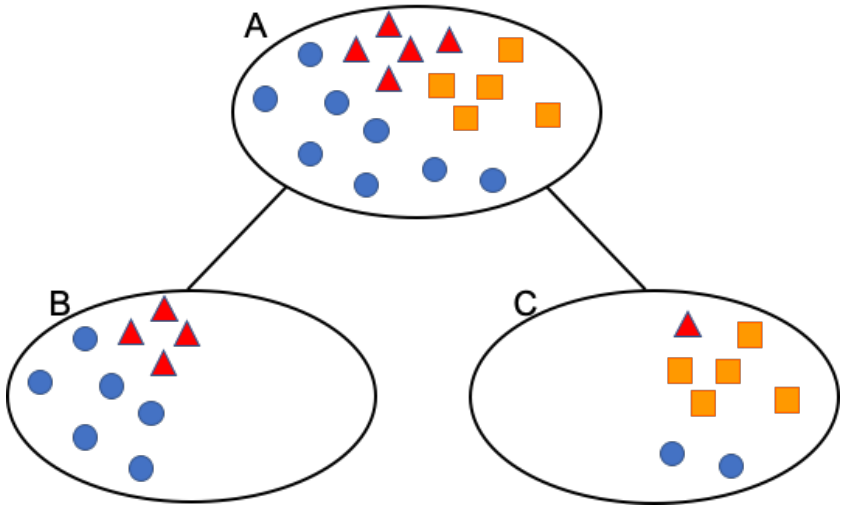
EE2211 Tutorial 9

Dr Feng LIN

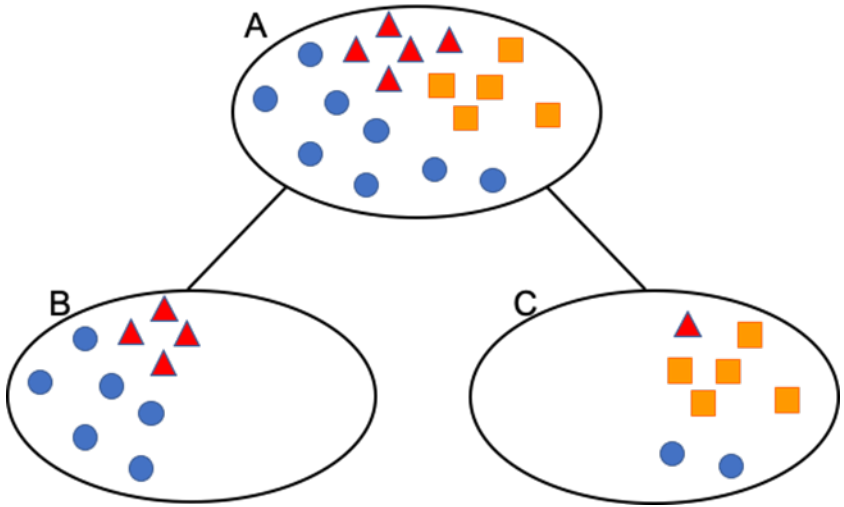
Q1

Compute the Gini impurity, entropy, misclassification rate for nodes A, B and C, as well as the overall metrics (Gini impurity, entropy misclassification error) at depth 1 of the decision tree shown below.



Q1

Let's assume class 1, class 2 and class 3 correspond to red triangles, orange squares and blue circles respectively.



- For node A, $p_1 = \frac{5}{18}, p_2 = \frac{5}{18}, p_3 = \frac{8}{18} = \frac{4}{9}$
- For node B, $p_1 = \frac{4}{10} = \frac{2}{5}, p_2 = \frac{0}{10} = 0, p_3 = \frac{6}{10} = \frac{3}{5}$
- For node C, $p_1 = \frac{1}{8}, p_2 = \frac{5}{8}, p_3 = \frac{2}{8} = \frac{1}{4}$
- For **Gini impurity**, recall formula is $1 - \Sigma_{i=1}^{K} p_i^2$
- Node A: $1 - \left(\frac{5}{18}\right)^2 - \left(\frac{5}{18}\right)^2 - \left(\frac{4}{9}\right)^2 = 0.6481$
- Node B: $1 - \left(\frac{2}{5}\right)^2 - (0)^2 - \left(\frac{3}{5}\right)^2 = 0.48$
- Node C: $1 - \left(\frac{1}{8}\right)^2 - \left(\frac{5}{8}\right)^2 - \left(\frac{1}{4}\right)^2 = 0.5312$
- Overall Gini at depth 1: $\left(\frac{10}{18}\right) 0.48 + \left(\frac{8}{18}\right) 0.5312 = 0.5028$
- Observe the decrease in Gini impurity from root (0.6481) to depth 1 (0.5028)

Q1

- For **entropy**, recall formula is $-\sum_i p_i \log_2 p_i$
- Node A: $-\left(\frac{5}{18}\right)\log_2\left(\frac{5}{18}\right)-\left(\frac{5}{18}\right)\log_2\left(\frac{5}{18}\right)-\left(\frac{4}{9}\right)\log_2\left(\frac{4}{9}\right)=1.5466$
- Node B: $-\left(\frac{2}{5}\right)\log_2\left(\frac{2}{5}\right)-(0)\log_2(0)-\left(\frac{3}{5}\right)\log_2\left(\frac{3}{5}\right)=0.9710$
- Node C: $-\left(\frac{1}{8}\right)\log_2\left(\frac{1}{8}\right)-\left(\frac{5}{8}\right)\log_2\left(\frac{5}{8}\right)-\left(\frac{1}{4}\right)\log_2\left(\frac{1}{4}\right)=1.2988$
- Overall entropy at depth 1: $\left(\frac{10}{18}\right)\ 0.9710\ +\ \left(\frac{8}{18}\right)1.2988=1.1167$
- Observe the decrease in entropy from root (1.5466) to depth 1 (1.1167)

Q1

- For **misclassification rate**, recall formula is $1 - \max_i p_i$
- Node A: $1 - \max\left(\left(\frac{5}{18}\right), \left(\frac{5}{18}\right), \left(\frac{4}{9}\right)\right) = 1 - \left(\frac{4}{9}\right) = \frac{5}{9} = 0.5556$
- Node B: $1 - \max\left(\left(\frac{2}{5}\right), 0, \left(\frac{3}{5}\right)\right) = 1 - \left(\frac{3}{5}\right) = \frac{2}{5}$
- Node C: $1 - \max\left(\left(\frac{1}{8}\right), \left(\frac{5}{8}\right), \left(\frac{1}{4}\right)\right) = 1 - \left(\frac{5}{8}\right) = \frac{3}{8}$
- Overall misclassification error rate at depth 1: $\left(\frac{10}{18}\right)\left(\frac{2}{5}\right) + \left(\frac{8}{18}\right)\left(\frac{3}{8}\right) = 0.3889$
- We can also double check that at depth 1, the 4 red triangles will be classified wrongly for node B and the 1 red triangle + 2 blue circles will be classified wrongly for node C. So in total, there will be 7 wrong classifications out of 18 datapoints, which corresponds to $\left(\frac{7}{18}\right) = 0.3889$
- Observe the decrease in misclassification rate from root (0.5556) to depth 1 (0.3889)

Q2

(MSE of regression trees)

Calculate the overall MSE for the following data at depth 1 of a regression tree assuming a decision threshold is taken at $x = 5.0$. How does it compare with the MSE at the root?

$\{x,y\}$: {1, 2}, {0.8, 3}, {2, 2.5}, {2.5, 1}, {3, 2.3}, {4, 2.8}, {4.2, 1.5}, {6, 2.6}, {6.3, 3.5}, {7, 4}, {8, 3.5}, {8.2, 5}, {9, 4.5}

Q2

At depth 1, when $x > 5$

- $y = \{2.6, 3.5, 4, 3.5, 5, 4.5\} => \bar{y} = 3.85$
- MSE $= \frac{1}{6}\left((2.6 - \bar{y})^2 + (3.5 - \bar{y})^2 + (4 - \bar{y})^2 + (3.5 - \bar{y})^2 + (5 - \bar{y})^2 + (4.5 - \bar{y})^2\right) = 0.5958$

At depth 1, when $x \leq 5$

- $y = \{2,\ 3,\ 2.5, 1, 2.3,\ 2.8, 1.5\} => \bar{y} = 2.1571$
- MSE $= \frac{1}{7}\left((2 - \bar{y})^2 + (3 - \bar{y})^2 + (2.5 - \bar{y})^2 + (1 - \bar{y})^2 + (2.3 - \bar{y})^2 + (2.8 - \bar{y})^2 + (1.5 - \bar{y})^2\right) = 0.4367$

Overall MSE at depth 1: $\frac{6}{13} \times 0.5958 + \frac{7}{13} \times 0.4367 = 0.5102$

Q2

At the root:

- $y = \{2,\ 3,\ 2.5, 1, 2.3,\ 2.8, 1.5, 2.6, 3.5, 4, 3.5, 5, 4.5\} => \bar{y} = 2.9385$
- MSE $= \frac{1}{13}((2.6 - \bar{y})^2 + (3.5 - \bar{y})^2 + (4 - \bar{y})^2 + (3.5 - \bar{y})^2 + (5 - \bar{y})^2 + (4.5 - \bar{y})^2 + (2 - \bar{y})^2 + (3 - \bar{y})^2 + (2.5 - \bar{y})^2 + (1 - \bar{y})^2 + (2.3 - \bar{y})^2 + (2.8 - \bar{y})^2 + (1.5 - \bar{y})^2) = 1.2224$

Therefore, MSE has decreased from 1.2224 at the root to 0.5102 at depth 1

(Regression tree, Python)

Q3

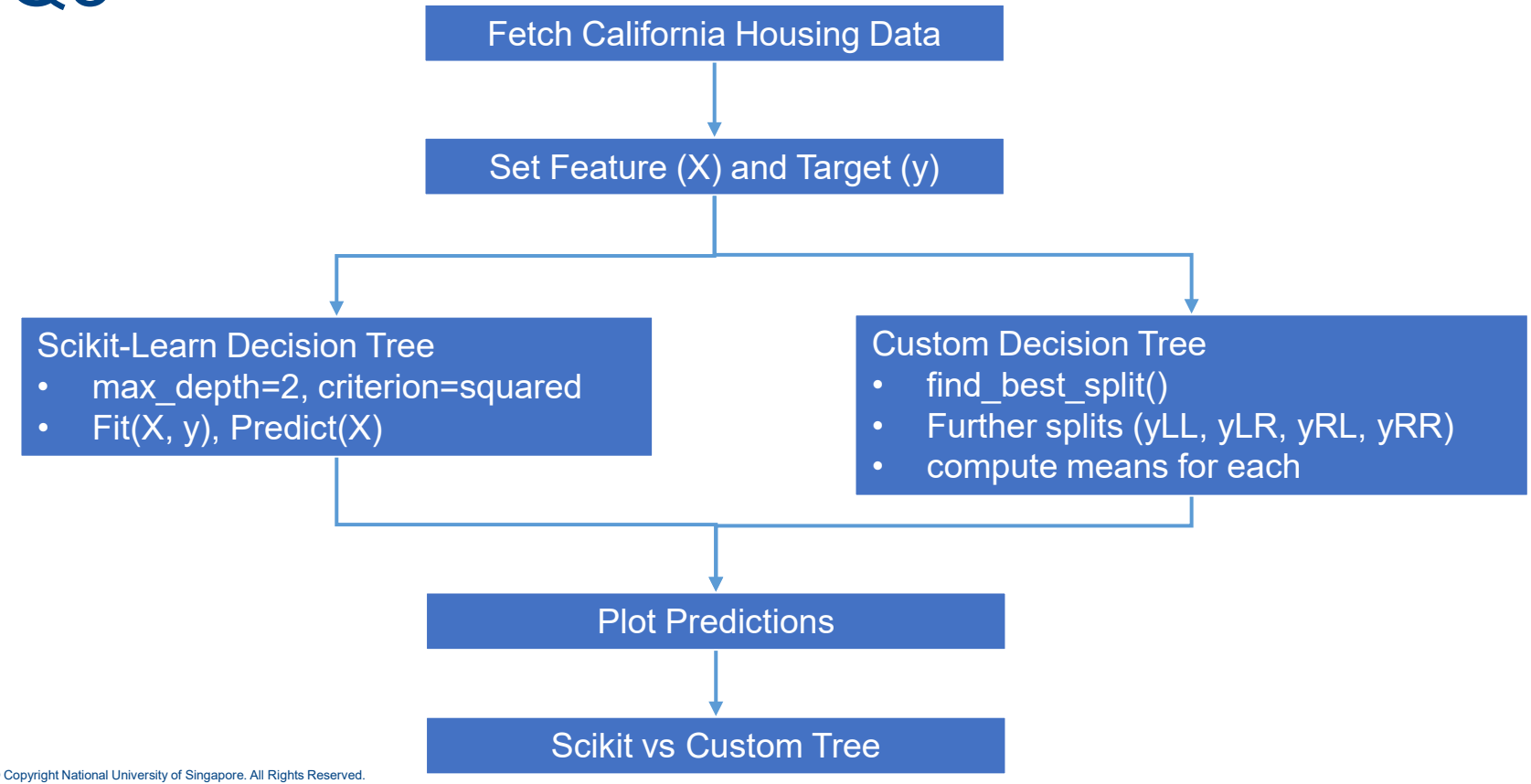
Import the California Housing dataset "`from sklearn.datasets import fetch_california_housing`" and "`housing = fetch_california_housing()`". This data set contains 8 features and 1 target variable listed below. Use "`MedInc`" as the input feature and "`MedHouseVal`" as the target output. Fit a regression tree to depth 2 and compare your results with results generated by "`from sklearn.tree import DecisionTreeRegressor`" using the "squared error" criterion.

```
Target: ['MedHouseVal']

Features:['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms',
'Population', 'AveOccup', 'Latitude', 'Longitude']
```

Q3

Block Diagram of the Code



Q3

```
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt
```

- numpy: For numerical computations.
- fetch_california_housing: To load the California housing dataset.
- DecisionTreeRegressor: Scikit-learn's decision tree implementation.
- matplotlib.pyplot: For plotting graphs.

Q3

```
def our_own_tree(y):

    # split data at first level
    # L stands for left, R stands for right
    yL, yR = find_best_split(y)

    # split data at second level
    yLL, yLR = find_best_split(yL)
    yRL, yRR = find_best_split(yR)

    # compute prediction
    yLL_pred = np.mean(yLL)*np.ones(len(yLL))
    yLR_pred = np.mean(yLR)*np.ones(len(yLR))
    yRL_pred = np.mean(yRL)*np.ones(len(yRL))
    yRR_pred = np.mean(yRR)*np.ones(len(yRR))
    y_pred = np.concatenate([yLL_pred, yLR_pred, yRL_pred, yRR_pred])

    return y_pred
```

our_own_tree(y):

This function implements a two-level decision tree. It splits the target data y into two groups, computes the prediction for each group (by taking their means), and returns the combined prediction for all points.

1) Split the data into two parts using the find_best_split() function.
2) Further split both parts (left and right) to form four groups in total.
3) For each of the four groups, compute the mean of the data in that group as the prediction and concatenate all predictions into one array.

Q3

```
def find_best_split(y):

    # index represents last element in the below threshold node
    sq_err_vec = np.zeros(len(y)-1)
    for index in range(0, len(y)-1):

        # split the data
        data_below_threshold = y[:index+1]
        data_above_threshold = y[index+1:]

        # Compute estimate
 ŷ1     mean_below_threshold = np.mean(data_below_threshold)
 ŷ2     mean_above_threshold = np.mean(data_above_threshold)

        # Compute total square error
        # Note that MSE = total square error divided by number of data points
 S1     below_sq_err = np.sum(np.square(data_below_threshold - mean_below_threshold))
 S2     above_sq_err = np.sum(np.square(data_above_threshold - mean_above_threshold))
        sq_err_vec[index] = below_sq_err + above_sq_err

    best_index = np.argmin(sq_err_vec)
    yL = y[:best_index+1]
    yR = y[best_index+1:]
    return yL, yR
```

find_best_split(y):

Finds the best point to split the data y to minimize the total squared error. It calculates the squared error for every possible split and selects the one with the smallest error.

1) Iterate through every possible index in y and compute the squared error for splitting the data at that index.
2) The index that minimizes the squared error is chosen as the best split point.
3) Return the two subsets of the data, one below and one above the threshold (split point).

$$S_m = \frac{1}{J_m}\sum_{j=1}^{J_m}(y_j - \hat{y}_m)^2$$

$$\text{MSE } S = \sum_m \frac{J_m}{N} S_m$$

J_m has been eliminated

The index that minimizes the squared error is chosen as the best split point.

Q3

```
housing = fetch_california_housing()
print(housing.target_names)
print(housing.feature_names)
X = housing.data[:,0]
y = housing.target
print(y)
sort_index = X.argsort()
X = X[sort_index]
y = y[sort_index]
```

Dataset Loading and Preparation:

1) The code fetches the California housing dataset, which contains median house values (y) and features such as the median income (X).
2) The feature X is sorted to ensure that predictions are consistent with the data order. Both X (median income) and y are sorted for smoother visualization in the plots.

Q3

```
print(X.reshape(-1,1))
# scikit decision tree regressor
scikit_tree = DecisionTreeRegressor(criterion='squared_error', max_depth=2)
scikit_tree.fit(X.reshape(-1,1), y) # reshape necessary because tree expects 2D array
scikit_tree_predict = scikit_tree.predict(X.reshape(-1,1))

# Our own tree regressor
tree_predict = our_own_tree(y)
```

Scikit-learn Tree:

- A decision tree regressor is created using DecisionTreeRegressor with a maximum depth of 2 and squared error as the criterion.
- The decision tree is fitted to the data, and predictions are made.

Custom Tree:

- The custom decision tree regressor (our_own_tree(y)) is used to predict the target values based on the predefined splitting rules.

Q3

```
# Plot
plt.figure(0, figsize=[9,4.5])
plt.rcParams.update({'font.size': 16})
plt.scatter(X, y, c='steelblue', s=20)
plt.plot(X, scikit_tree_predict, color='black', lw=2, label='scikit-learn')
plt.plot(X, tree_predict, color='red', linestyle='--', lw=2, label='our own tree')
plt.xlabel('MedInc')
plt.ylabel('MedHouseVal')
plt.legend(loc='upper right',ncol=3, fontsize=15)
plt.savefig('FigTut9_Q3.png')
```

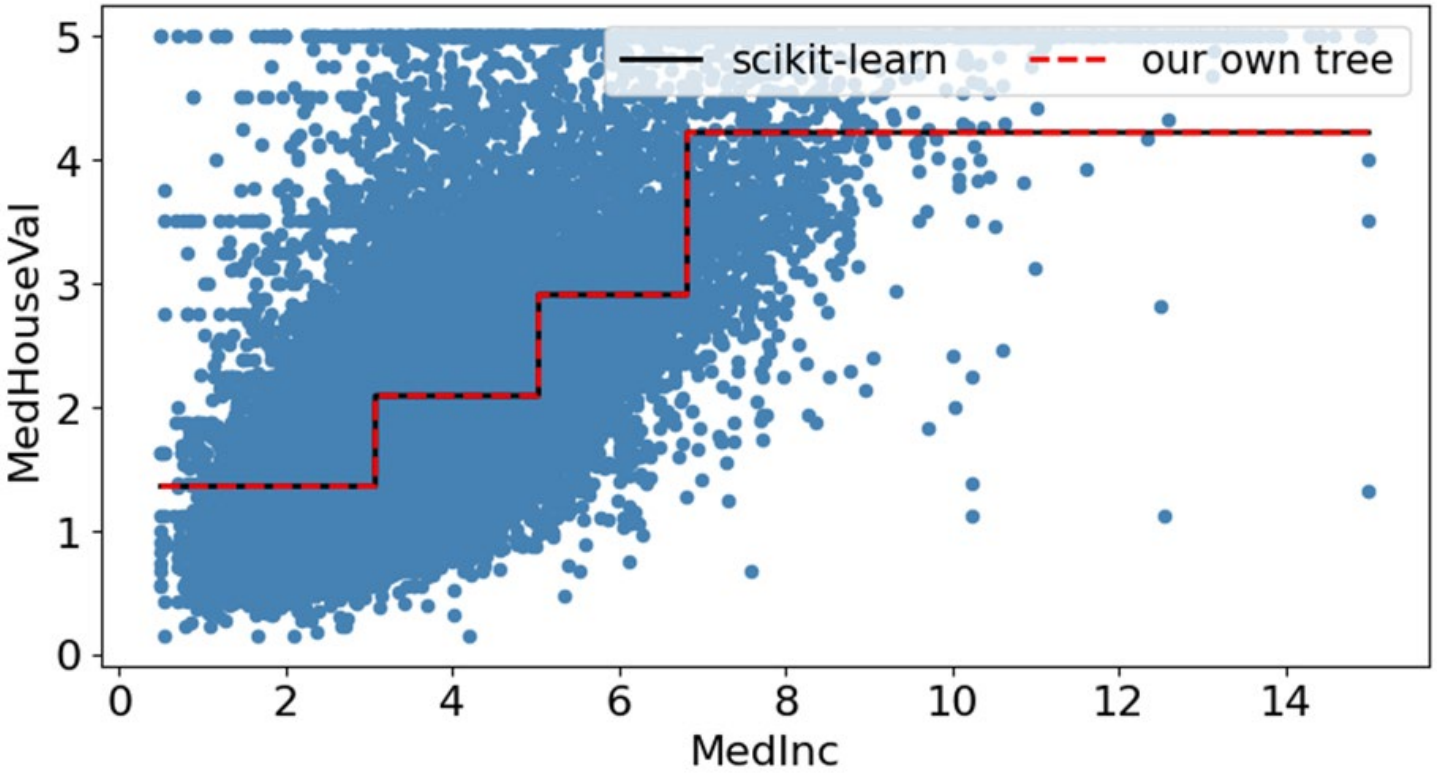
Plotting:

- The scatter plot shows the data, while two curves represent the predictions made by scikit-learn's tree and the custom tree.

Q3

Please refer to Tut9_Q3_zhou.py. We can exactly replicate the results from scikit-learn. Note that in the plot below, the blue dots are the training datapoints. The curves from scikit-learn (black line) and our own tree (red dashed line) are on top of each other, so they might be hard to tell apart.

Q3


scikit-learn
our own tree
MedHouseVal
MedInc
0
1
2
3
4
5
0
2
4
6
8
10
12
14

(Classification tree, Python)

Q4

Get the data set “from sklearn.datasets import load_iris”. Perform the following tasks.

(a) Split the database into two sets: 80% of samples for training, and 20% of samples for testing using random_state=0

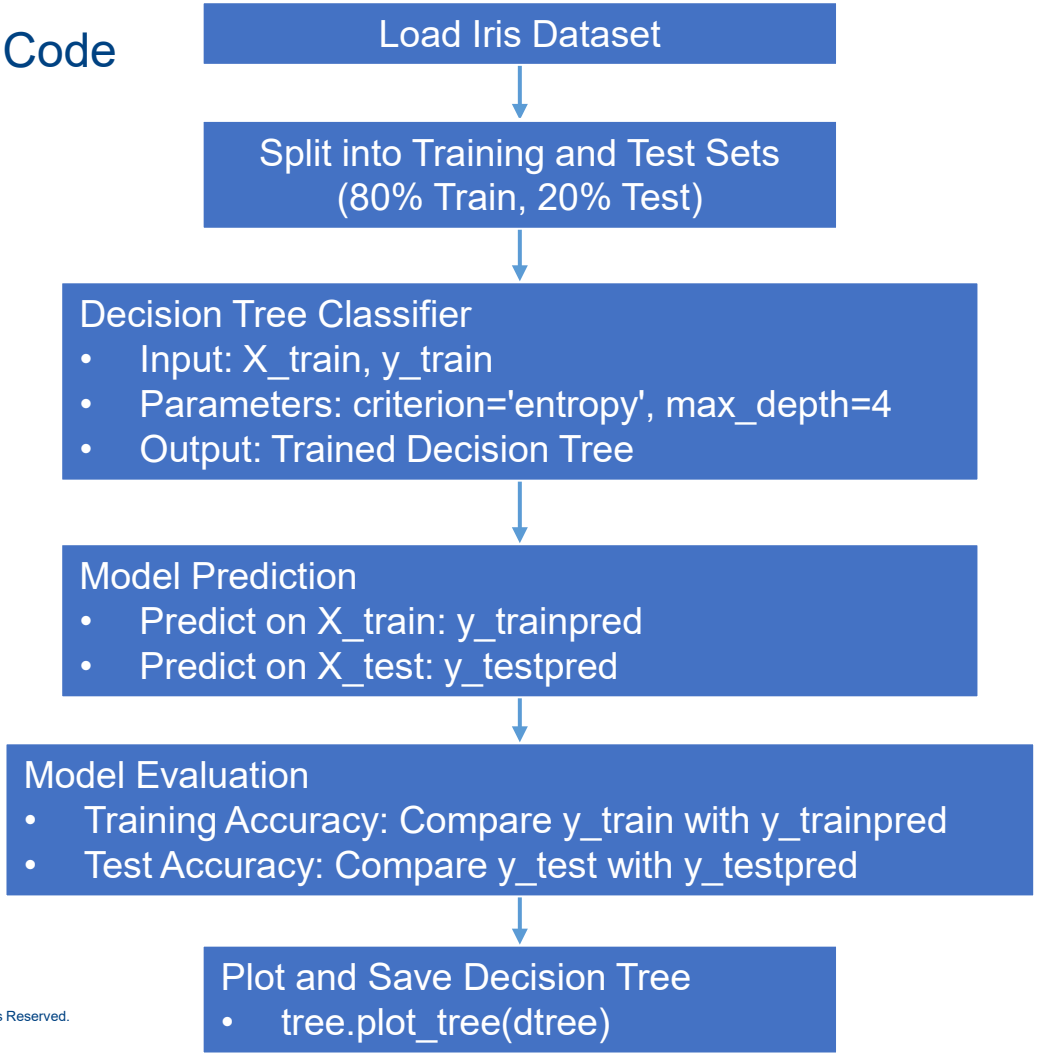
(b) Train a decision tree classifier (i.e., “tree.DecisionTreeClassifier” from sklearn) using the training set with a maximum depth of 4 based on the “entropy” criterion.

(c) Compute the training and test accuracies. You can use accuracy_score from sklearn.metrics for accuracy computation

(d) Plot the tree using “tree.plot_tree”.

Block Diagram of the Code

Q4



Q4

```
from sklearn.datasets import load_iris
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn import metrics
import matplotlib.pyplot as plt
```

Libraries Imported:

- load_iris: Loads the well-known Iris dataset.
- tree: Contains the decision tree classifier and plotting utilities.
- train_test_split: Splits the dataset into training and test sets.
- metrics: Contains functions to evaluate the model (like accuracy score).
- matplotlib.pyplot: For plotting graphs, specifically the decision tree.

Q4

```
# load data
iris_dataset = load_iris()

# split dataset
X_train, X_test, y_train, y_test = train_test_split(iris_dataset['data'],
                                                    iris_dataset['target'],
                                                    test_size=0.20,
                                                    random_state=0)
```

Load the Iris Dataset:

- The **Iris dataset** is a small dataset with 150 instances of iris flowers, each described by four features (sepal length, sepal width, petal length, and petal width) and categorized into one of three classes (setosa, versicolor, and virginica).

Split the Dataset:

- The dataset is split into training and test sets using an 80%-20% ratio. The train_test_split function creates X_train and y_train for the training data and X_test and y_test for the testing data.
- The random_state=0 ensures that the split is reproducible.

Q4

```
# fit tree
dtree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=4)
dtree = dtree.fit(X_train, y_train)

# predict
y_trainpred = dtree.predict(X_train)
y_testpred = dtree.predict(X_test)
```

Train the Decision Tree Classifier:

- A decision tree classifier (DecisionTreeClassifier) is initialized with two key parameters:
  - **criterion='entropy'**: The tree is trained using information gain as the splitting criterion (based on entropy).
  - **max_depth=4**: The tree is restricted to a maximum depth of 4 to prevent overfitting.
- The tree is trained using the training data (X_train and y_train).

Make Predictions:

- The trained decision tree is used to make predictions on both the training data (y_trainpred) and the test data (y_testpred).

Q4

```
# print accuracies
print("Training accuracy: ", metrics.accuracy_score(y_train, y_trainpred))
print("Test accuracy: ", metrics.accuracy_score(y_test, y_testpred))

# Plot tree
tree.plot_tree(dtree)
plt.savefig('FigTut9_Q4.eps')
plt.show()
```

Evaluate the Model:

- The accuracy of the model is calculated for both the training and test sets using metrics.accuracy_score(). This function computes the percentage of correct predictions by comparing the predicted values (y_trainpred and y_testpred) with the true values (y_train and y_test).

Plot the Decision Tree:

- The tree.plot_tree() function visualizes the decision tree. The plot is saved as an EPS file ('FigTut9_Q4.eps').

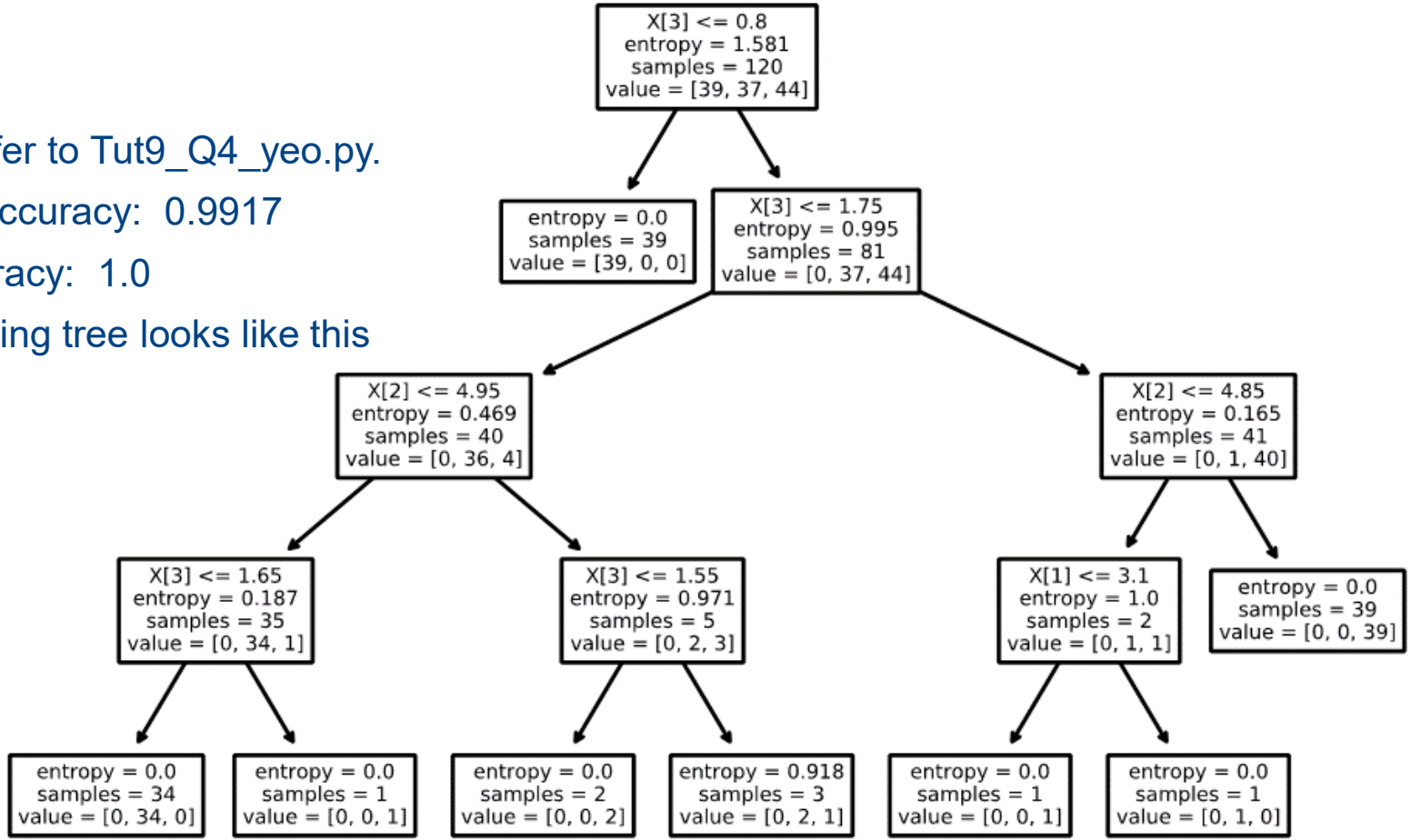
Q4

Please refer to Tut9_Q4_yeo.py.

Training accuracy: 0.9917

Test accuracy: 1.0

The resulting tree looks like this



THANK YOU