Machine Learning

Lecture 04

Min-Kuan Chang

minkuanc@nchu.edu.tw

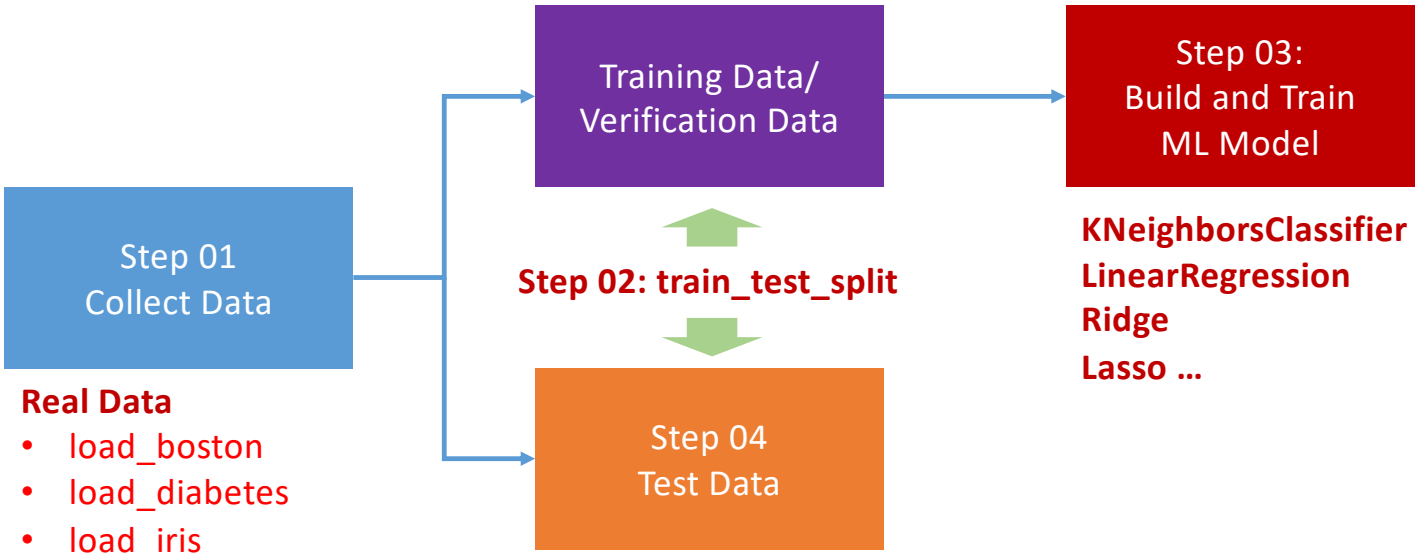
EE, College of EECS

Implementation Using Scikit-Learn

Topic 01

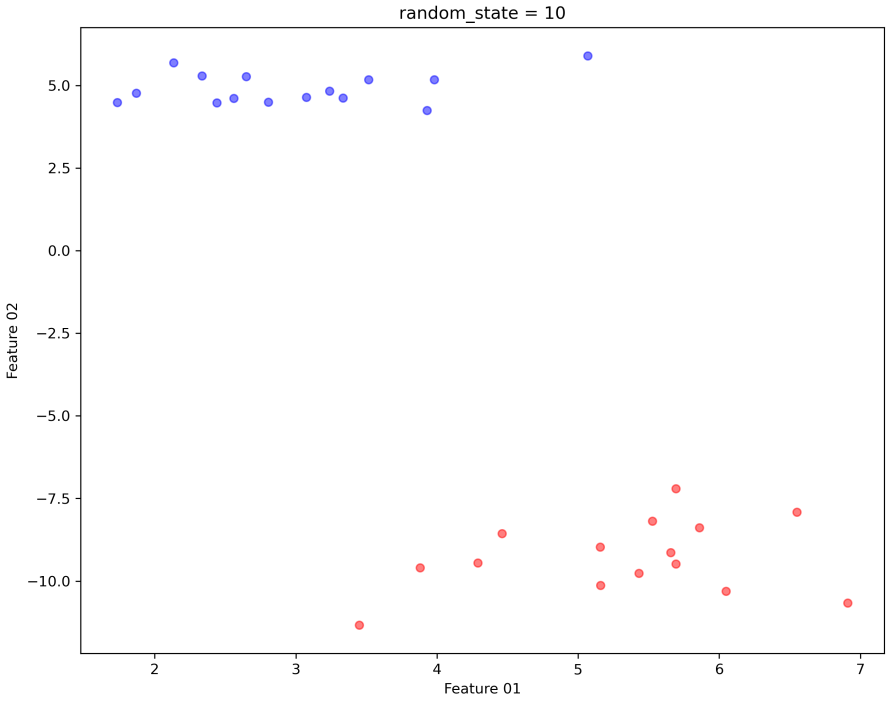
KNN

How to Build a ML Model



Step 01: Dataset Preparation

- In this illustration, we use 'make_blobs' to create a forged dataset
- 'make_blobs' can create data points of different classes
  - generate isotropic Gaussian blobs for clustering

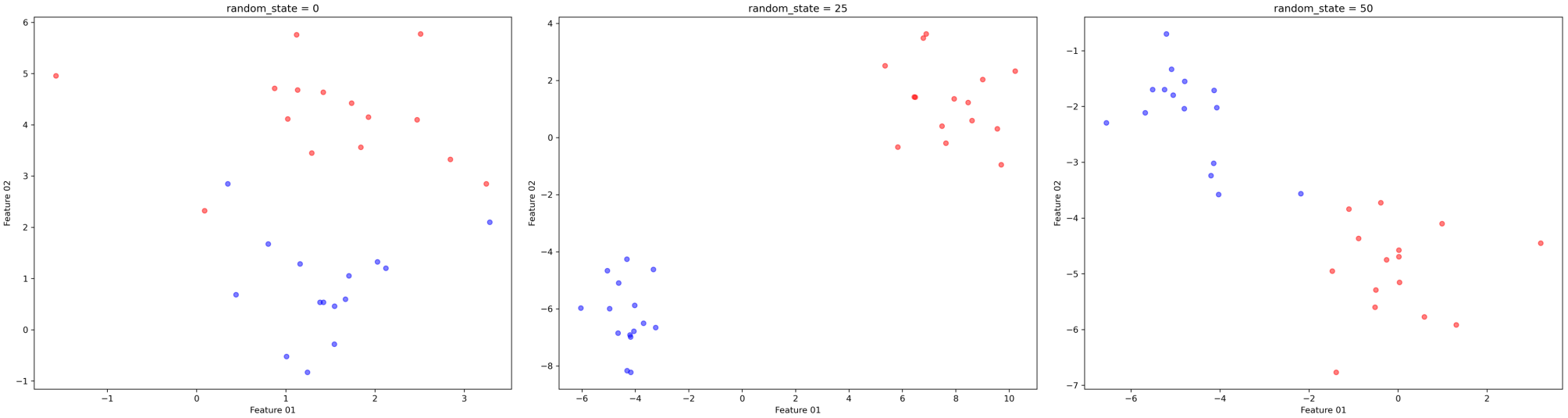


Step 01: Dataset Preparation

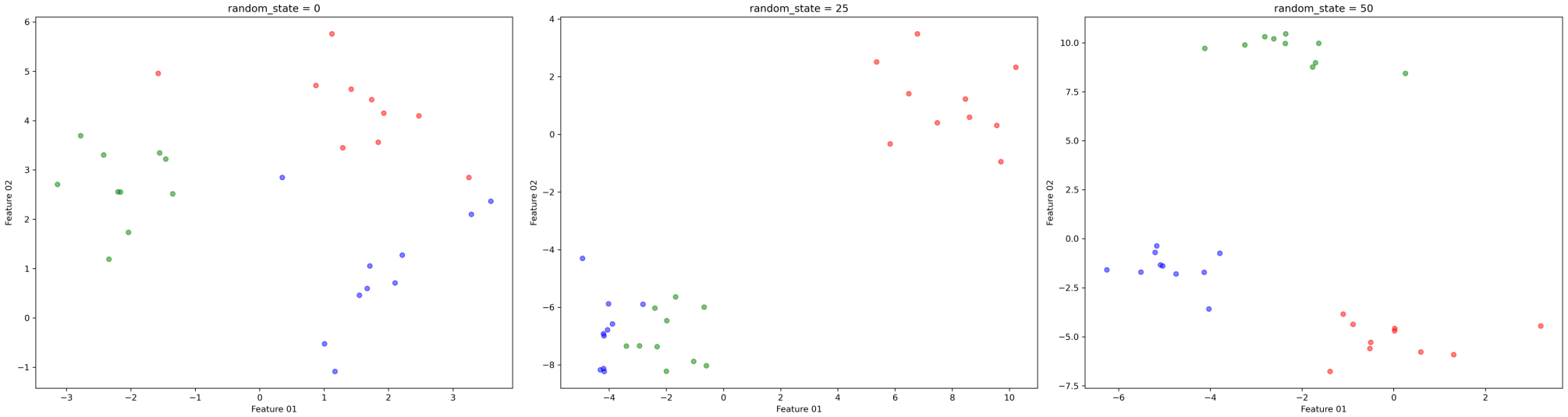
```
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples = 100, n_features = 2, centers = 2, cluster_std = 1.2, random_state = 20)
```

- X, y are the data points and the corresponding class labels, respectively
- n_samples specifies how many data points are generated
- n_features is the dimension of data points
- centers is the number of classes
- cluster_std determines how disperse the data points are
- random_state determines how to generate data points randomly

Step 01: Dataset Preparation



Step 01: Dataset Preparation



Step 02: Training/Testing Dataset

- Training dataset helps to determine the parameters in ML model
  - to perceive how the built ML model performs when facing the unknown data points during the training, a small portion of training dataset becomes the validation dataset, which does not participate in training ML model, and acts as the unknown data points during the training
- Testing dataset helps to understand how ML model functions when facing unknown data points

Step 02: Training/Testing Dataset

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 10)
```

- X_train, X_test are the training dataset and the training dataset and y_train and y_test are the corresponding targets
- X, y are the data points and the corresponding targets, respectively
- test_size specifies the portion of dataset that should be in the testing dataset

Step 03: Build/Train ML Model

- In scikit-learn, the class called 'KNeighborsClassifier' is used to build a KNN model
- Since 'KNeighborsClassifier' is a class, the model is initialized through the declaration of its object

```
from sklearn.neighbors import KNeighborsClassifier
KNN_Model = KNeighborsClassifier(n_neighbors = 3)
```

  - n_neighbors determines the value of k in KNN model

Step 03: Build/Train ML Model

- To train KNN model, we simply use the class method 'fit' to fit the model from the training dataset

```
KNN_Model.fit(X_train, y_train)
```

  - in 'fit' method, we simply let X_train and y_train as arguments to call this method

Step 04: Test the ML model

- Predict the class of a new data point

```
KNN_Model.predict([[1,1]])
```

  - simply put this new data point in the argument of 'predict' method
  - the output looks like

```
array([1])
```

- Predict the classes of a dataset

```
KNN_Model.predict(X_test)
```

  - use the dataset as the argument to call 'predict' method
  - the output looks like

```
array([1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1])
```

Step 04: Test the ML model

- The performance of KNN model can be accessed by calling 'score' method
- The 'score' method returns the accuracy of KNN model of a dataset

```
KNN_Model.score(X_test, y_test)
```

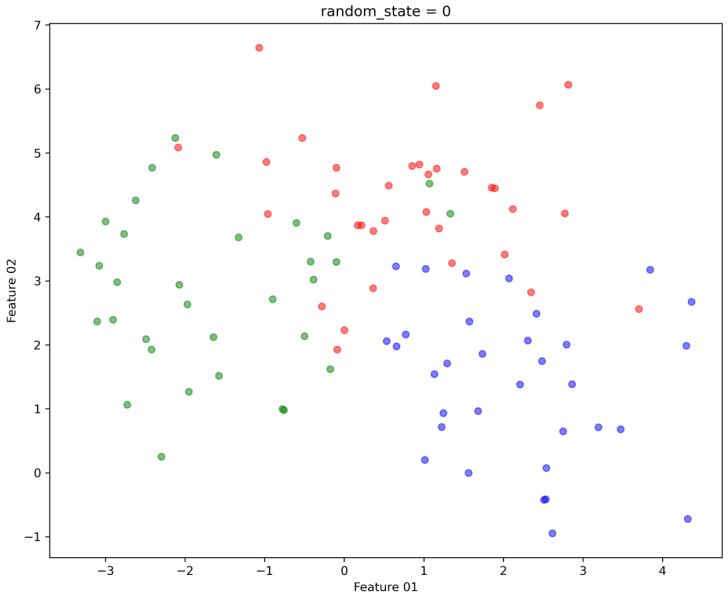
  - X_test and y_test are its arguments
  - it returns the testing accuracy

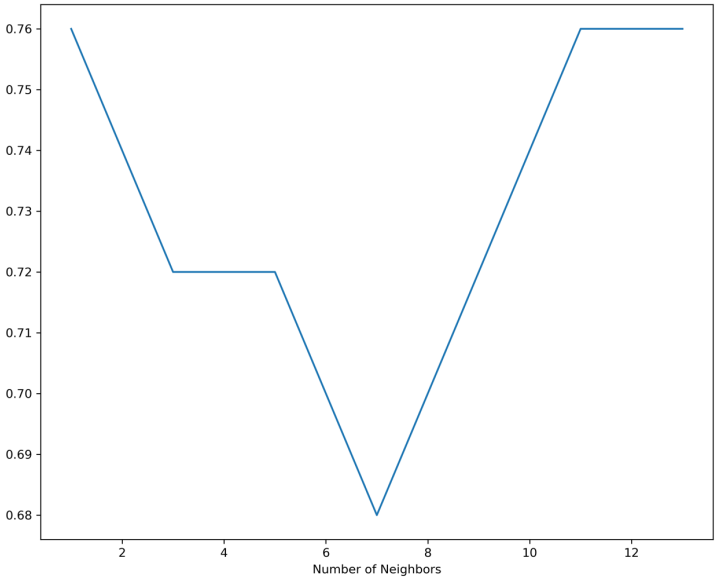
```
KNN_Model.score(X_train, y_train)
```

  - it returns the training accuracy

Discussion

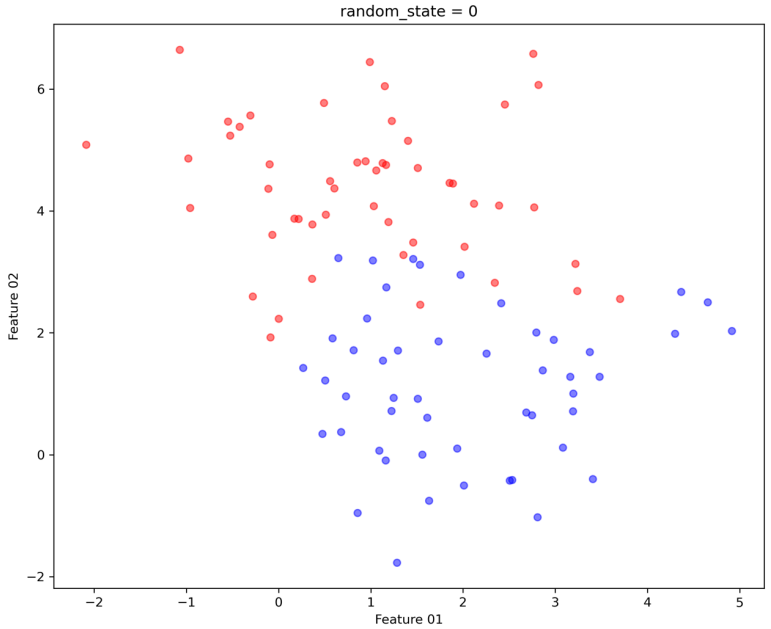
- The best value of k in KNN model should be determined

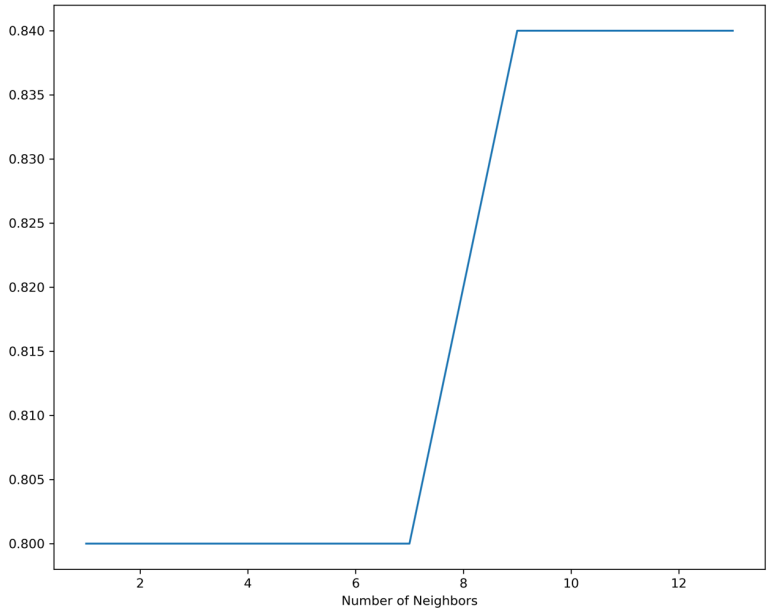




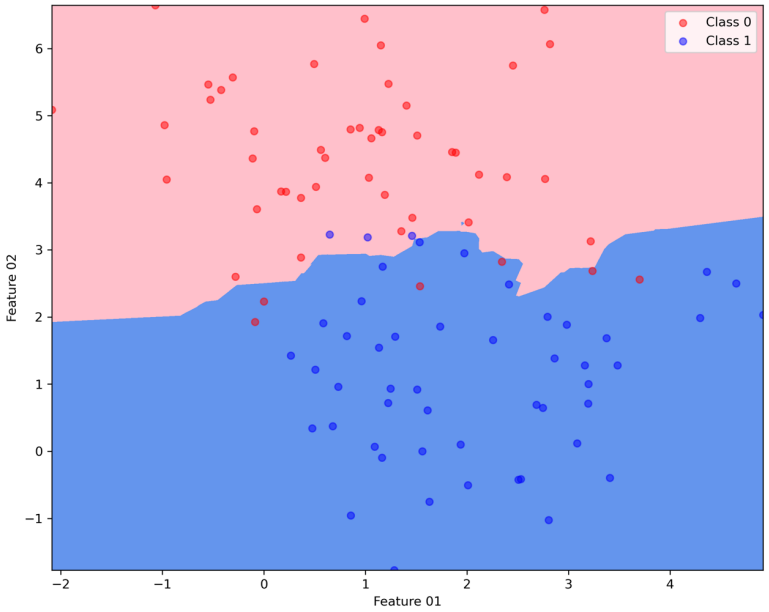
Hints of Implementation

- The best value of k in KNN model should be determined





Summarization



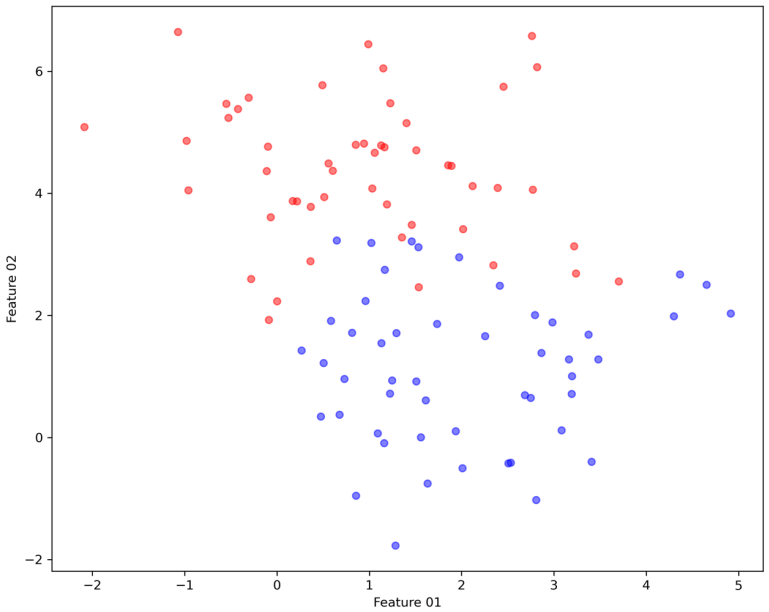
Implementation Using Scikit-Learn

Topic 02

Decision Tree

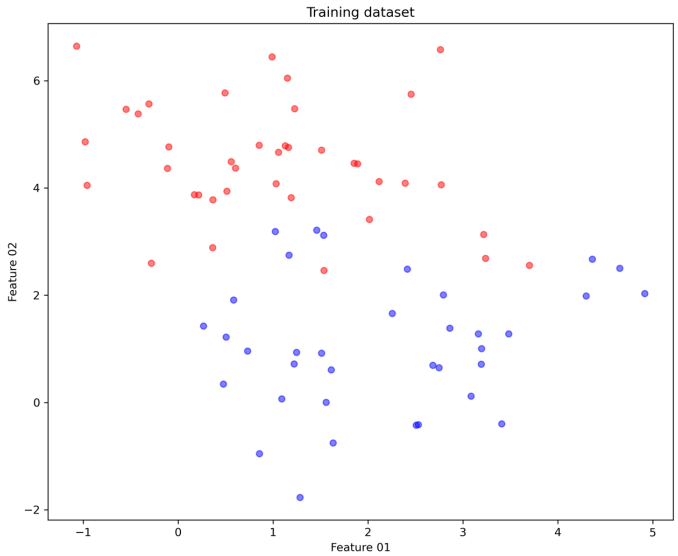
Step 01: Dataset Preparation

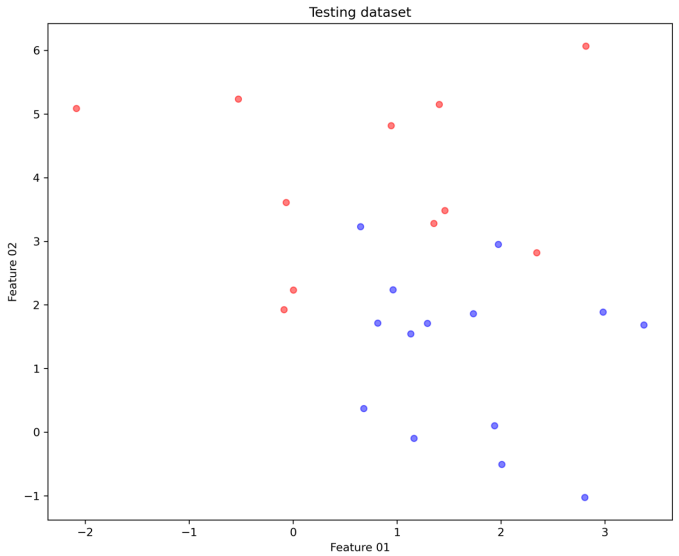
```
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples = 100, n_features = 2, centers = 2, cluster_std = 1.2, random_state = 20)
```



Step 02: Training/Testing Dataset

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 10)
```





Step 03: Build/Train ML Model

- In scikit-learn, the class 'DecisionTreeClassifier' builds the decision tree to fit the dataset
- We need to declare an object of 'DecisionTreeClassifier' to initialize the settings of the decision tree

```
from sklearn.tree import DecisionTreeClassifier
DecisionTree_Model = DecisionTreeClassifier(criterion='gini', max_depth=1, min_samples_split=2, min_samples_leaf=1)
```

  - criterion can be 'gini', 'entropy' and 'log_loss'
  - max_depth, min_samples_split and min_samples_leaf are three arguments to prevent the model from overfitting

Step 03: Build/Train ML Model

- To train the decision model, we simply call the class method ‘fit’

```
DecisionTree_Model.fit(X_train, y_train)
```

  - in ‘fit’ method, we simply let X_train and y_train as arguments to call this method

Step 04: Test the ML model

- Predict the class of a new data point

```
DecisionTree_Model.predict([[1,1]])
```

  - simply put this new data point in the argument of 'predict' method
  - the output looks like

```
array([1])
```

- Predict the classes of a dataset

```
DecisionTree_Model.predict(X_test)
```

  - use the dataset as the argument to call 'predict' method
  - the output looks like

```
array([1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

Step 04: Test the ML model

- The performance of decision tree model can be accessed by calling 'score' method
- The 'score' method returns the accuracy of decision tree model of a dataset

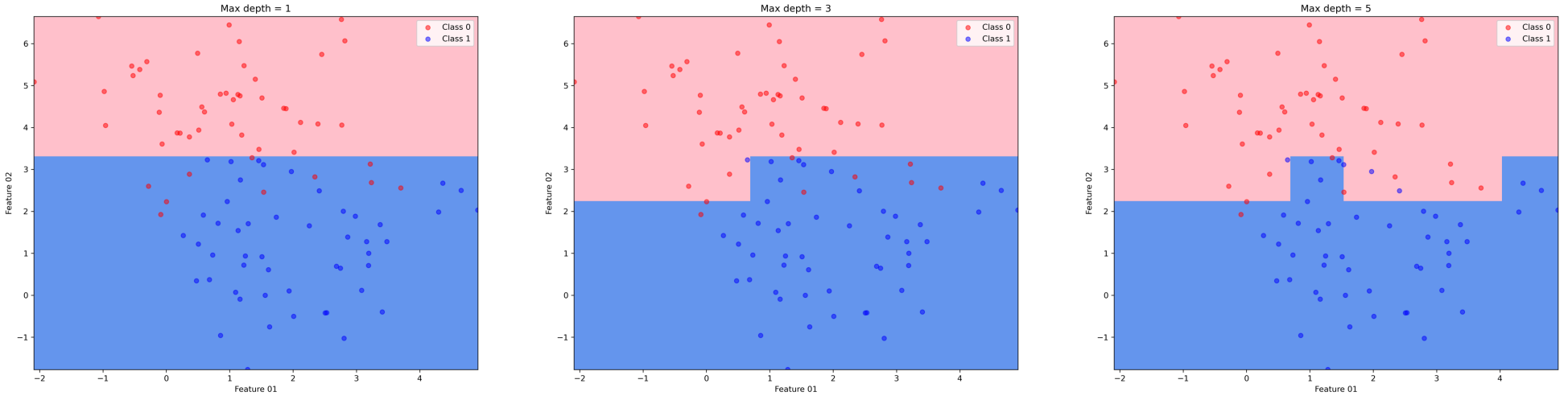
```
DecisionTree _Model.score(X_test, y_test)
```

  - X_test and y_test are its arguments
  - it returns the testing accuracy

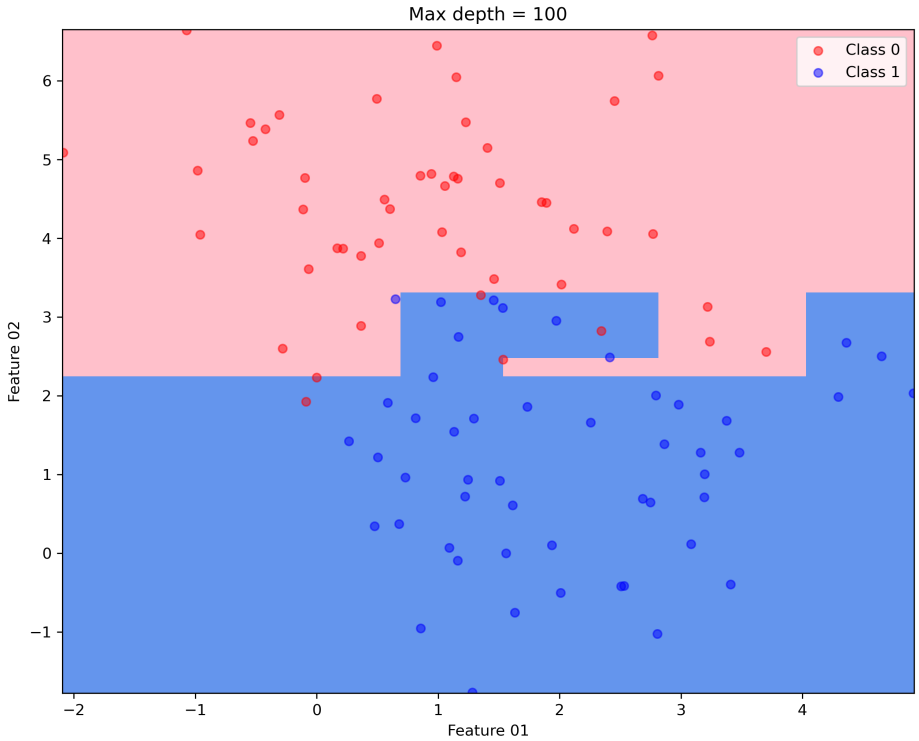
```
DecisionTree _Model.score(X_train, y_train)
```

  - it returns the training accuracy

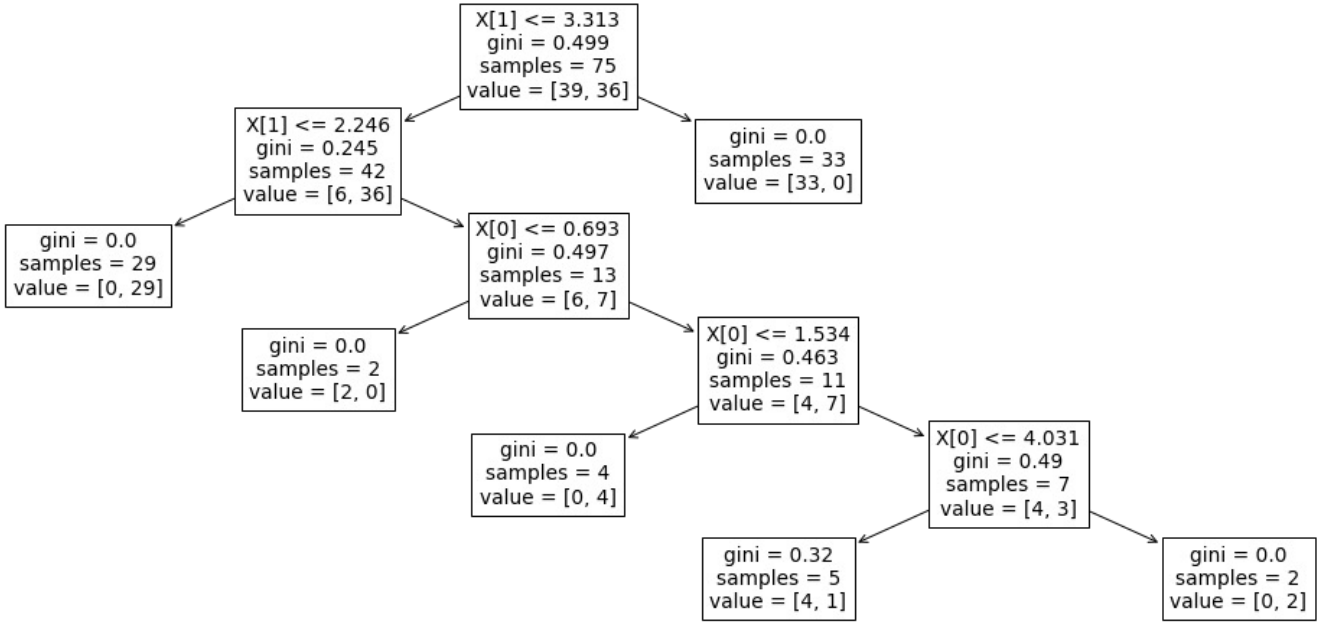
Step 04: Test the ML model



Step 04: Test the ML model

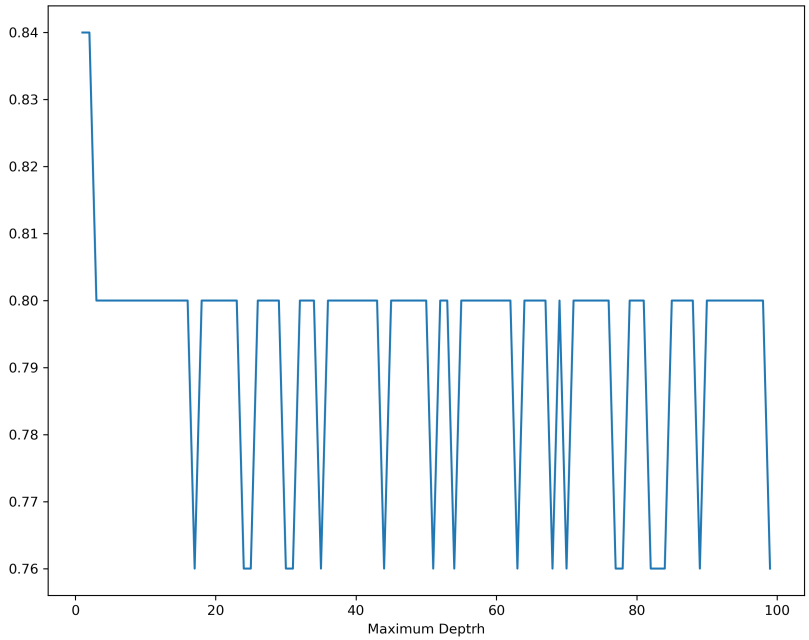


Step 04: Test the ML model

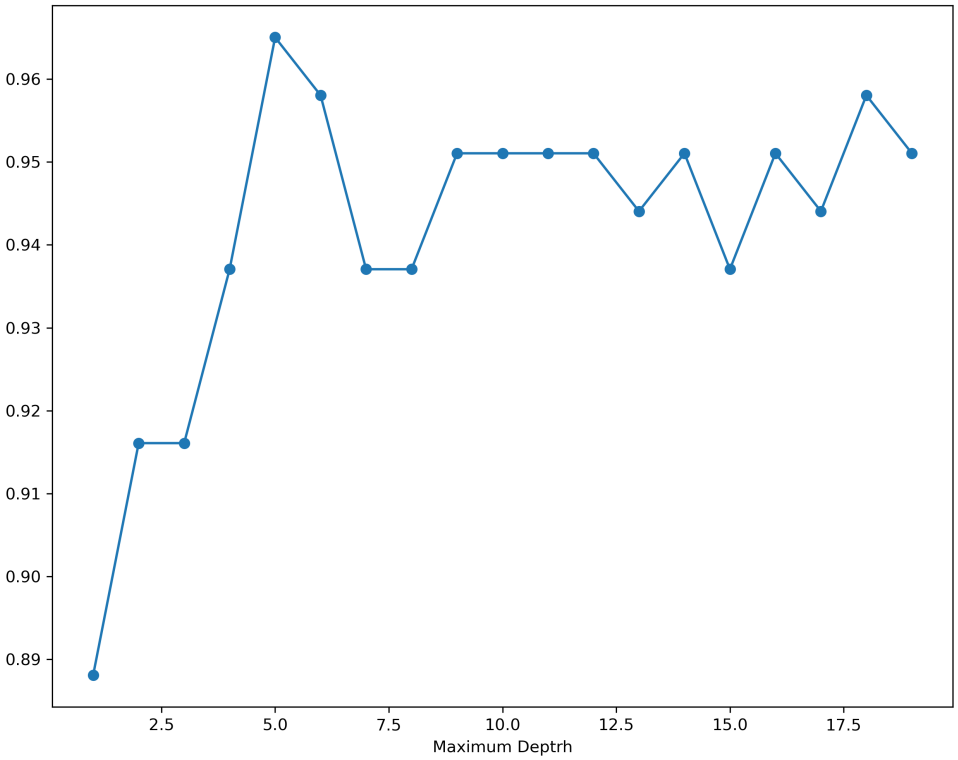


Discussion

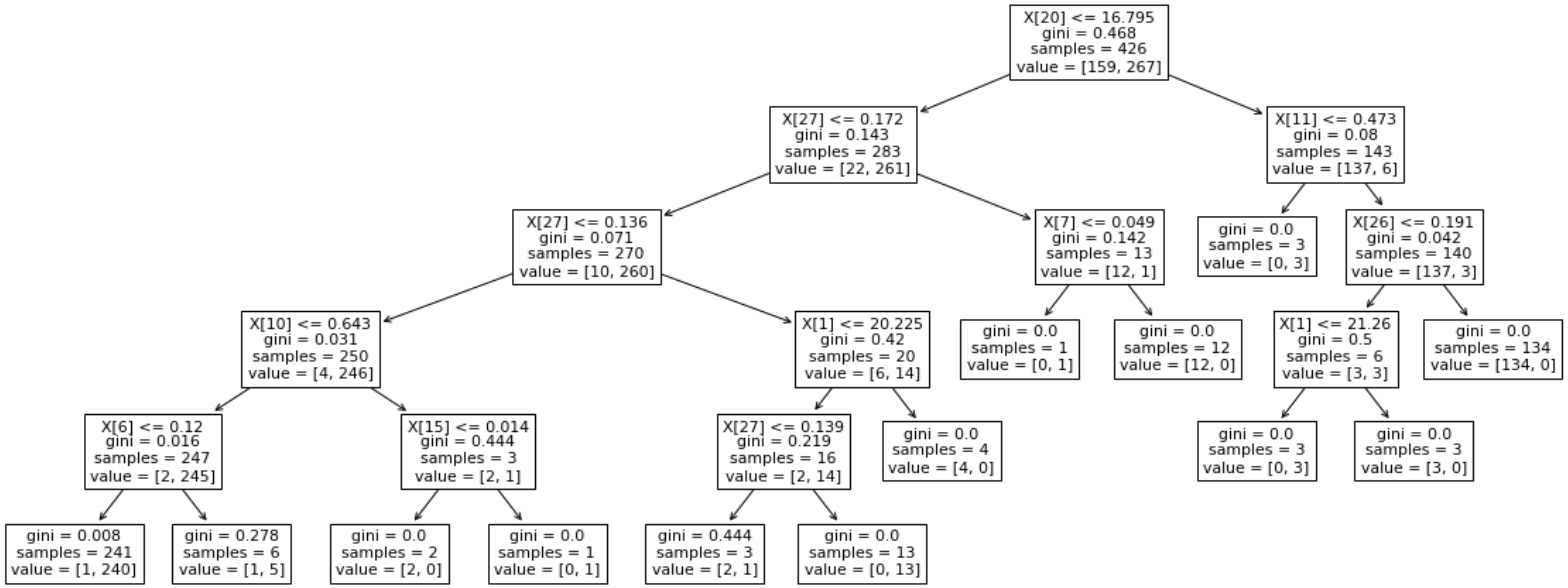
- The maximum depth of decision trees affects the performance



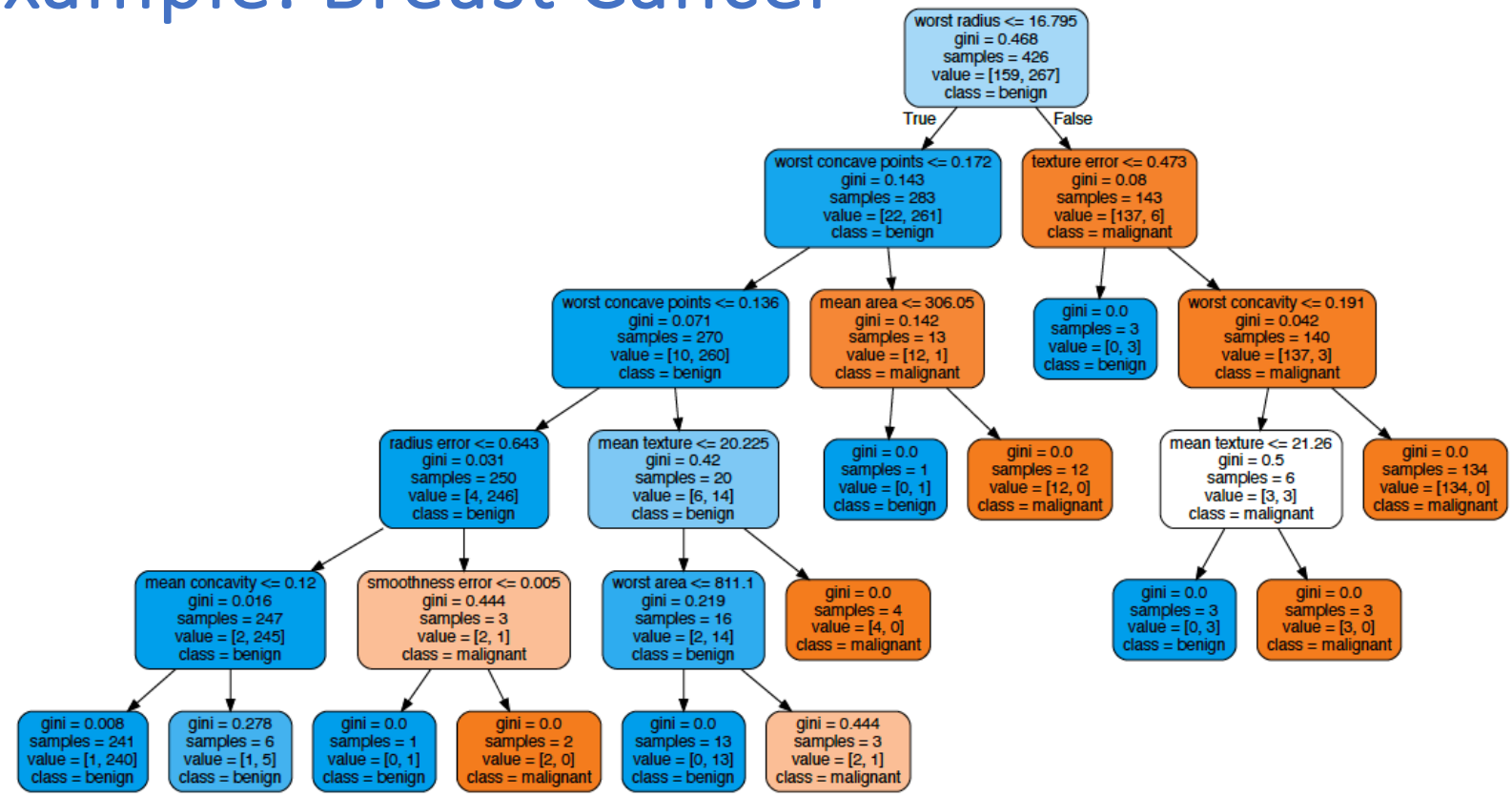
Example: Breast Cancer



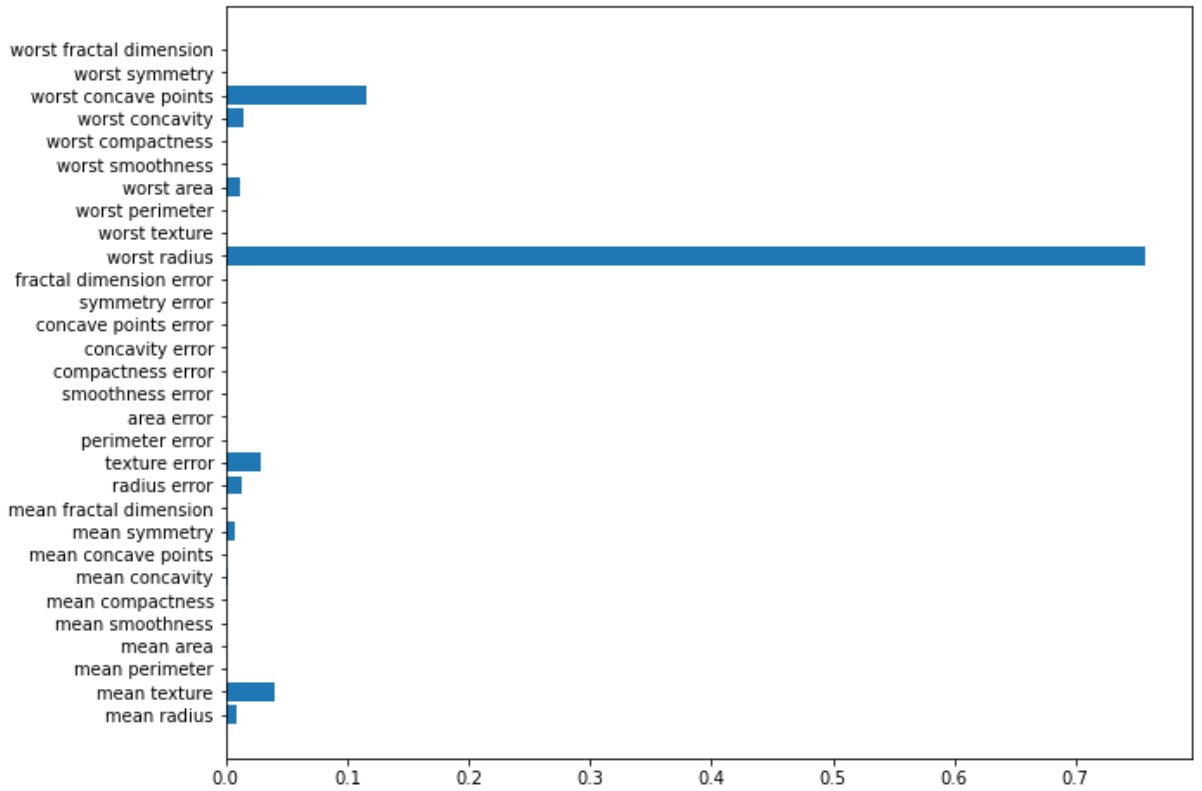
Example: Breast Cancer



Example: Breast Cancer



Example: Breast Cancer



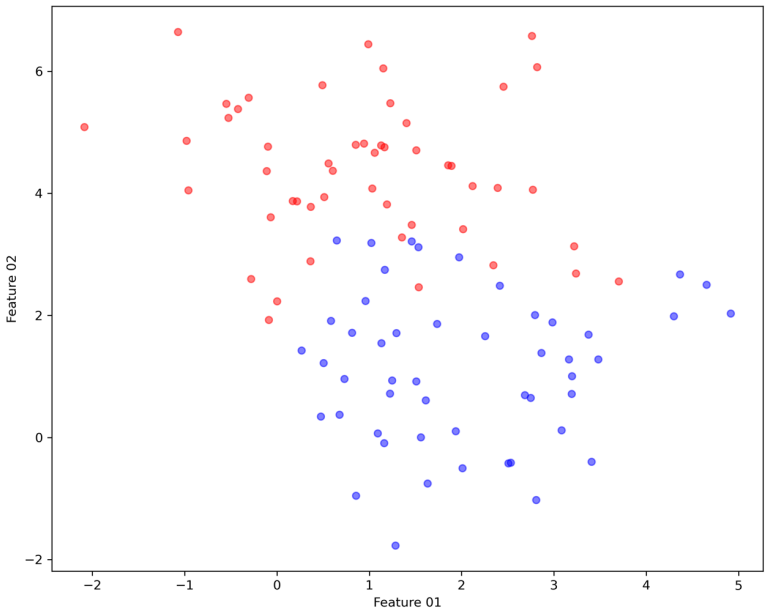
Implementation Using Scikit-Learn

Topic 03

Random Forest

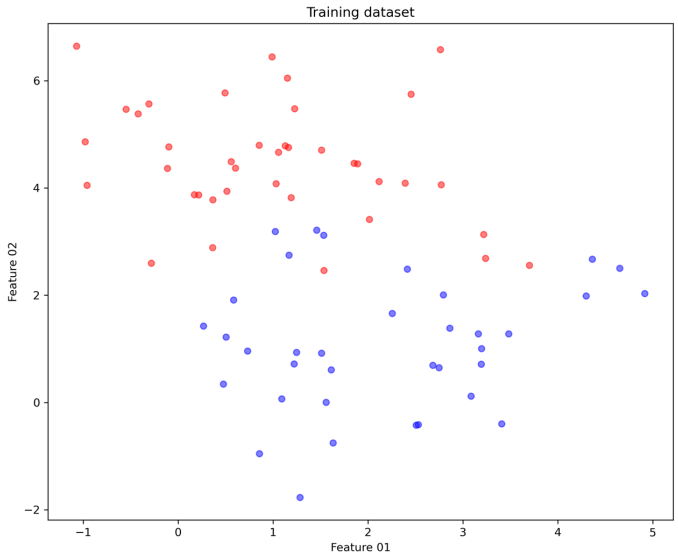
Step 01: Dataset Preparation

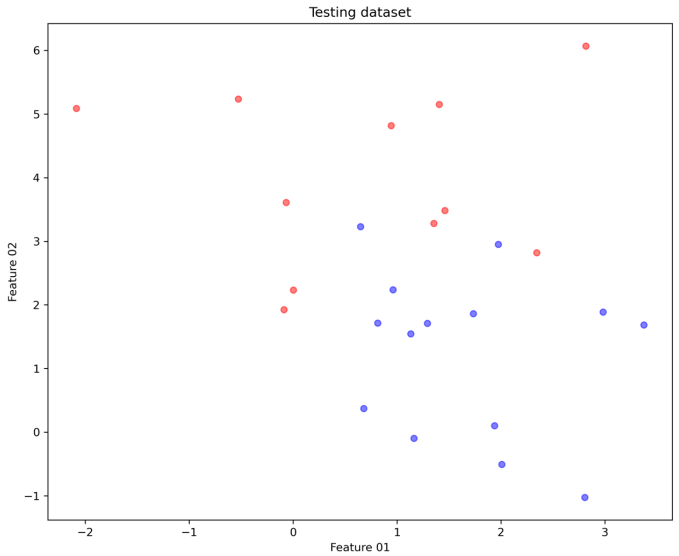
```
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples = 100, n_features = 2, centers = 2, cluster_std = 1.2, random_state = 0)
```



Step 02: Training/Testing Dataset

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 10)
```





Step 03: Build/Train ML Model

- In Scikit-Learn, the model of random forest is built by declaring an object of the class 'RandomForestClassifier'

```
from sklearn.ensemble import RandomForestClassifier
RandomForest_Model = RandomForestClassifier(n_estimators=5, criterion='gini', max_depth=2)
```

  - n_estimators is to specify how many different decision trees are grown
  - criterion can be 'gini', 'entropy' or 'log_loss'
  - max_depth, min_samples_split and min_samples_leaf are also the arguments to prevent the model from overfitting

Step 03: Build/Train ML Model

- The class function ‘fit’ is called to train the model

```
RandomForest_Model.fit(X_train, y_train)
```

  - in ‘fit’ method, we simply let X_train and y_train as arguments to call this method

Step 04: Test the ML model

- We can predict the class of a new data

```
RandomForest_Model.predict([[1,6]])
```

  - simply put this new data point in the argument of 'predict' method
  - the output looks like

```
array([0])
```

- We can also obtain the accuracy of a dataset

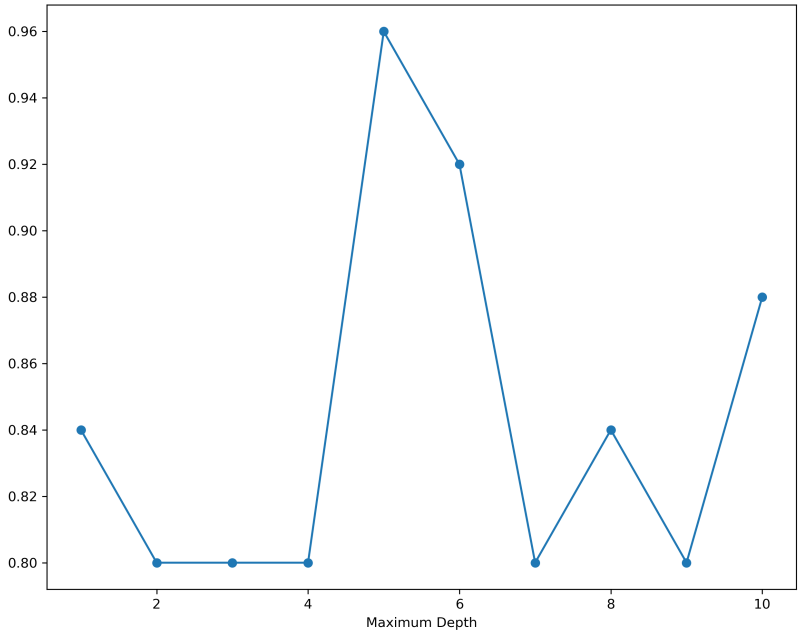
```
RandomForest_Model.score(X_test, y_test)
```

  - simply use the data and its corresponding target as argument to call the method 'score'
  - the output looks like

```
0.84
```

Discussion

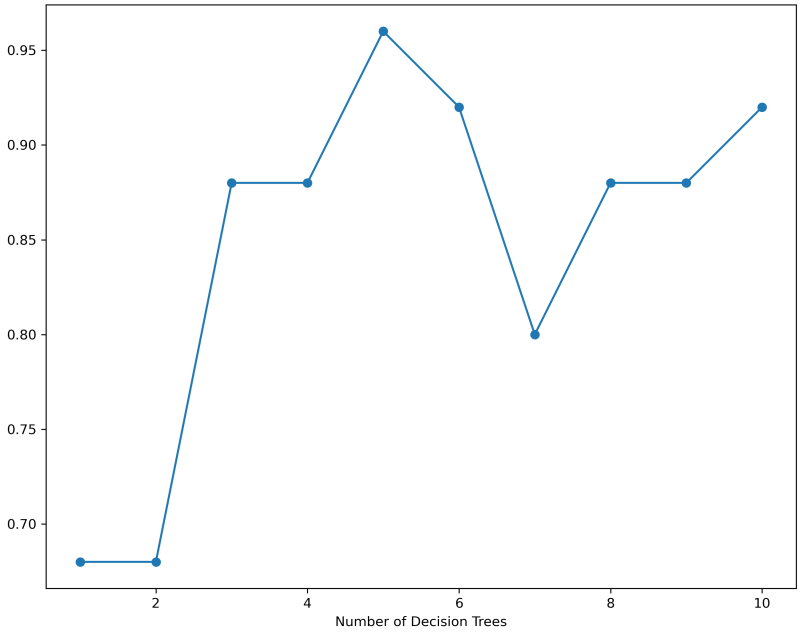
- We need to discuss how the maximum depth of each decision tree in the random forest affect the accuracy



Number of Decision Trees is 5

Discussion

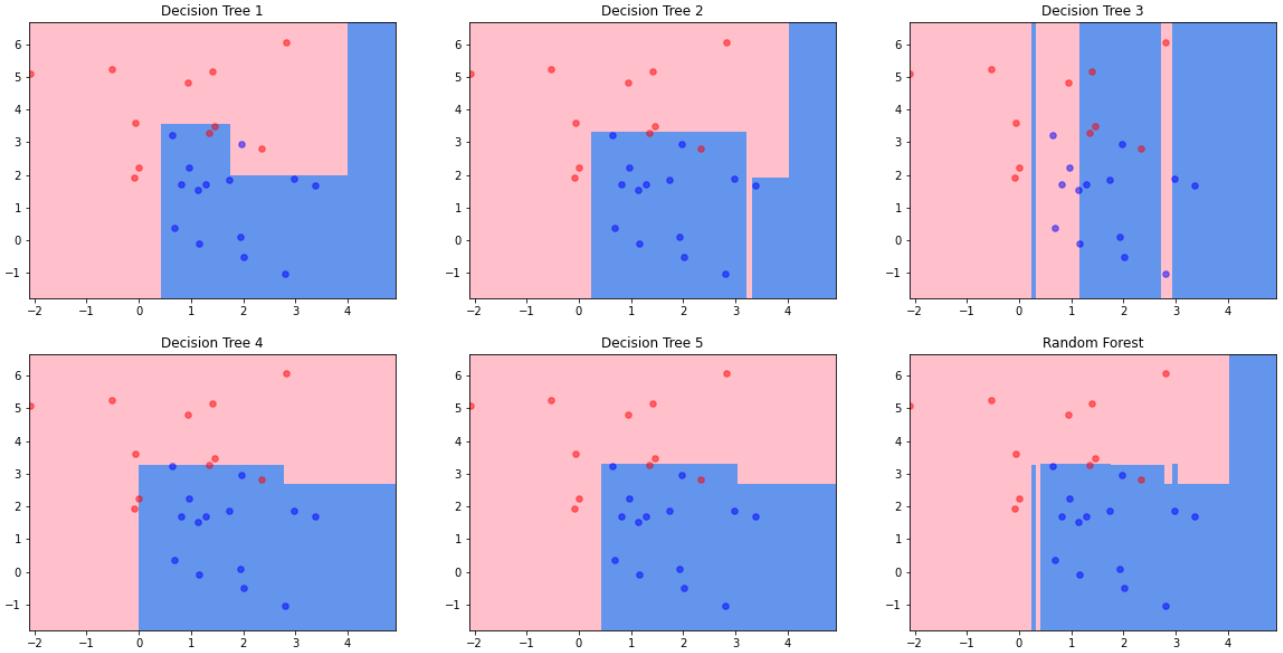
- We also need to understand how the number of decision trees in a random forest affect the performance



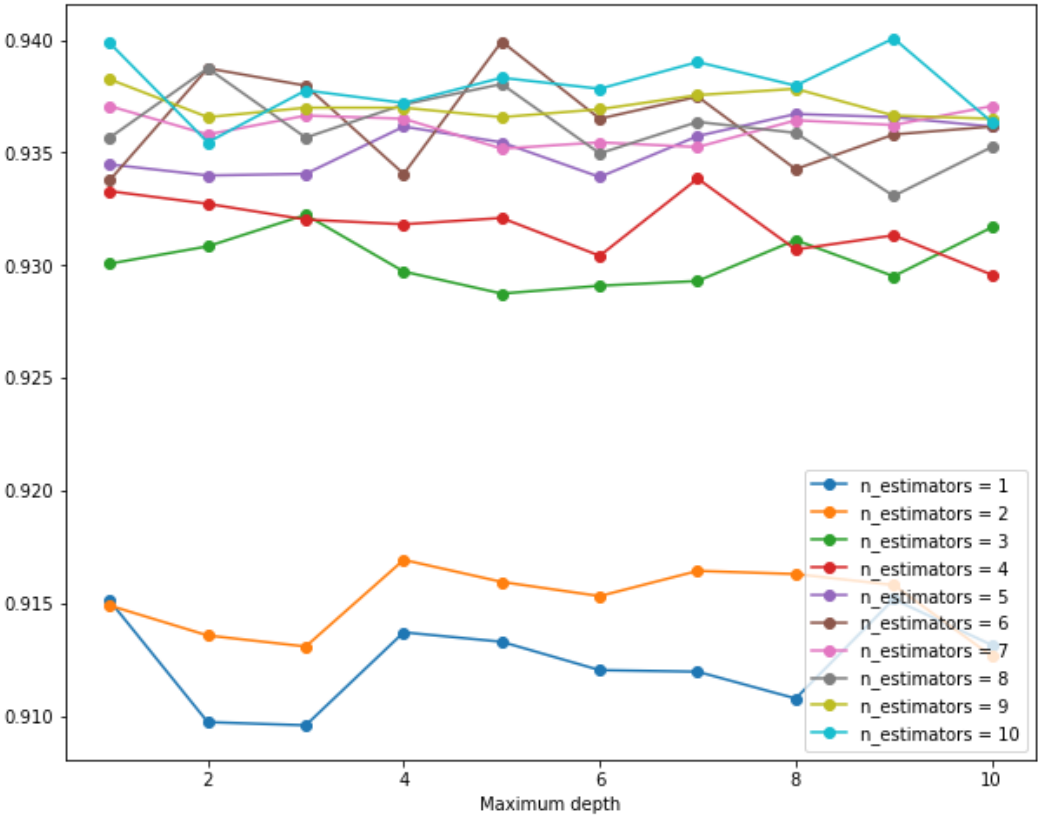
Maximum Depth = 5

Discussion

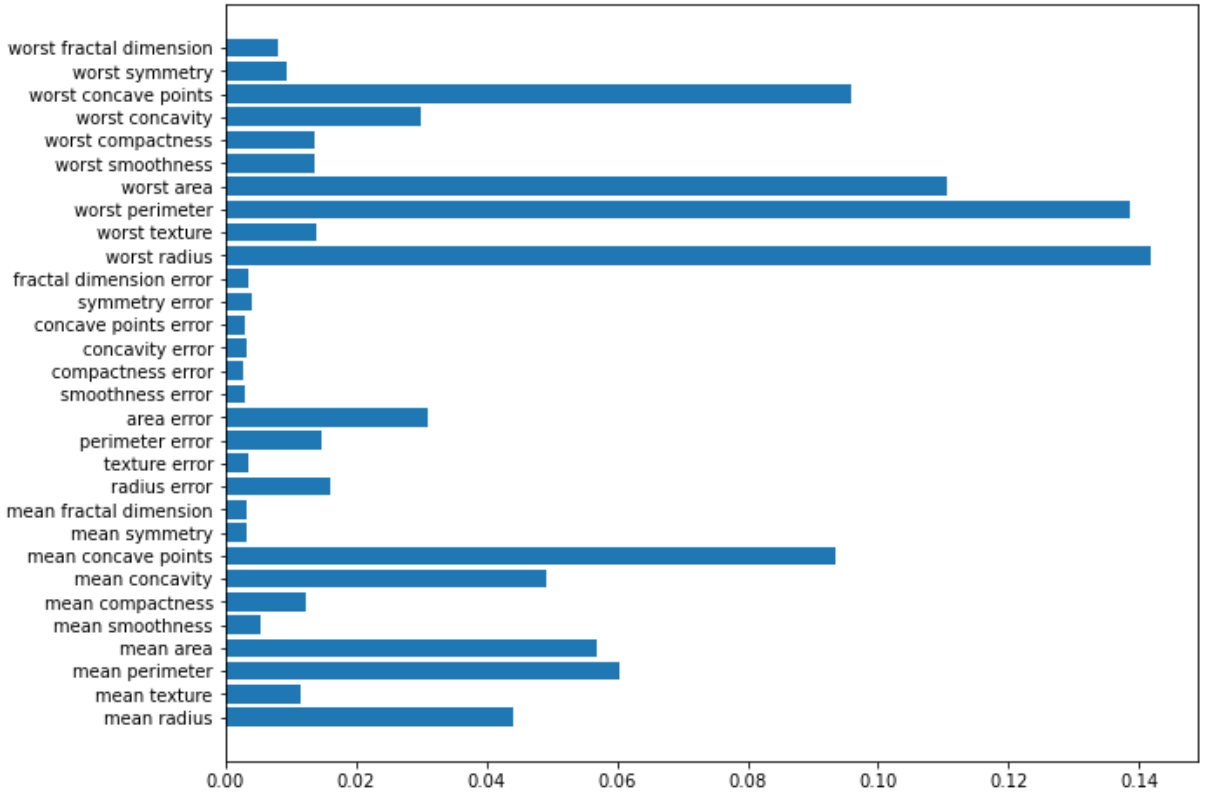
- We are also interested in the decision boundary of a random forest if possible



Example: Breast Cancer



Example: Breast Cancer



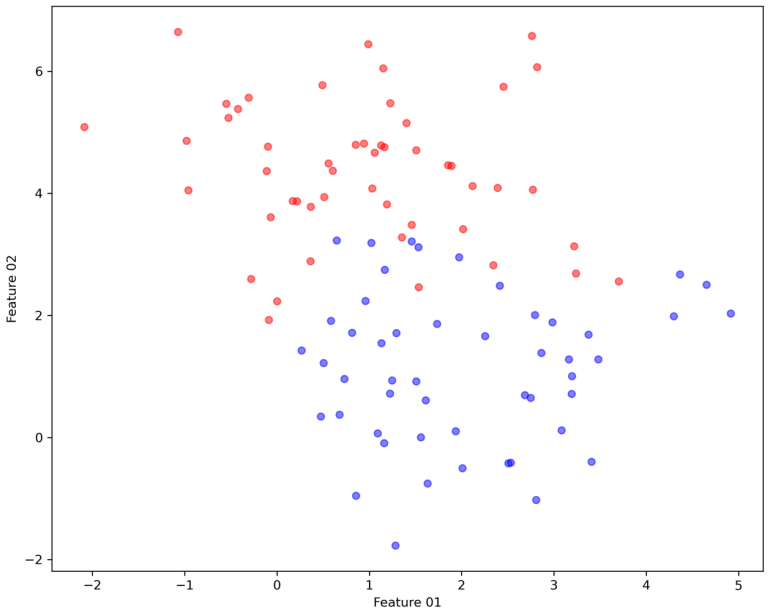
Implementation Using Scikit-Learn

Topic 04

AdaBoost

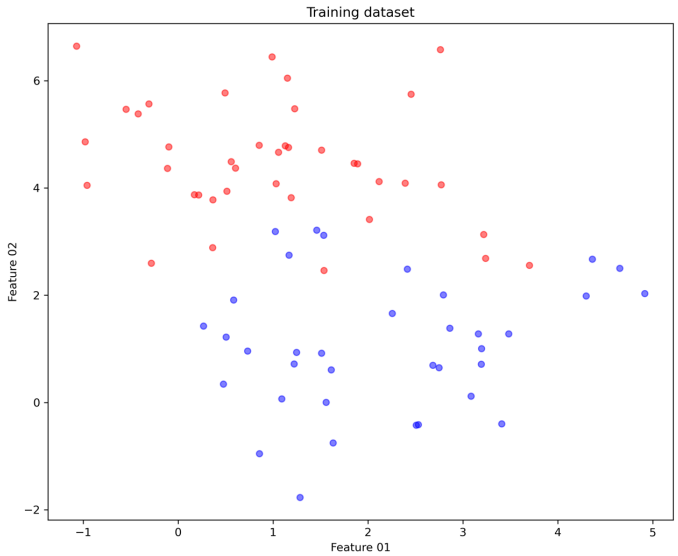
Step 01: Dataset Preparation

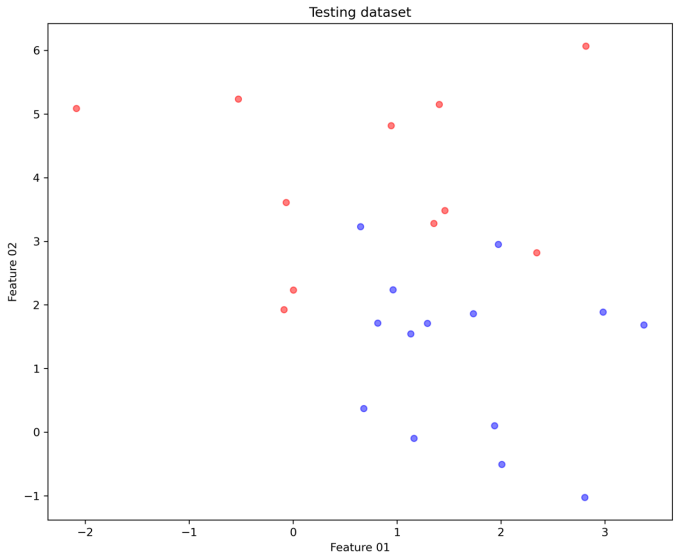
```
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples = 100, n_features = 2, centers = 2, cluster_std = 1.2, random_state = 0)
```



Step 02: Training/Testing Dataset

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 10)
```





Step 03: Build/Train ML Model

- In Scikit-Learn, the model of AdaBoost is built by declaring an object of the class 'AdaBoostClassifier'

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
Base_Model = DecisionTreeClassifier(max_depth=1)
AdaBoost_Model = AdaBoostClassifier(base_estimator=Base_Model, n_estimators=5, learning_rate=1.0)
```

  - base_estimator specifies the base model and the default model is decision tree with maximum depth equal to 1
  - n_estimators is to specify how many different decision trees are grown
  - learning_rate is to the weight applied to each classifier at each boosting iteration

Step 03: Build/Train ML Model

- The class function ‘fit’ is called to train the model

```
AdaBoost_Model.fit(X_train, y_train)
```

  - in ‘fit’ method, we simply let X_train and y_train as arguments to call this method

Step 04: Test the ML model

- We can predict the class of a new data

```
AdaBoost_Model.predict([[6,5]])
```

  - simply put this new data point in the argument of 'predict' method
  - the output looks like

```
array([0])
```

- We can also obtain the accuracy of a dataset

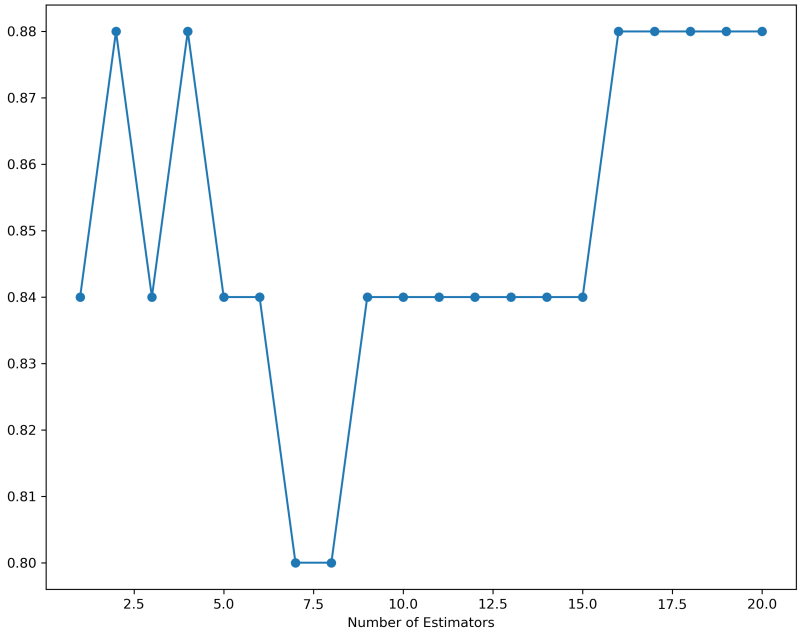
```
AdaBoost_Model.score(X_test, y_test)
```

  - the arguments in score are the data and its corresponding target
  - the output looks like

```
0.84
```

Discussion

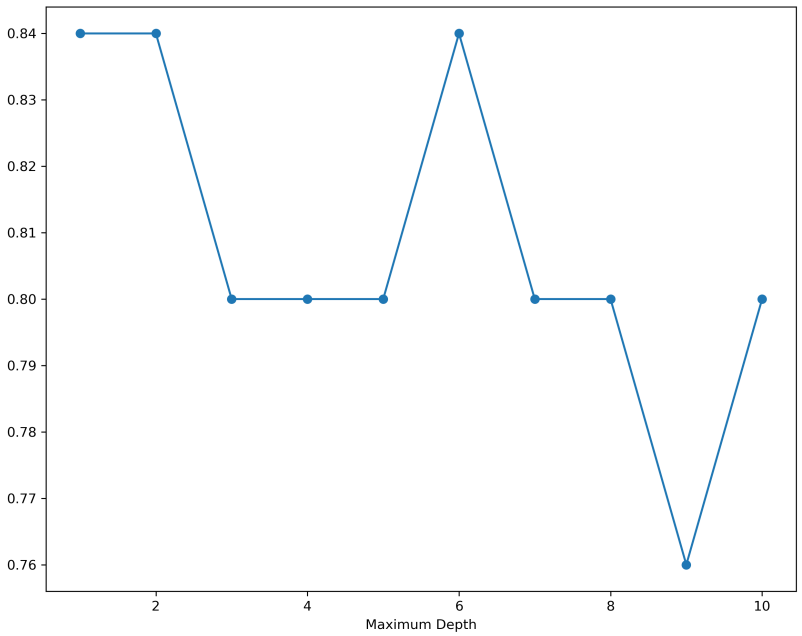
- We need to discuss how the number of estimators in the AdaBoost affect the accuracy



The base model is the decision tree with maximum depth equal to 1

Discussion

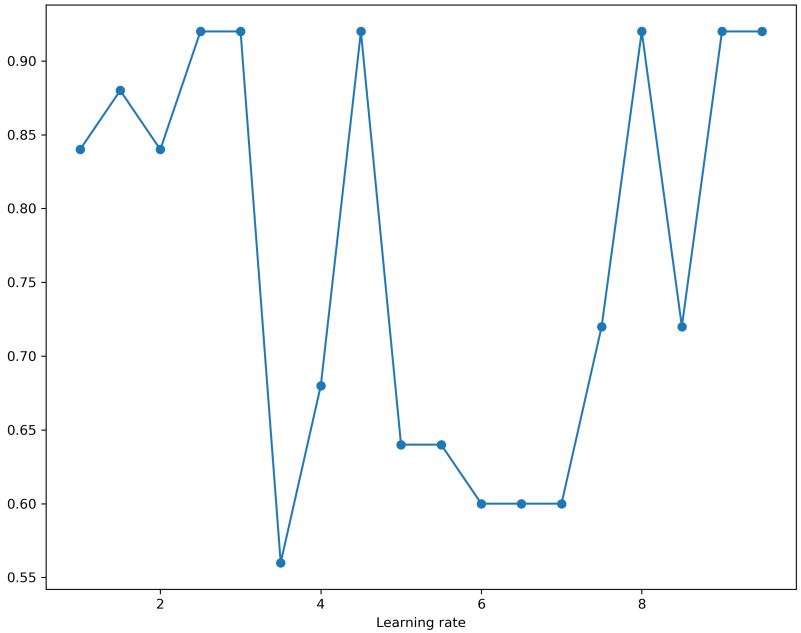
- We need to discuss how the maximum depth of the base model in the AdaBoost affects the accuracy



Number of Decision Trees grown is 5

Discussion

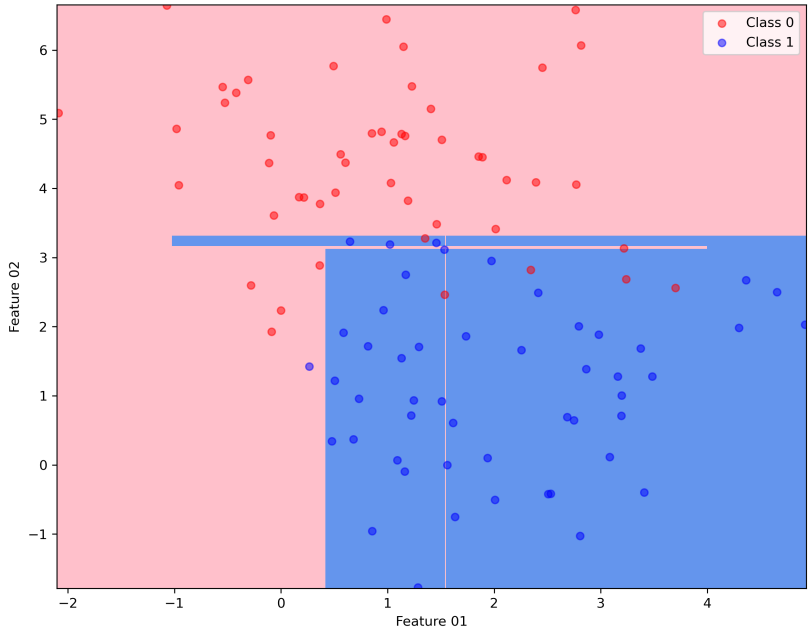
- We need to discuss how the learning rate at each step in the AdaBoost affects the accuracy



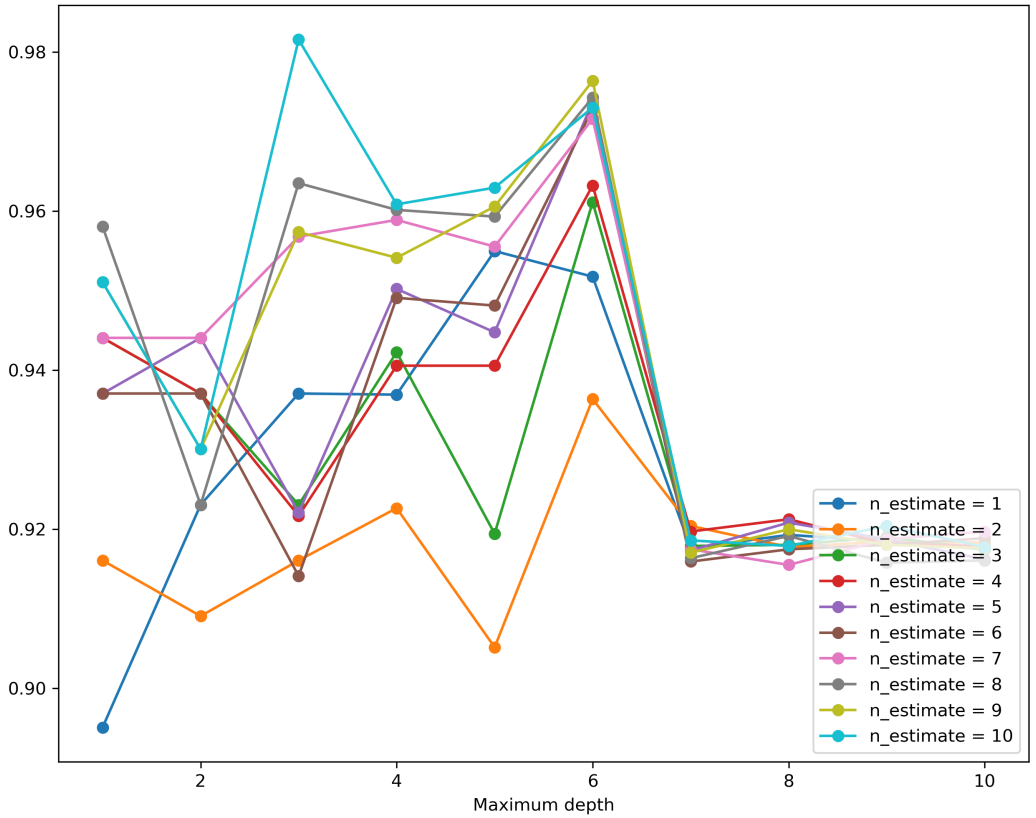
The base model is the decision tree with maximum depth equal to 2

Discussion

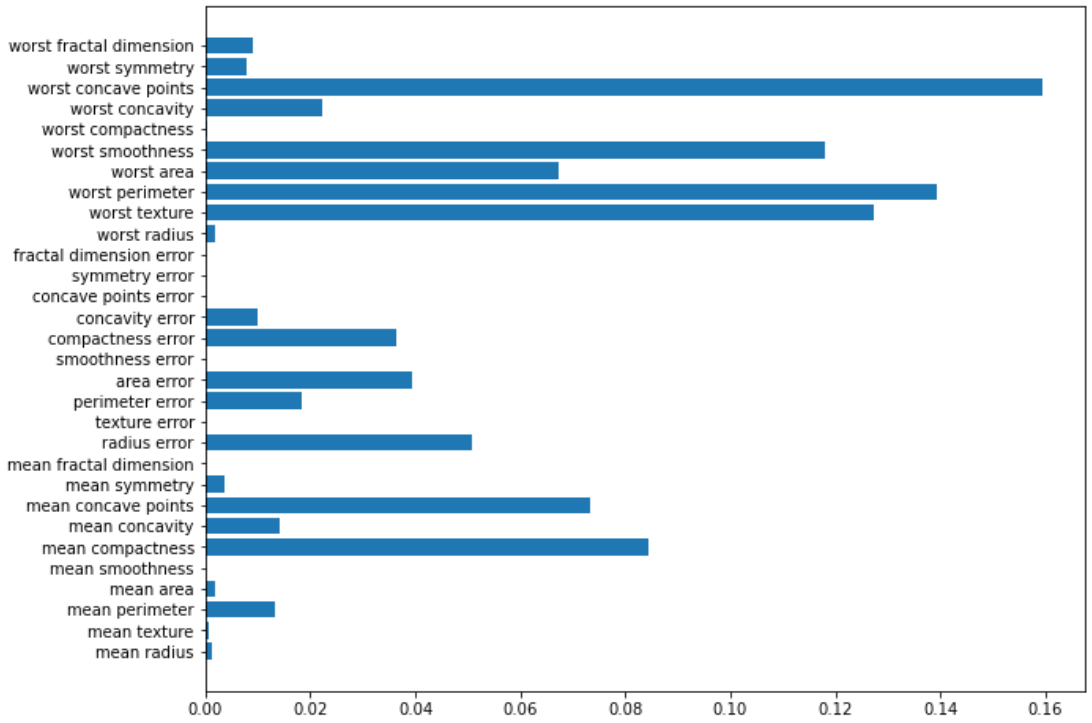
- The decision boundary of AdaBoost



Example: Breast Cancer



Example: Breast Cancer



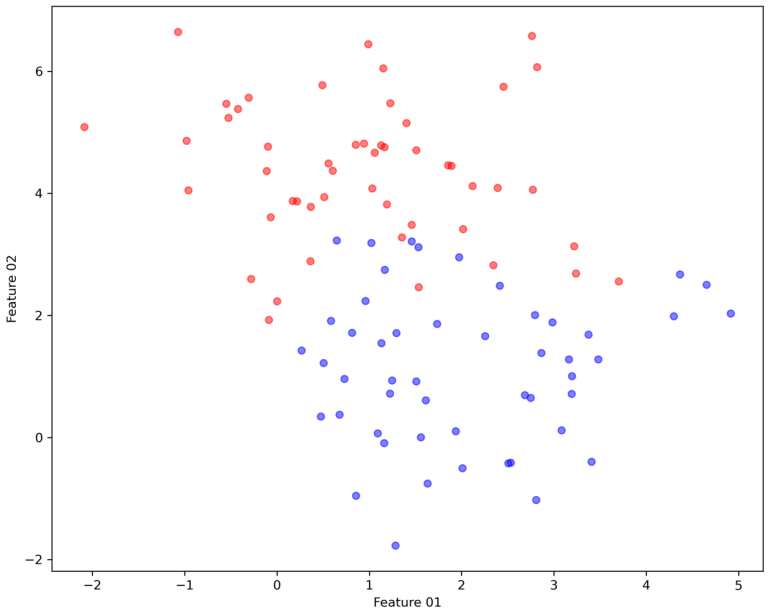
Implementation Using Scikit-Learn

Topic 05

Gradient Boosting

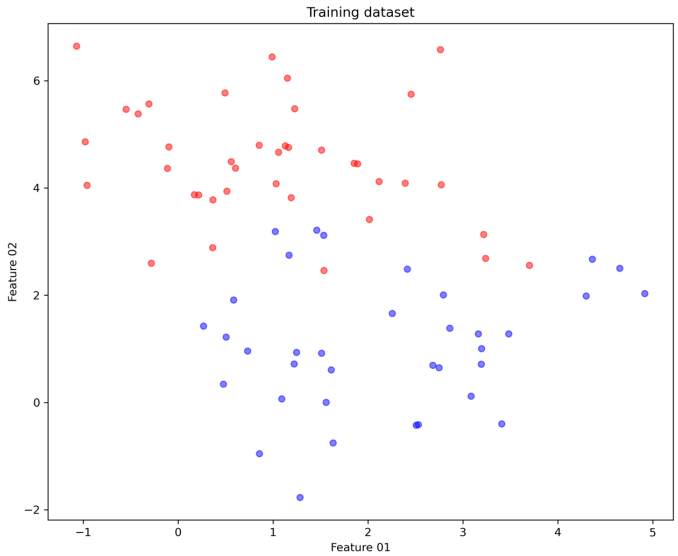
Step 01: Dataset Preparation

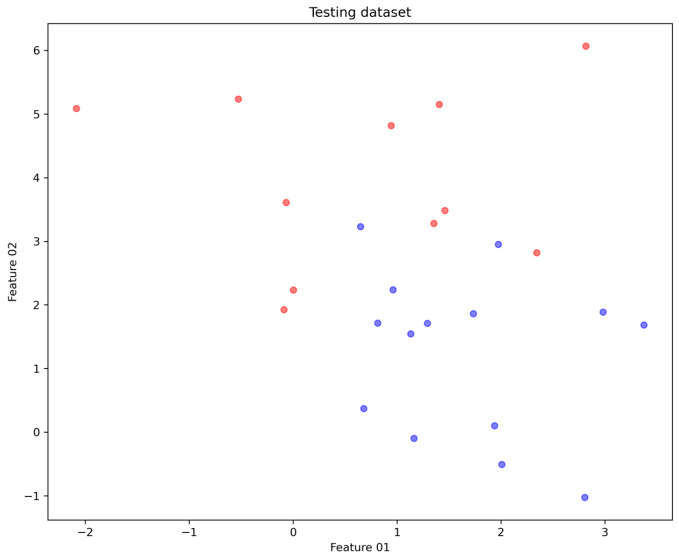
```
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples = 100, n_features = 2, centers = 2, cluster_std = 1.2, random_state = 0)
```



Step 02: Training/Testing Dataset

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 10)
```





Step 03: Build/Train ML Model

- In Scikit-Learn, the model of gradient boosting is built by declaring an object of the class 'GradientBoostingClassifier'

```
from sklearn.ensemble import GradientBoostingClassifier
GradientBoosting_Model = GradientBoostingClassifier(loss='log_loss', learning_rate=0.1, n_estimators=5, max_depth=2)
```

  - n_estimators is to specify how many different decision trees are grown
  - learning_rate is to the weight applied to each classifier at each boosting iteration
  - loss can be 'log_loss' or 'exponential'
    - when using 'exponetital', gradient boosting recovers the AdaBoost algorithm

Step 03: Build/Train ML Model

- The class function ‘fit’ is called to train the model

```
GradientBoosting_Model.fit(X_train, y_train)
```

  - in ‘fit’ method, we simply let X_train and y_train as arguments to call this method

Step 04: Test the ML model

- We can predict the class of a new data

```
GradientBoosting_Model.predict([[4,5]])
```

  - simply put this new data point in the argument of 'predict' method
  - the output looks like

```
array([0])
```

- We can also obtain the accuracy of a dataset

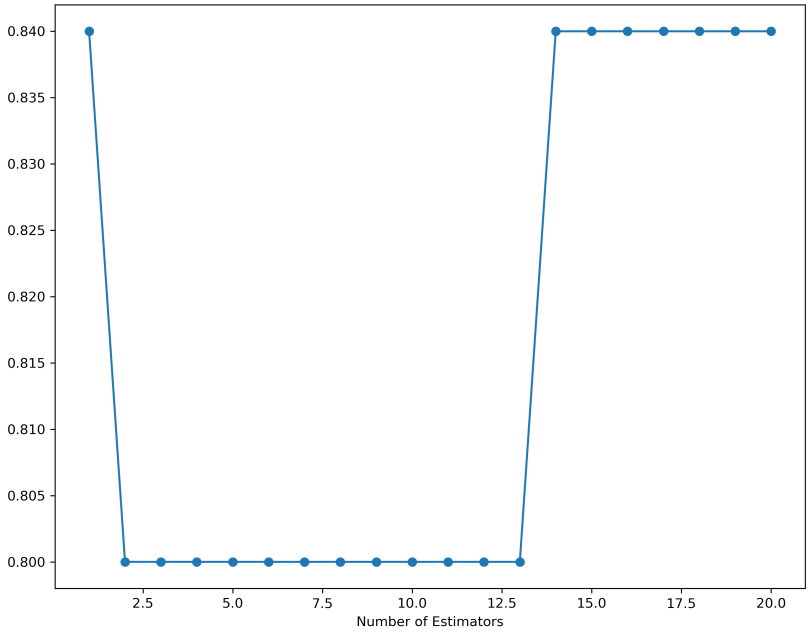
```
GradientBoosting_Model.score(X_test, y_test)
```

  - the arguments in score are the data and its corresponding target
  - the output looks like

```
0.84
```

Discussion

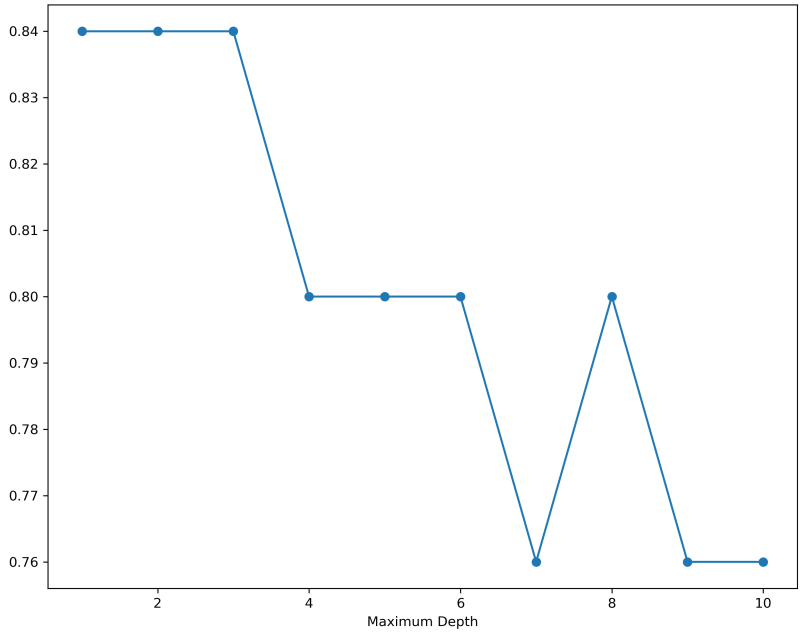
- We need to discuss how the number of estimators in the gradient boosting affects the accuracy



Maximum depth equals to 3

Discussion

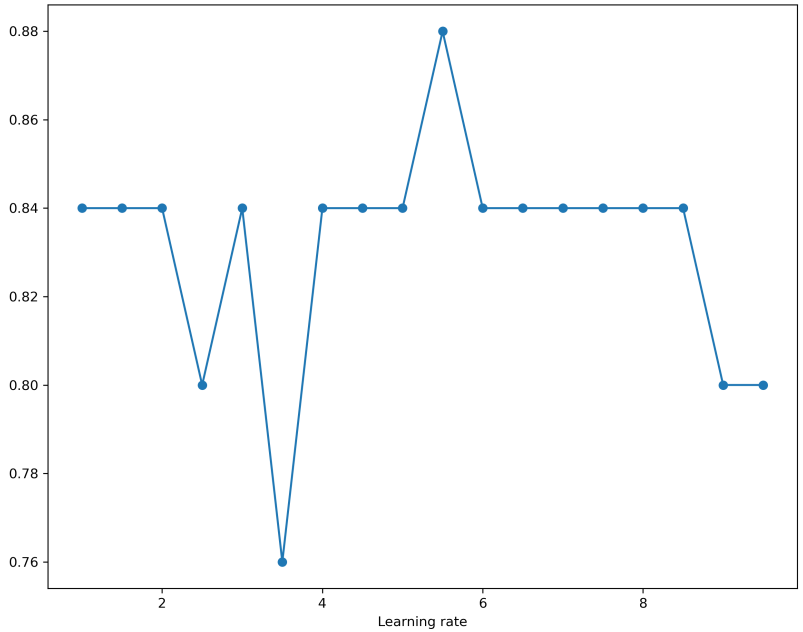
- We need to discuss how the maximum depth affects the accuracy



Number of Decision Trees grown is 14

Discussion

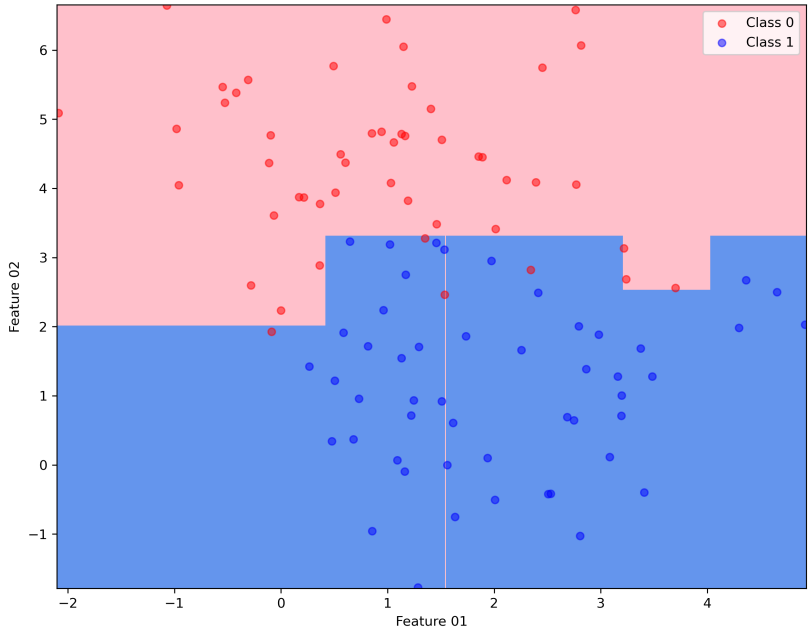
- We need to discuss how the learning rate at each step in the AdaBoost affect the accuracy



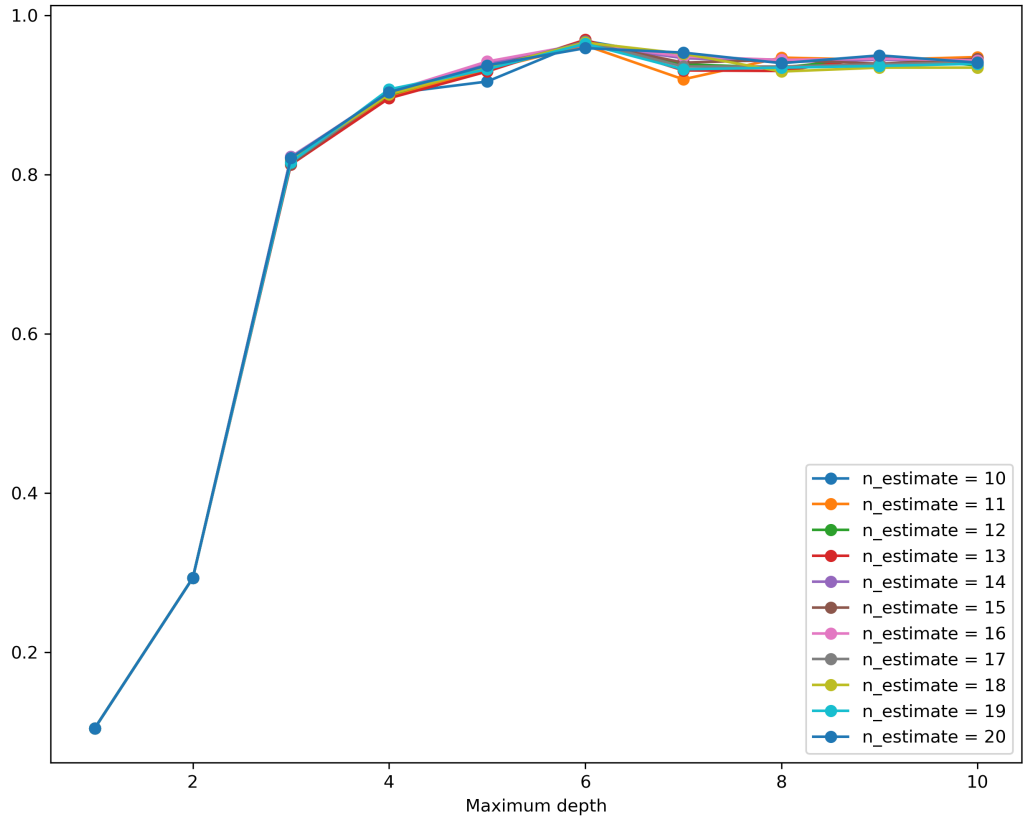
Number of Decision Trees grown is 14 and maximum depth is equal to 3

Discussion

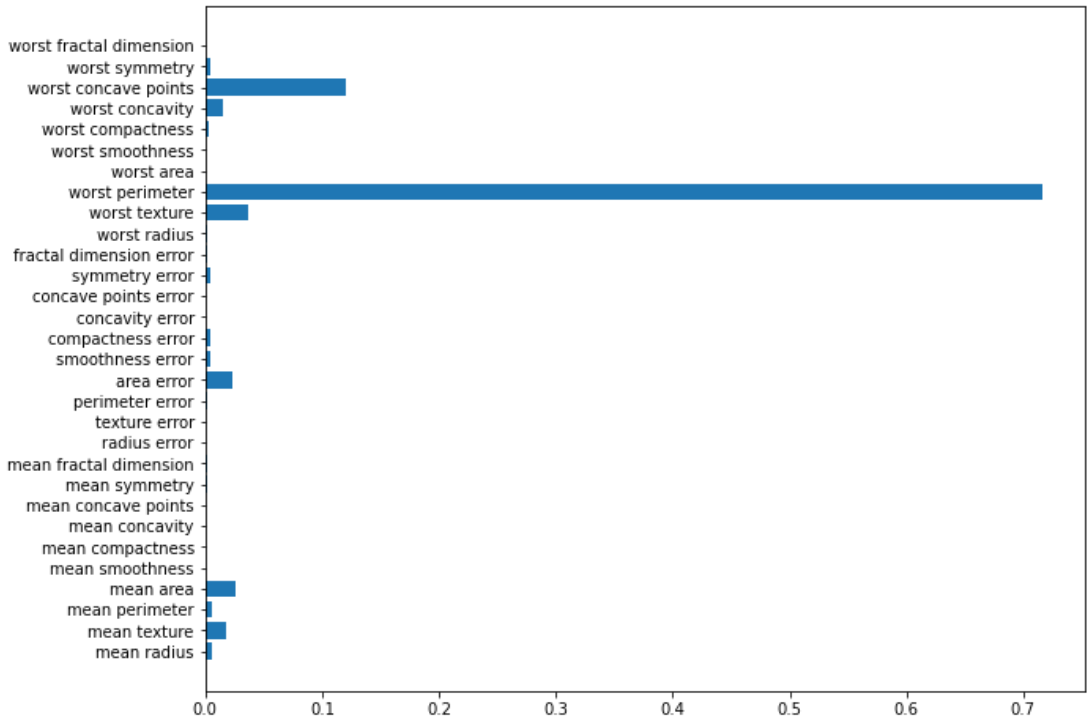
- The decision boundary of the gradient boosting



Example: Breast Cancer



Example: Breast Cancer



Q&A