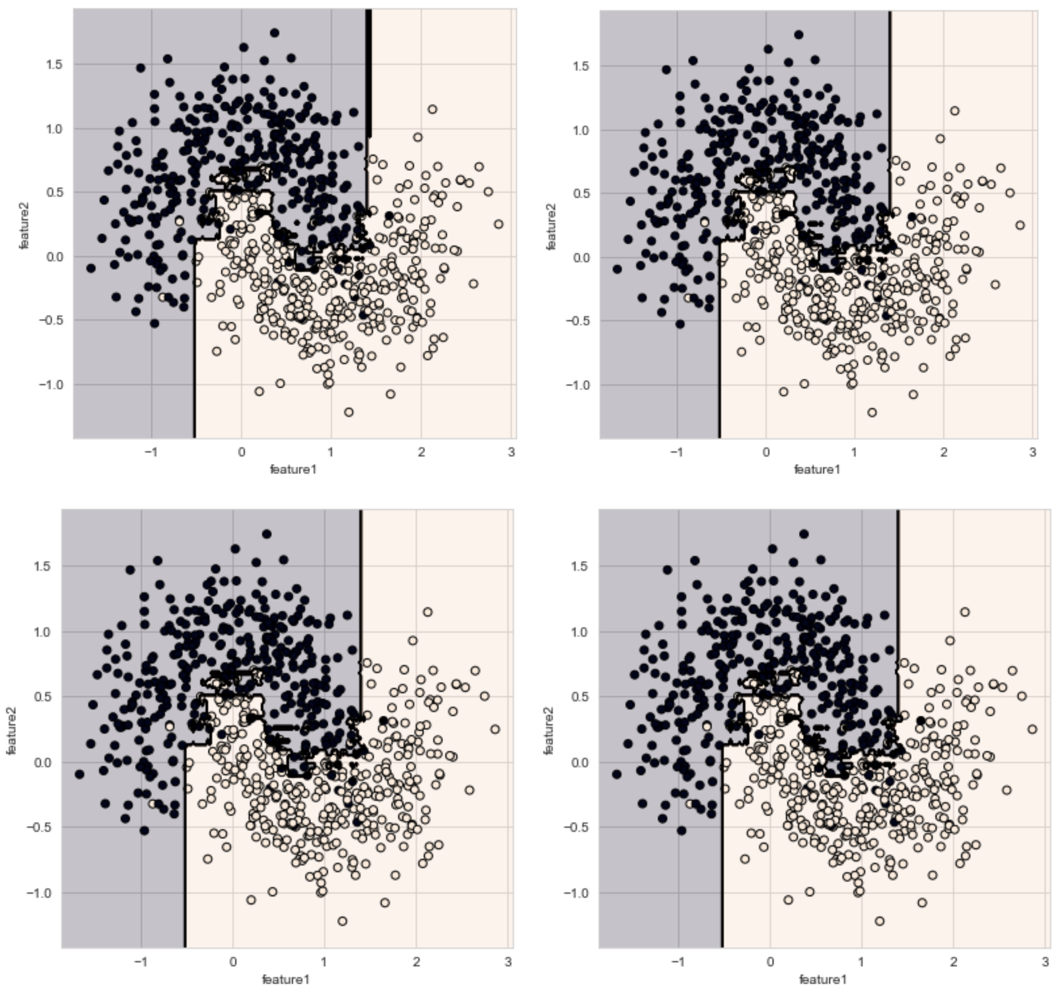








We observe that initially, in the case of the Bagging Classifier, the model was trying to overfit the data, but it became more generalized with the increase in the number of estimators. Although the accuracy scores obtained show little variation.



In the case of the Random Forest classifier, we don't see a lot of changes in the decision boundaries. Furthermore, the accuracy scores become nearly constant.

**Task-2**

Implementing a Bagging algorithm from scratch

A new class was defined for the Bagging Classifier. It took in the number of estimators in the constructor. By default, we took the estimators as Decision Tree Classifiers (inbuilt from sklearn). The Bagging Classifier had a `fit()` method that took in the training data and then, using random sampling (with replacement), trained the Decision Tree Classifier models, which would get saved in the object itself. Then, we defined a `predict()` method to give the predictions of the Bagging Classifier on passing the test dataset as an argument. The method took the help of another helper method called `majority_voting()` which used the collection of predictions from all the classifiers on the test dataset and generated a final prediction array by choosing the most frequent class in the predictions from all the classifier models for each data point.

The class also had a `score()` method, which took in the test dataset and test labels and reported the accuracy of the test dataset by the classifier using the predict() method.

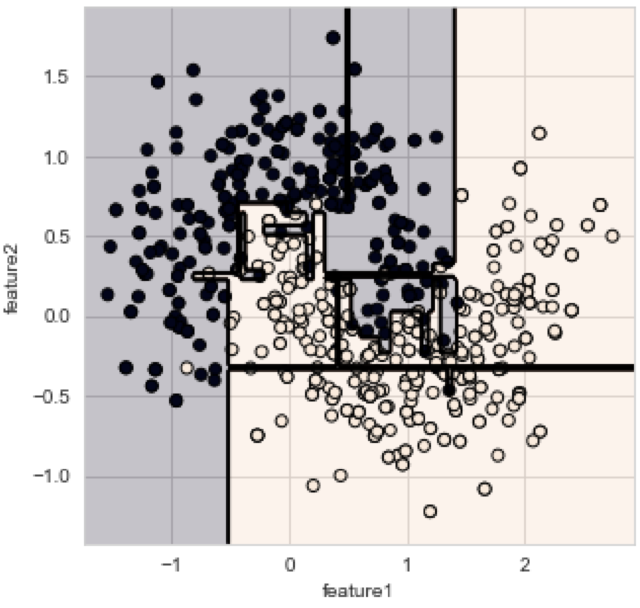
We also defined two additional methods, `individual_estimator_performance()` and `average_estimators_performance()`, which reported the performance of the individual trees in the Bagging Classifier both numerically and visually (using a decision boundary on the sampled training dataset) and the average performance of all the trees, respectively.

Training & Performance Evaluation

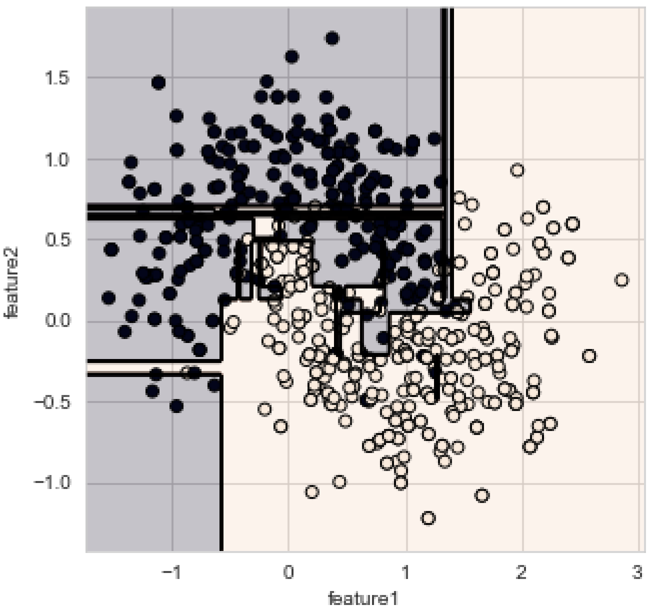
The classifier was trained on the previously used training dataset with the number of estimators as 10. The accuracy score varied from around 0.88 to 0.92 after every run (due to the variation in sampling in each run).

For instance, when we obtained the accuracy score of 0.9166666666666666, the following performance for each individual tree was obtained, as tabulated below.

Accuracy score for estimator-1: 0.88

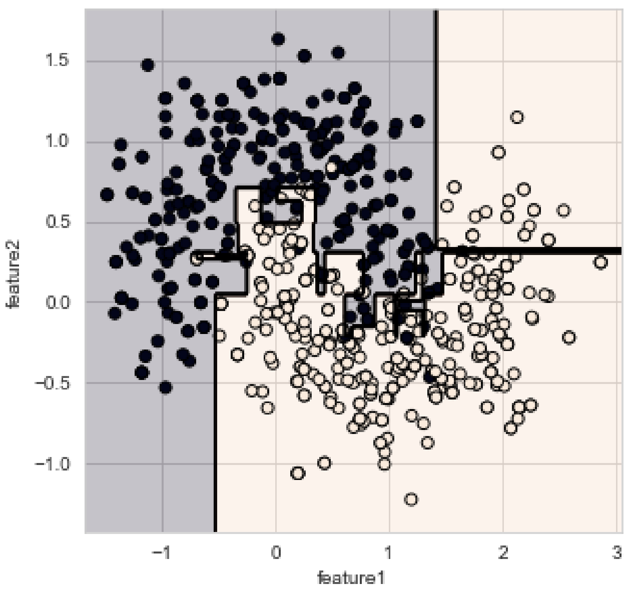


Accuracy score for estimator-2: 0.8966666666666666

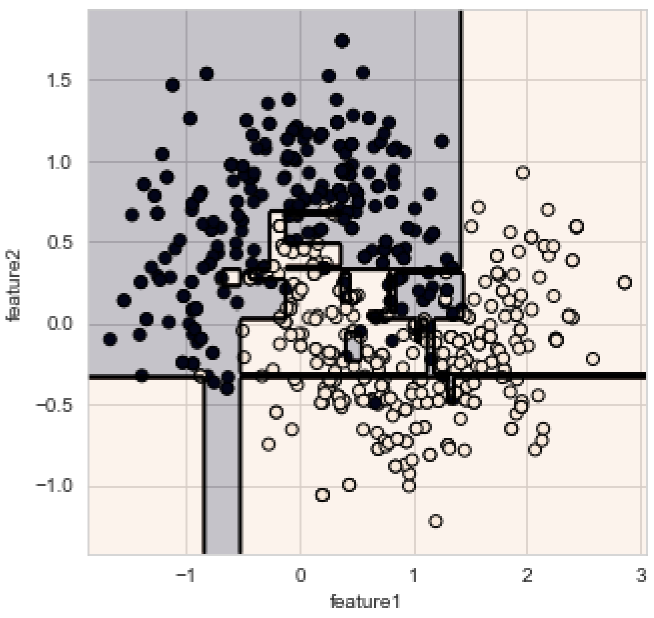


Accuracy score for estimator-3: 0.89

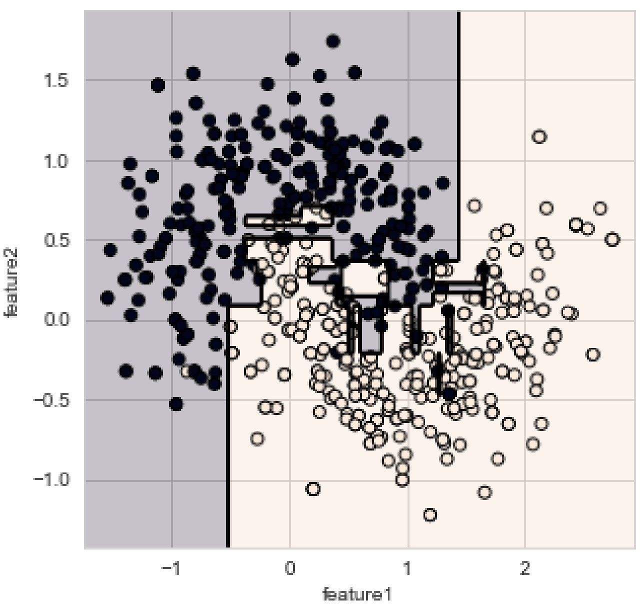
Accuracy score for estimator-4: 0.8733333333333333



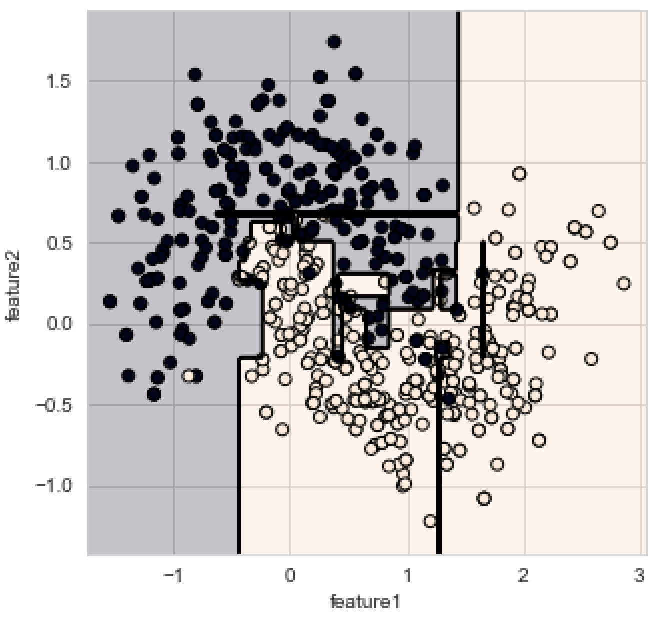
Accuracy score for estimator-5: 0.8866666666666667



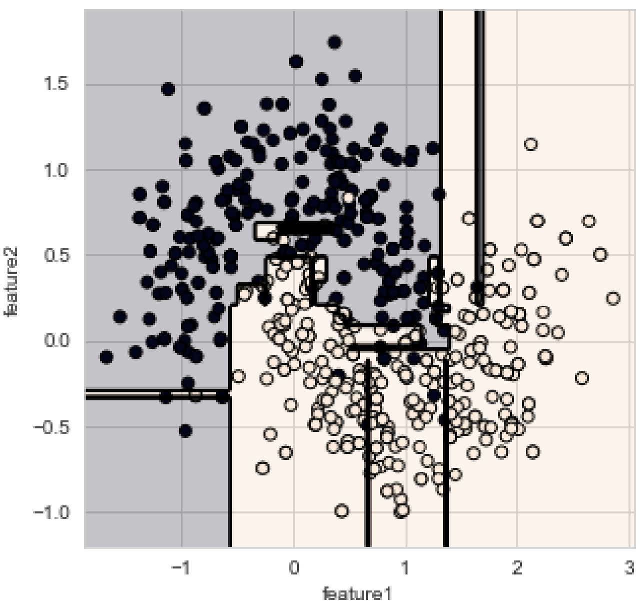
Accuracy score for estimator-6: 0.8966666666666666

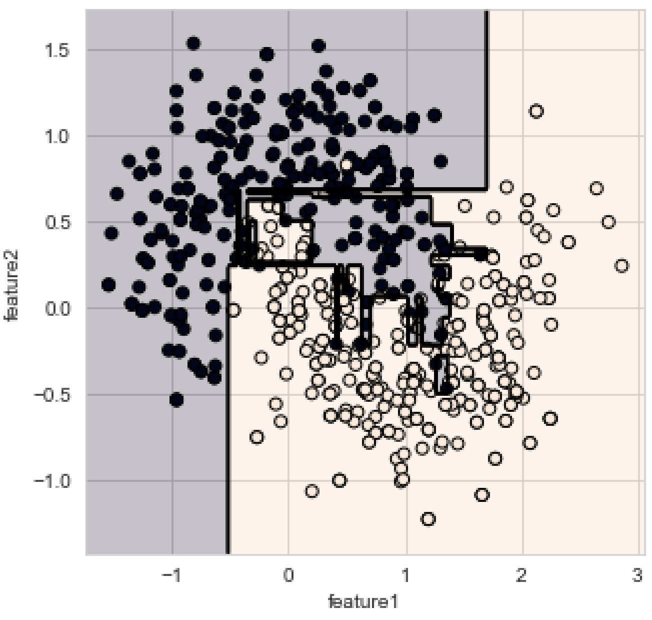


Accuracy score for estimator-7: 0.8933333333333333



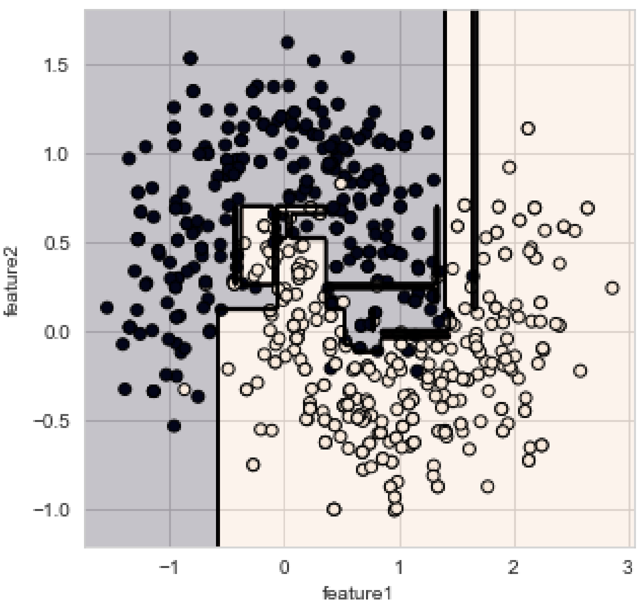
Accuracy score for estimator-8: 0.89

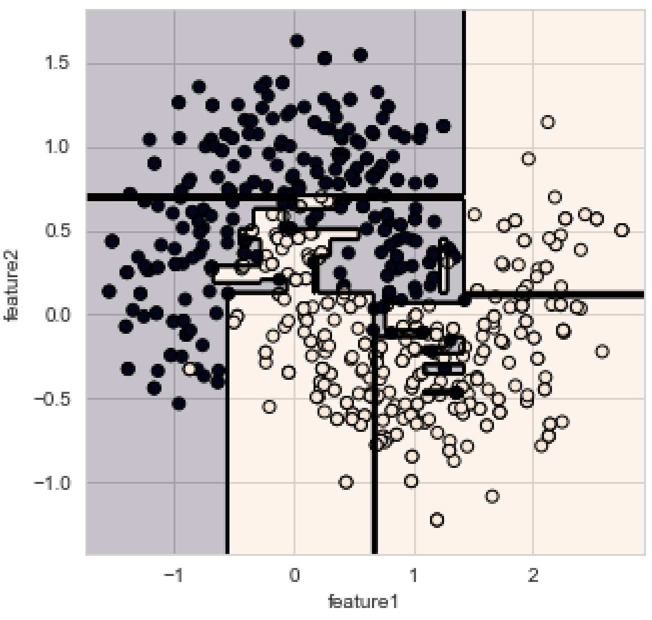




Accuracy score for estimator-9: 0.9

Accuracy score for estimator-10: 0.88





We can see that the performance of each tree is less than that of the classifier as a whole.

Average Performance

Also, the average of the accuracy scores of all the esimators was 0.8886666666666667.

## QUESTION-2 [Boosting]

### Training an Adaboost Classifier

Using the `sklearn.ensemble.AdaBoostClassifier()`, an AdaBoost classifier, was trained on the training dataset. The classifier reported an accuracy score of 0.8966666666666666 on the testing dataset.

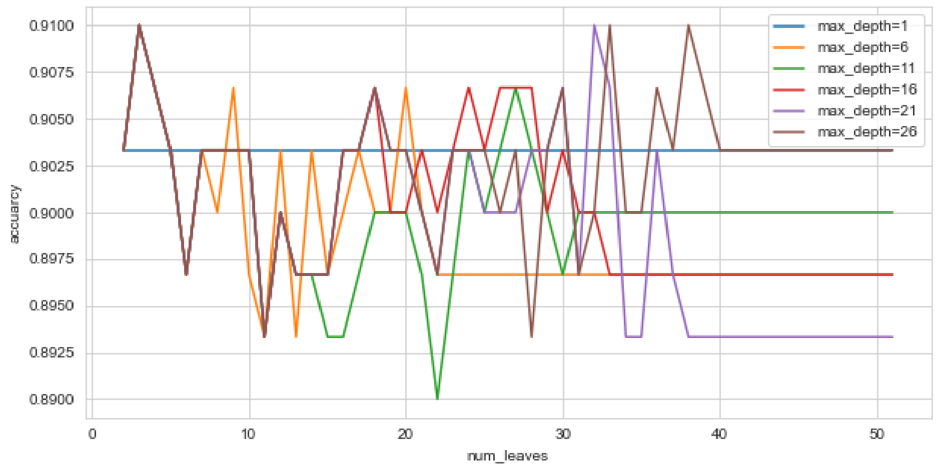
### Training an XGBoost Model

Using `XGBClassifier` from the `xgboost` library, an XGBoost classifier was trained on the training dataset, keeping `subsample=0.7`. The model reported an accuracy score of 0.9066666666666666 on the testing dataset.

### Training a LightGBM Model

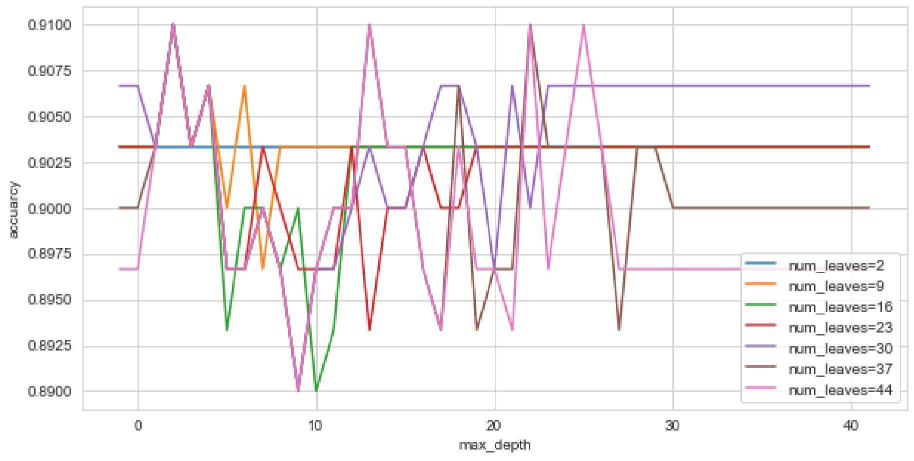
Using the `LGBMClassifier` from the `lightgbm` library, a LightBGM classifier was trained on the training dataset. We created a function to vary the `max_depth` and `num_leaves` hyperparameters of the classifier and print the accuracy scores at each hyperparameter.

**Analyzing the relation between max_depth and num_leavest**



*Accuracy v/s num_leaves for various max_depths*

We observe that after a particular value of `num_leaves` close to 40 for most of the classifiers, shown above, the accuracy gets reduced and becomes constant. This can be considered the point for overfitting.



*Accuracy v/s max_depths for various num_leaves*

A similar observation is obtained in the above graph for `max_depth` > 30.

We also tried to find the best hyperparameters for the lightGBM model by using nested for loops and finding the best values for `max_depth` and `num_leaves` that gave the best accuracy.

```python
best_max_depth = 0
best_num_leaves = 0
best_accuracy = 0

for depth in range(1, 31):
    for leaf in range(2, 51):
        score = lightGBM_accuracy(train_X, train_y, test_X, test_y, max_depth=depth, num_leaves=leaf)
        if score > best_accuracy:
            best_accuracy = score
            best_max_depth = depth
            best_num_leaves = leaf
```

```
best max_depth: 4
best num_leaves: 6
accuracy: 0.9133333333333333
```

The shown values were obtained.

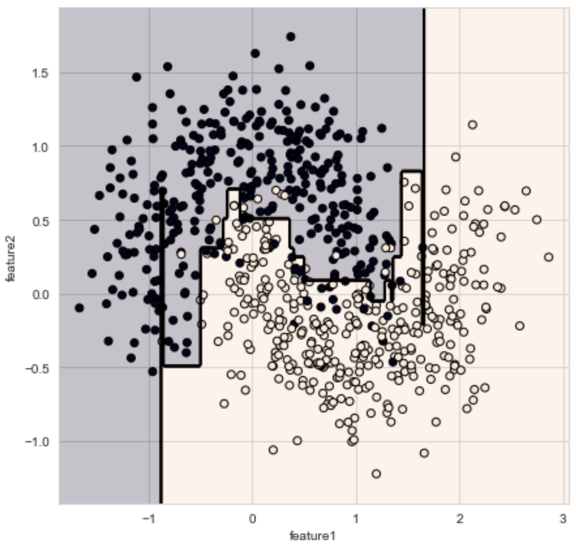
**Parameters used for better accuracy:-**
(taking reference from the official documentation of the LightGBM model)

- Use large `max_bin` (may be slower):
  `max_bin` refers to the max number of bins that feature values will be bucketed in.
- Use `small learning_rate` with large `num_iterations`
- Use large `num_leaves` (may cause over-fitting):
  `num_leaves` is the max number of leaves in one tree
- Use bigger training data

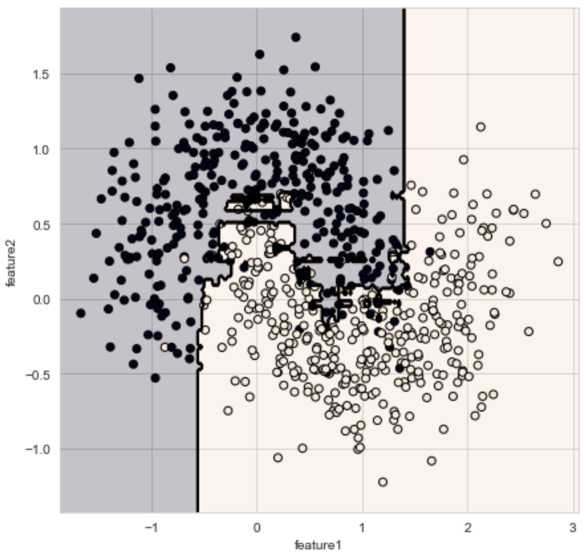
**Parameters used to avoid overfitting:-**
(taking reference from the official documentation of LightGBM model)

- Decreasing `max_depth`:
  It is used to limit the depth of the tree. This is used to deal with over-fitting when the dataset size is large.
- Decreasing `num_leaves`:
  Unconstrained leaves may lead to overfitting. We should try to keep the `num_leaves` less than 2^(`max_depth`) when tuning it.
- Using bigger training data
- Trying `lambda_l1`, `lambda_l2` and `min_gain_to_split` for regularization

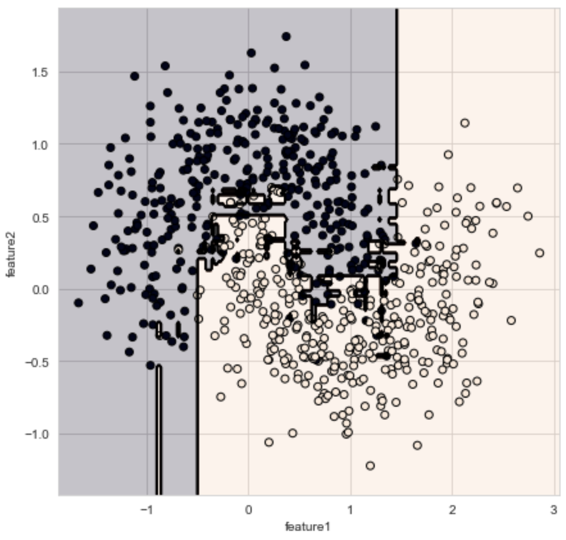
Decision boundaries for the three models



*For AdaBoost Classifier*



*For XGBoost Model*



*For LightGBM Model*

**Performance comparison among the three models:**
Based on the accuracy scores for the classifier on the testing dataset, i.e.

- Adaboost Classifier: 0.8966666666666666
- XGBoost Classifier: 0.9066666666666666
- LightGBM Classifier: 0.9

The following order was obtained: *XGBoost* > *LightGBM* > *Adaboost*.

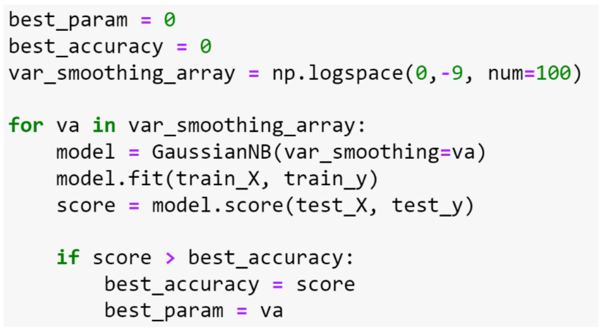
## QUESTION-3

### Training a Bayes classification model

Performed using `sklearn.naive_bayes.GaussianNB()` class. A Bayes classifier was trained using the previously used training dataset.

**Hyperparameter tuning:**

The Gaussian Naive Bayes class in sklearn has only two hyperparameters: `priors` and `var_smoothing`. `priors` do not need to be tuned. But we can tune the `var_smoothing` parameter for better performance.

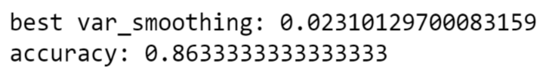


```
best_param = 0
best_accuracy = 0
var_smoothing_array = np.logspace(0,-9, num=100)

for va in var_smoothing_array:
    model = GaussianNB(var_smoothing=va)
    model.fit(train_X, train_y)
    score = model.score(test_X, test_y)

    if score > best_accuracy:
        best_accuracy = score
        best_param = va
```

*Looping over various values for the parameter to find the one that gives the most accuracy*



```
best var_smoothing: 0.02310129700083159
accuracy: 0.8633333333333333
```

*Best hyperparameter and the corresponding accuracy obtained*

### VotingClassifier

The Voting Classifier was created using `sklearn.ensemble.VotingClassifier()` class.

The accuracy scores for all the models so far:

- DecisionTreeClassifier: 0.8866666666666667
- BaggingClassifier: 0.9066666666666666
- RandomForestClassifer: 0.9133333333333333
- Bagging Classifier from scratch: between 0.89 and 0.92
- Adaboost Classifier: 0.8966666666666666
- XGBoost Classifier: 0.9066666666666666
- LightGBM Classifier: 0.9
- Bayes Classifier: 0.8633333333333333

The three trained models to be grouped with the trained Bayes classification include: RandomForestClassifier, XGBoost Classifier and Bagging Classifier

### VotingClassifier performance comparison between the individual models

The voting classifier performed better than all the individual models used to train it. It gave an accuracy score of 0.92 on the testing dataset, which was more than that of any of the individual classifiers.