uOttawa

Faculté de génie
Faculty of Engineering

Applied Machine Learning - Summer 2021

Assignment 4 - Decision Tree and Ensemble Learning

Submitted by:

Name	Email
Youssef Metwally	ymetw027@uOttawa.ca
Dina Ibrahim Morsy	dabde007@uOttawa.ca
Adel El Nabarawy	aelna025@uOttawa.ca

Submitted to:

Dr. Murat SIMSEK, SMIEEE

Numerical Questions

Q1 (decision tree by using Gini Index):

Gini index for ‘Weather’:

Weather	Yes	No	Number of instances
Cloudy	2	1	3
Sunny	2	1	3
Rainy	1	3	4

Gini (Weather = Cloudy) = $1 – (2/3)^2 – (1/3)^2 = 0.444$

Gini (Weather = Sunny) = $1 – (2/3)^2 – (1/3)^2 = 0.444$

Gini (Weather = Rainy) = $1 – (1/4)^2 – (3/4)^2 = 0.375$

Then, we will calculate weighted sum of Gini indexes for ‘Weather’ feature.

Gini (Weather) = (3/10) x 0.444 + (3/10) x 0.444 + (4/10) x 0.375 = 0.4164

Gini index for ‘Temperature’:

Temperature	Yes	No	Number of instances
Hot	2	2	4
Mild	3	2	5
Cool	1	0	1

Gini (Temperature = Hot) = $1 – (2/4)^2 – (2/4)^2 = 0.5$

Gini (Temperature = Mild) = $1 – (3/5)^2 – (2/5)^2 = 0.48$

Gini (Temperature = Cool) = $1 – (1/1)^2 – (0/1)^2 = 0$

Then, we will calculate weighted sum of Gini indexes for ‘Temperature’ feature.

Gini (Temperature) = (4/10) x 0.5 + (5/10) x 0.48 + (1/10) x 0 = 0.44

Gini index for ‘Humidity’:

Humidity	Yes	No	Number of instances
High	3	4	7
Normal	2	1	3

Gini (Humidity = High) = $1 – (3/7)^2 – (4/7)^2 = 0.489$

Gini (Humidity = Normal) = $1 – (2/3)^2 – (1/3)^2 = 0.444$

Then, we will calculate weighted sum of Gini indexes for ‘Humidity’ feature.

Gini (Humidity) = (7/10) x 0.489+ (3/10) x 0.444 = 0.4755

Gini index for ‘Wind’:

Wind	Yes	No	Number of instances
Weak	3	1	4
Strong	2	4	6

Gini (Wind = Cloudy) = $1 – (3/4)^2 – (1/4)^2 = 0.375$

Gini (Wind = Sunny) = $1 – (2/6)^2 – (4/6)^2 = 0.444$

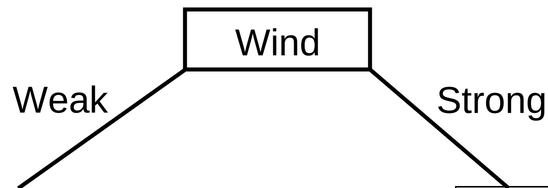
Then, we will calculate weighted sum of Gini indexes for ‘Wind’ feature.

Gini (Wind) = (4/10) x 0.375 + (6/10) x 0.444 = 0.4164

Splitting Decision:

We’ve calculated Gini index values for each feature. We will choose Weather feature or Wind feature randomly since they have the same BEST (lowest) Gini Index. Wind feature will be chosen.

Feature	Gini Index
Weather	0.4164
Temperature	0.44
Humidity	0.4755
Wind	0.4164



Weather	Temperature	Humidity	Wind	Hiking
Cloudy	Hot	High	Weak	No
Sunny	Hot	High	Weak	Yes
Rainy	Mild	High	Weak	Yes
Cloudy	Hot	Normal	Weak	Yes

Weather	Temperature	Humidity	Wind	Hiking
Sunny	Mild	Normal	Strong	Yes
Cloudy	Mild	High	Strong	Yes
Rainy	Mild	High	Strong	No
Rainy	Cool	Normal	Strong	No
Sunny	Hot	High	Strong	No
Rainy	Mild	High	Strong	No

Gini index of ‘Weather’ for 'Weak Wind':

Weather	Yes	No	Number of instances
Cloudy	1	1	2
Sunny	1	0	1
Rainy	1	0	1

Gini (Weather = Cloudy & Wind = Weak) = $1 - (1/2)^2 - (1/2)^2 = 0.5$

Gini (Weather = Sunny & Wind = Weak) = $1 - (1/1)^2 - (0/1)^2 = 0$

Gini (Weather = Rainy & Wind = Weak) = $1 - (1/1)^2 - (0/1)^2 = 0$

Then, we will calculate weighted sum of Gini indexes for ‘Weather’ feature.

Gini (Weather) = $(2/4) \times 0.5 + (1/4) \times 0 + (1/4) \times 0 = 0.25$

Gini index of ‘Temperature’ for 'Weak Wind':

Temperature	Yes	No	Number of instances
Hot	2	1	3
Mild	1	0	1
Cool	0	0	0

Gini (Temperature = Hot & Wind = Weak) = $1 – (2/3)^2 – (1/3)^2 = 0.5$

Gini (Temperature = Mild & Wind = Weak) = $1 – (1/1)^2 – (0/1)^2 = 0$

Gini (Temperature = Cool & Wind = Weak) = $1 – (0)^2 – (0)^2 = 1$

Then, we will calculate weighted sum of Gini indexes for ‘Temperature’ feature.

Gini (Temperature) = (3/4) x 0.5 + (1/4) x 0 + (0/4) x 1 = 0.375

Gini index of ‘Humidity’ for 'Weak Wind':

Humidity	Yes	No	Number of instances
High	2	1	3
Normal	1	0	1

Gini (Humidity = High & Wind = Weak) = $1 – (2/3)^2 – (1/3)^2 = 0.444$

Gini (Humidity = Normal & Wind = Weak) = $1 – (1/1)^2 – (0/1)^2 = 0$

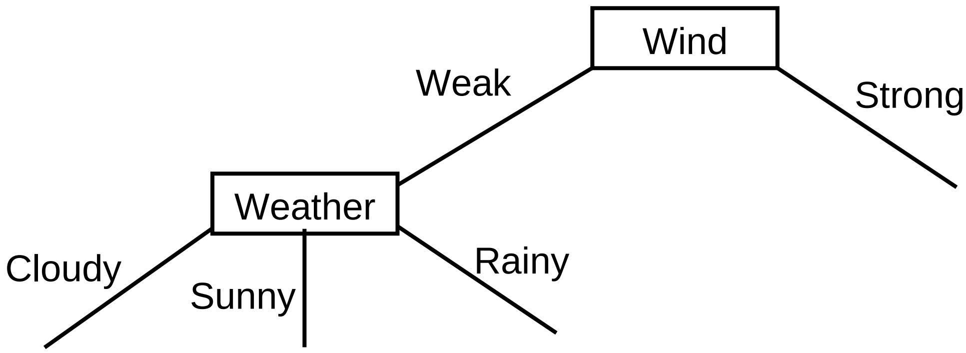
Then, we will calculate weighted sum of Gini indexes for ‘Humidity’ feature.

Gini (Humidity) = (3/4) x 0.444+ (1/4) x 0= 0.333

Splitting Decision:

The winner will be Weather feature because its cost is the lowest.

Feature	Gini Index
Weather	0.25
Temperature	0.375
Humidity	0.333



Gini index for ‘Weather’ for 'Strong Wind’:

Weather	Yes	No	Number of instances
Cloudy	1	0	1
Sunny	1	1	2
Rainy	3	0	3

Gini (Weather = Cloudy) = $1 – (1/1)^2 – (0/1)^2 = 0$

Gini (Weather = Sunny) = $1 – (1/2)^2 – (1/2)^2 = 0.5$

Gini (Weather = Rainy) = $1 – (3/3)^2 – (0/3)^2 = 0$

Then, we will calculate weighted sum of Gini indexes for ‘Weather’ feature.

Gini (Weather) = (1/6) x 0 + (2/6) x 0.5 + (3/6) x 0 = 0.167

Gini index for ‘Temperature’ for 'Strong Wind’:

Temperature	Yes	No	Number of instances
Hot	0	1	1
Mild	2	2	4
Cool	0	1	1

Gini (Temperature = Hot) = $1 – (0/1)^2 – (1/1)^2 = 0$

Gini (Temperature = Mild) = $1 – (2/4)^2 – (2/4)^2 = 0.5$

Gini (Temperature = Cool) = $1 – (0/1)^2 – (1/1)^2 = 0$

Then, we will calculate weighted sum of Gini indexes for ‘Temperature’ feature.

Gini (Temperature) = (1/6) x 0 + (4/6) x 0.5 + (1/6) x 0 = 0.333

Gini index for ‘Humidity’ for 'Strong Wind’:

Humidity	Yes	No	Number of instances
High	1	3	4
Normal	1	1	2

Gini (Humidity = High) = $1 – (1/4)^2 – (3/4)^2 = 0.375$

Gini (Humidity = Normal) = $1 – (1/2)^2 – (1/2)^2 = 0.5$

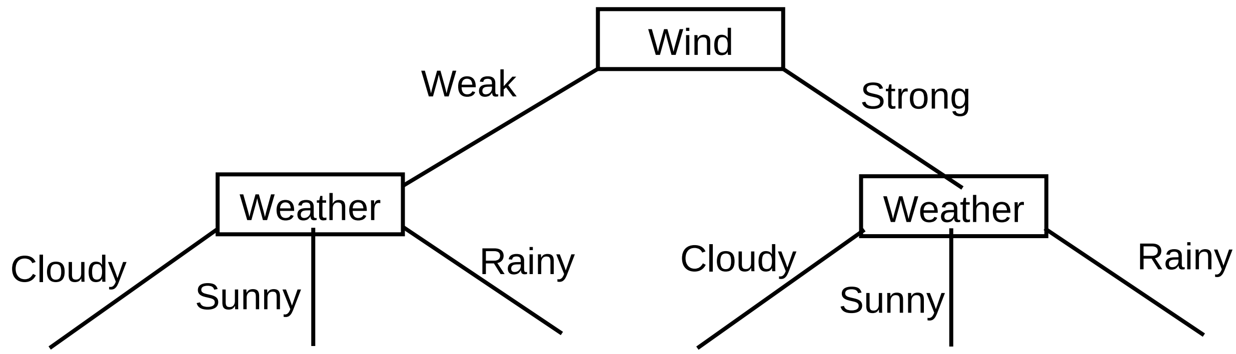
Then, we will calculate weighted sum of Gini indexes for ‘Humidity’ feature.

Gini (Humidity) = (4/6) x 0.375 + (2/6) x 0.5 = 0.417

Splitting Decision:

The winner will be Weather feature because its cost is the lowest.

Feature	Gini Index
Weather	0.167
Temperature	0.333
Humidity	0.417



Gini index of ‘Temperature’ for 'Weak Wind & Cloudy Weather':

Temperature	Yes	No	Number of instances
Hot	1	1	2

Gini (Temperature = Hot & Wind = Weak & Weather = Cloudy) = $1 – (1/2)^2 – (1/2)^2 = 0.5$
Then, we will calculate Gini index for ‘Temperature’ feature.
Gini (Temperature) = (2/2) x 0.5 = 0.5

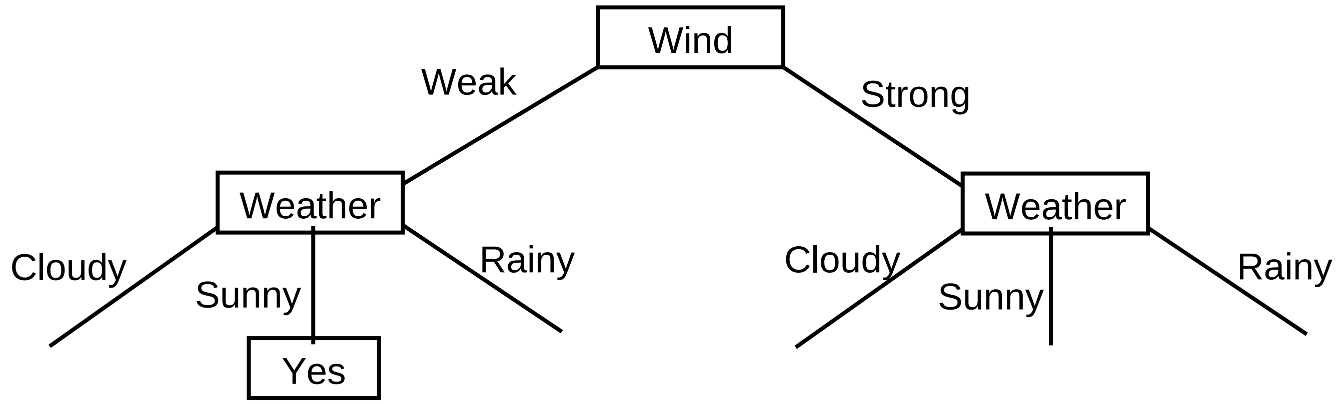
Gini index of ‘Temperature’ for 'Weak Wind & Sunny Weather':

Temperature	Yes	No	Number of instances
Hot	1	0	1

Gini (Temperature = Hot & Wind = Weak & Weather = Sunny) = $1 – (1/1)^2 – (0/1)^2 = 0$

Then, we will calculate Gini index for ‘Temperature’ feature.
Gini (Temperature) = (1/1) x 0 = 0

This result is because that sub dataset in the Sunny leaf has only 1 record with yes decisions. This means that Sunny leaf is over.



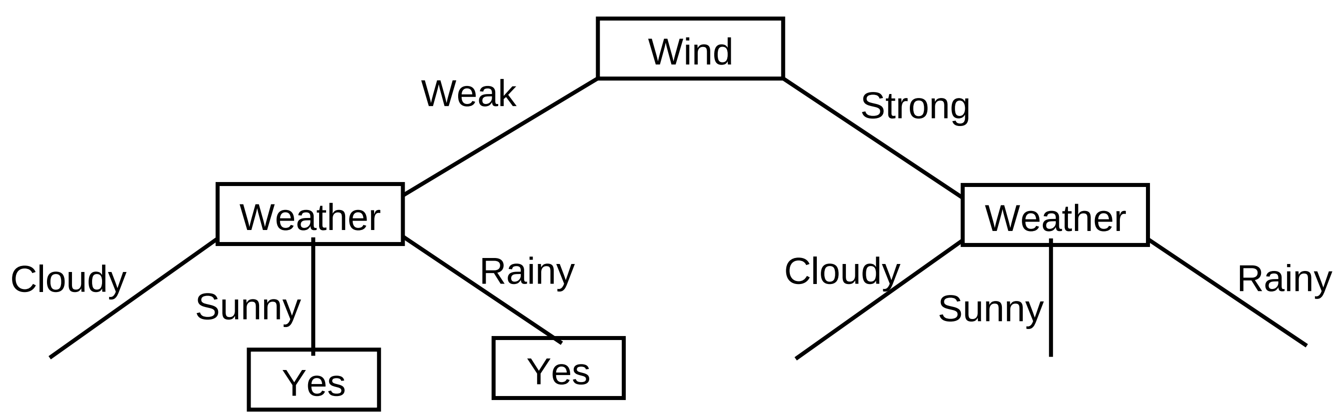
<u>Gini index of ‘Temperature’ for 'Weak Wind & Rainy Weather':</u>

Temperature	Yes	No	Number of instances
Mild	1	0	1

Gini (Temperature = Mild & Wind = Weak & Weather = Rainy) = $1 – (1/1)^2 – (0/1)^2 = 0$

Then, we will calculate Gini index for ‘Temperature’ feature.
Gini (Temperature) = (1/1) x 0 = 0

This result is because that sub dataset in the Rainy leaf has only 1 record with yes decisions. This means that Rainy leaf is over.



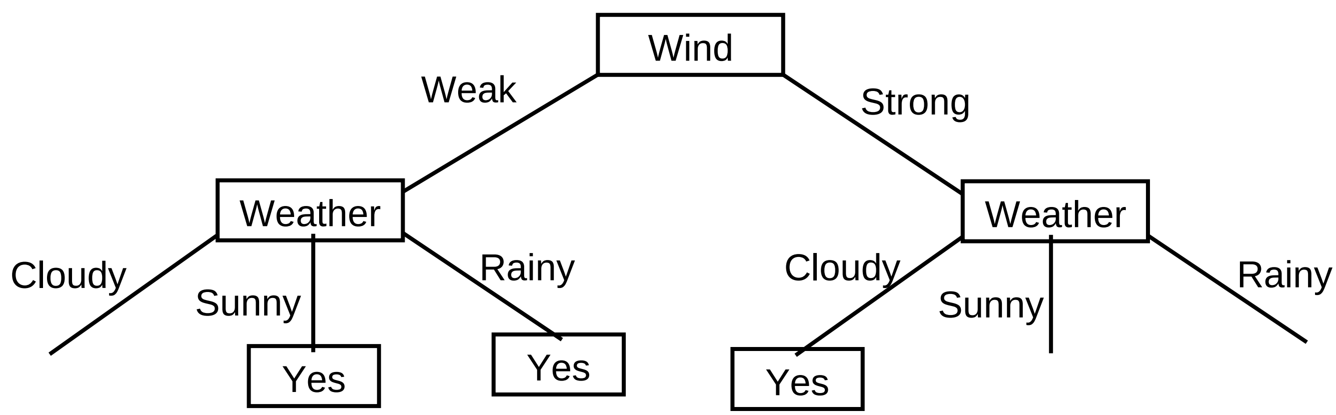
Gini index for 'Strong Wind & Cloudy Weather':

Weather	Yes	No	Number of instances
Cloudy	1	0	1

Gini (Weather = Cloudy & Wind = Strong) = 1 – $(1/1)^2$ – $(0/1)^2$ = 0

Then, we will calculate Gini index for ‘Weather’ feature.
Gini (Weather) = (1/6) x 0 = 0

This result is because that sub dataset in the Cloudy leaf has only 1 record with yes decisions. This means that Cloudy leaf is over.



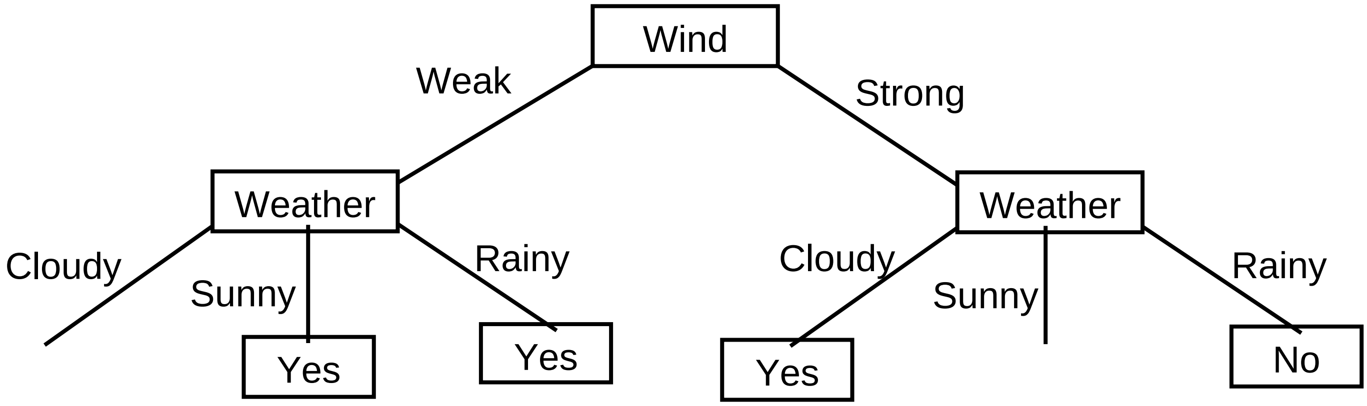
Gini index for 'Strong Wind & Rainy Weather':

Weather	Yes	No	Number of instances
Rainy	0	3	3

Gini (Weather = Rainy & Wind = Strong) = 1 – $(0/3)^2$ – $(3/3)^2$ = 0

Then, we will calculate Gini index for ‘Temperature’ feature.
Gini (Weather) = (3/6) x 0 = 0

This result is because that sub dataset in the Rainy leaf has only 1 record with yes decisions. This means that Rainy leaf is over.



Gini index of ‘Temperature’ for 'Weak Wind & Cloudy Weather':

Temperature	Yes	No	Number of instances
Hot	1	1	2

Gini (Temperature = Hot & Wind = Weak & Weather = Cloudy) = 1 – $(1/2)^2$ – $(1/2)^2$ = 0.5

Then, we will calculate Gini index for ‘Temperature’ feature.
Gini (Temperature) = (2/2) x 0.5 = 0.5

Gini index of ‘Humidity’ for 'Weak Wind & Cloudy Weather':

Humidity	Yes	No	Number of instances
High	0	1	1
Normal	1	0	1

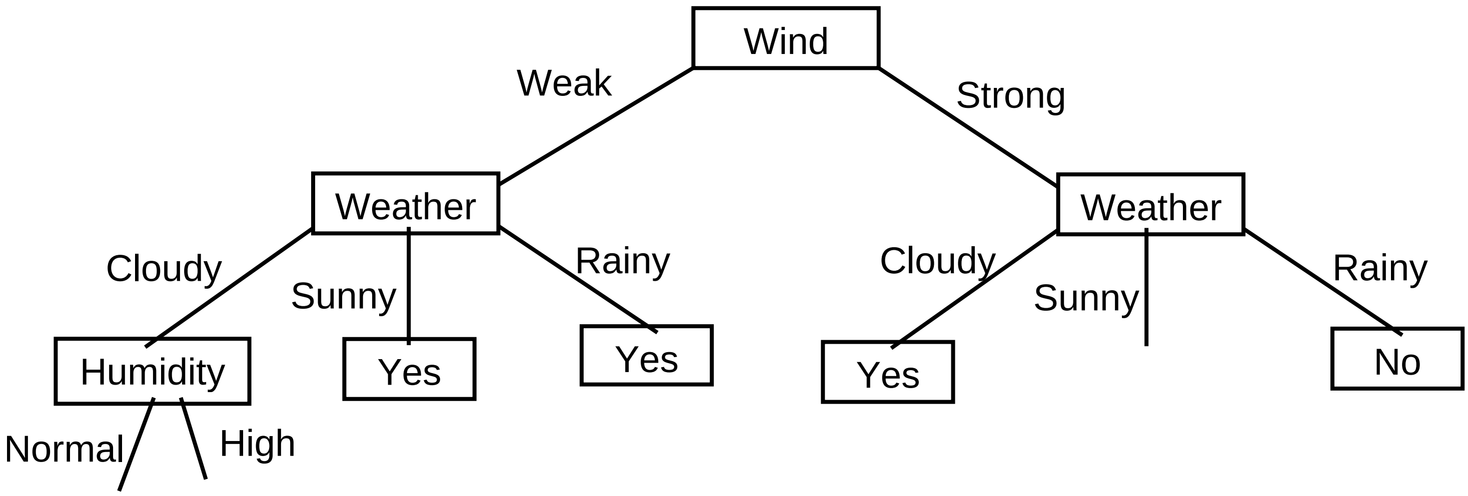
Gini (Humidity = High & Wind = Weak & Weather = Cloudy) = 1 – $(0/1)^2$ – $(1/1)^2$ = 0
Gini (Humidity = Normal & Wind = Weak & Weather = Cloudy) = 1 – $(1/0)^2$ – $(0/1)^2$ = 0

Then, we will calculate Gini index for ‘Temperature’ feature.
Gini (Temperature) = (1/2) x 0 + (1/2) x 0 = 0

Splitting Decision:

The winner will be Humidity feature because its cost is the lowest.

Feature	Gini Index
Temperature	0.5
Humidity	0



Gini index of 'Temperature' for 'Strong Wind & Sunny Weather':

Temperature	Yes	No	Number of instances
Mild	1	0	1
Hot	0	1	1

Gini (Temperature = Mild & Wind = Strong & Weather = Sunny) = $1 - (1/1)^2 - (0/1)^2 = 0$
Gini (Temperature = Hot & Wind = Strong & Weather = Sunny) = $1 - (0/1)^2 - (1/1)^2 = 0$

Then, we will calculate Gini index for 'Temperature' feature.
Gini (Temperature) = $(1/2) \times 0 + (1/2) \times 0 = 0$

Gini index of ‘Humidity’ for 'Weak Wind & Sunny Weather':

Humidity	Yes	No	Number of instances
High	0	1	1
Normal	1	0	1

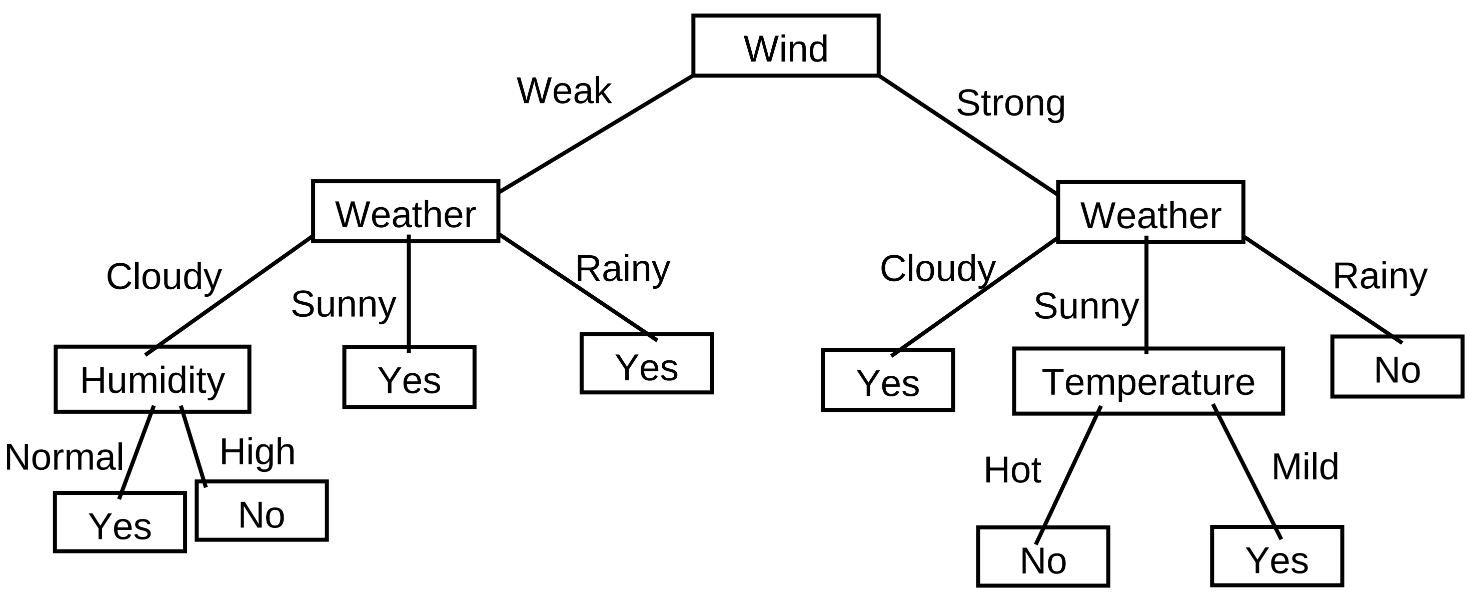
Gini (Humidity = High & Wind = Strong & Weather = Sunny) = $1 – (0/1)^2 – (1/1)^2 = 0$
Gini (Humidity = Normal & Wind = Strong & Weather = Sunny) = $1 – (1/0)^2 – (0/1)^2 = 0$

Then, we will calculate Gini index for ‘Temperature’ feature.
Gini (Humidity) = (1/2) x 0 + (1/2) x 0 = 0

Splitting Decision:

We will choose a random one of them since they have the same Gini Index. Temperature feature will be chosen.

Feature	Gini Index
Temperature	0
Humidity	0



Q2 (decision tree by using Information Gain):

info(s) = I[5,5] = - (5/10)*log(5/10) - (5/10)*log2(5/10) = 1

Splitting weather attribute

- weather = cloudy
  I[2,1] = - (2/3)*log2(2/3) - (1/3)*log2(1/3) = 0.918
- weather = sunny
  I[2,1] = - (2/3)*log2(2/3) - (1/3)*log2(1/3) = 0.918
- weather = rainy
  I[1,3] = - (1/4)*log2(1/4) - (3/4)*log2(3/4) = 0.811

Entropy(weathet) = (3/10)* 0.918 + (3/10)* 0.918 + (4/10)* 0.811 = 0.875

IG (weathet) = 1 - 0.875 = 0.125

Splitting temperature attribute

- temperature = hot
  I[2,2] = - (2/4)*log2(2/4) - (2/4)*log2(2/4) = 1
- temperature = mild
  I[3,2] = - (2/5)*log2(2/5) - (3/5)*log2(3/5) = 0.971
- temperature = cool
  I[1,0] = 0

Entropy(temperature) = (4/10)*1 + (5/10)* 0.971 + (1/10)*0 = 0.886

IG (temperature) = 1 - 0.886 = 0.114

Splitting Humidity attribute

- Humidity = normal
  I[2,1] = - (2/3)*log2(2/3) - (1/3)*log2(1/3) = 0.918
- Humidity = high
  I[3,4] = - (4/7)*log2(4/7) - (3/7)*log2(3/7) = 0.985

Entropy(Humidity) = (3/10)* 0.918 + (7/10)*0.985 = 0.965

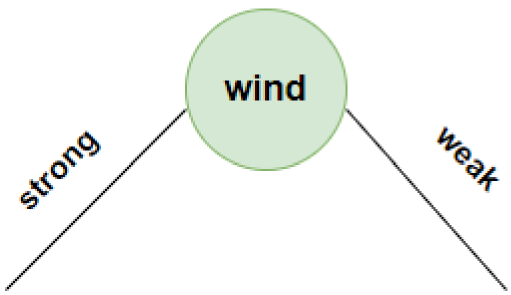
IG (Humidity) = 1 - 0.965 = 0.035

Splitting Wind attribute

- Wind = weak
  I[3,1] = - (1/4)*log2(1/4) - (3/4)*log2(3/4) = 0.811
- Wind = high
  I[2,4] = - (4/6)*log2(4/6) - (2/6)*log2(2/6) = 0.918

Entropy(Wind) = (4/10)* 0.811 + (6/10)*0.918 = 0.875

IG (Wind) = 1 - 0.875 = 0.125



the wind and weather have the heighest IG So, I will choose the weather

Info(strong) = 0.918

Wind = strong, weather

- Info(sunny) = i[1,1] = 1
- Info(cloudy) = i[1,0] = 0
- Info(rainy) = i[0,3] = 0

Entropy(Wind = strong, weather) = (2/6)*1 + 0 + 0 = 1/3

IG (Wind = strong, weather) = 0.918 - 0.33 = 0.585

Wind = strong, Temperature

- Info(hot) = i[0,1] = 0
- Info(mild) = i[2,2] = 1
- Info(cold) = i[0,1] = 0

Entropy(Wind = strong, **Temperature**) = (4/6)*1 + 0 + 0 = 0.667

IG (Wind = strong, **Temperature**) = 0.918 - 0.667 = 0.251

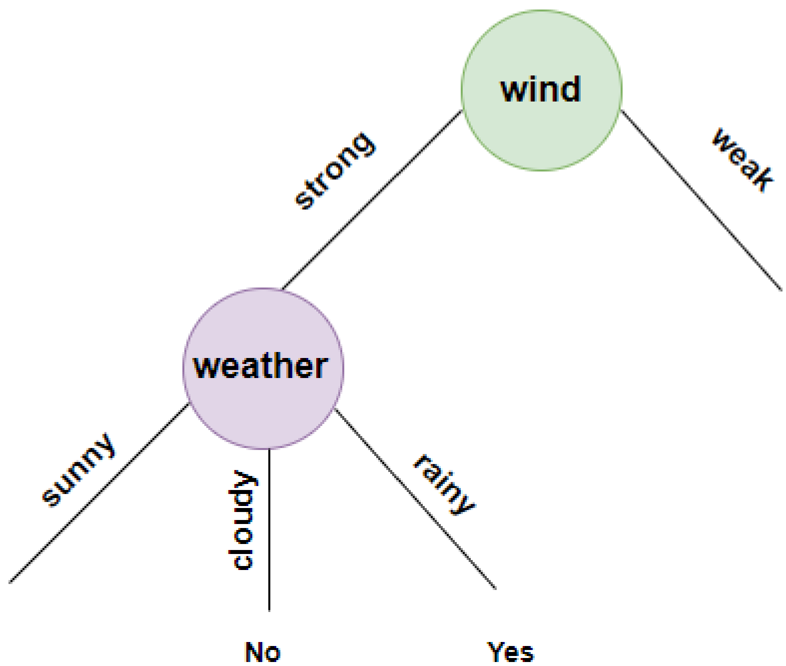
Wind = strong, humidity

- Info(high) = i[1,3] = - (1/4)*log2(1/4) - (3/4)*log2(3/4) = 0.811
- Info(normal) = i[1,1] = 1

Entropy(Wind = strong, **humidity**) = (4/6)* 0.811 + (2/6)*1 = 0.874

IG (Wind = strong, **humidity**) = 0.918 - 0.874 = 0.044

the weather have the heighest IG So, I will choose it



Info(sunny) = i[1,1] = 1

Wind = strong, weather = sunny, Temperature

- Info(hot) = i[0,1] = 0
- Info(mild) = i[1,0] = 0
- Info(cold) = i[0,0] = 0

Entropy(Wind = strong, weather = sunny, Temperature) = 0

IG (**Wind = strong, weather = sunny, Temperature**) = 1 - 0 = 1

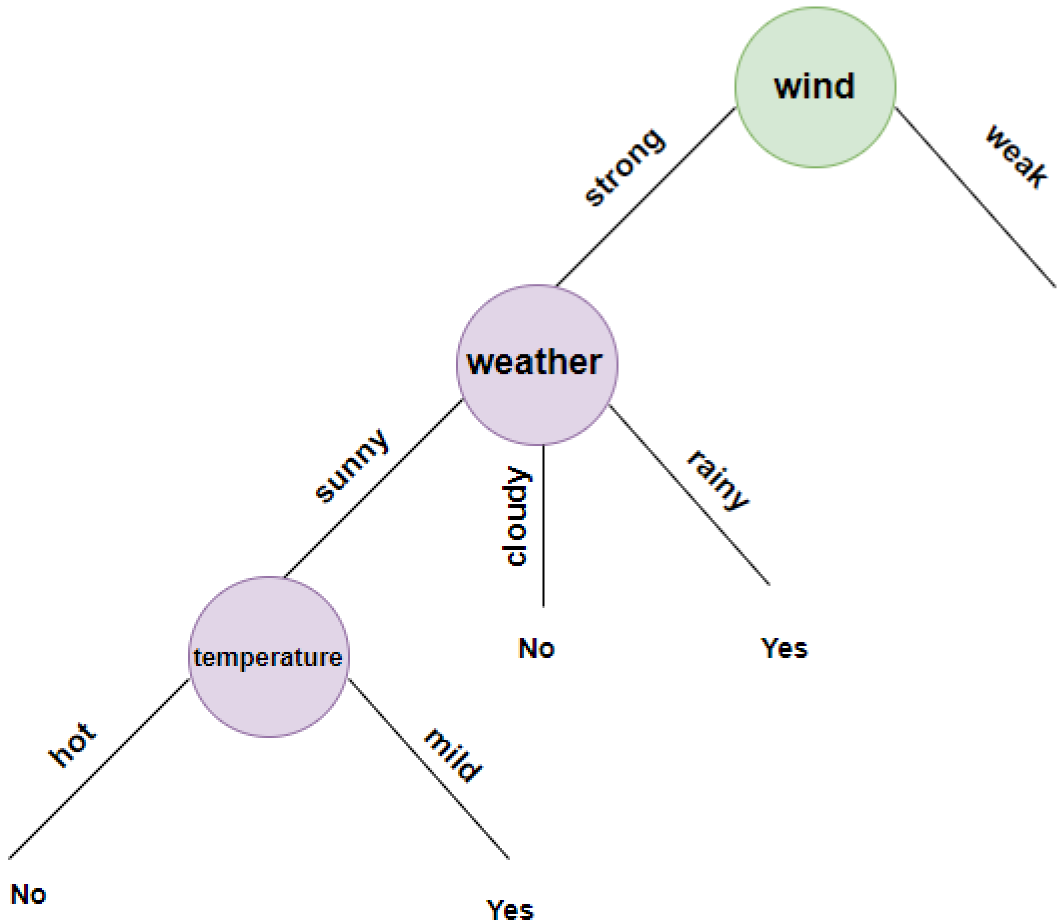
Wind = strong, weather = sunny, humidity

- Info(high) = i[0,1] = 0
- Info(normal) = i[1,0] = 0

Entropy(Wind = strong, weather = sunny, humidity) = 0

IG (**Wind = strong, weather = sunny, humidity**) = 1 - 0 = 1

the temperature and humidity have the heighest IG So, I will choose the temperature



Info(weak) = 0.811

Wind = weak**, weather**

- Info(sunny) = i[1,0] = 0
- Info(cloudy) = i[1,1] = 1
- Info(rainy) = i[1,0] = 0

Entropy(Wind = weak, weather) = (2/4)*1 + 0 + 0 = 1/2

IG (Wind = weak, weather) = 0.811 - 0.5 = 0.311

Wind = weak**, Temperature**

- Info(hot) = i[2,1] = - (2/3)*log2(2/3) - (1/3)*log2(1/3) = 0.918
- Info(mild) = i[1,0] = 0
- Info(cold) = i[0,0] = 0

Entropy(Wind = weak, weather) = (3/4)* 0.918 + 0 + 0 = 0.689

IG (Wind = weak, temperature) = 0.811 - 0.689 = 0.122

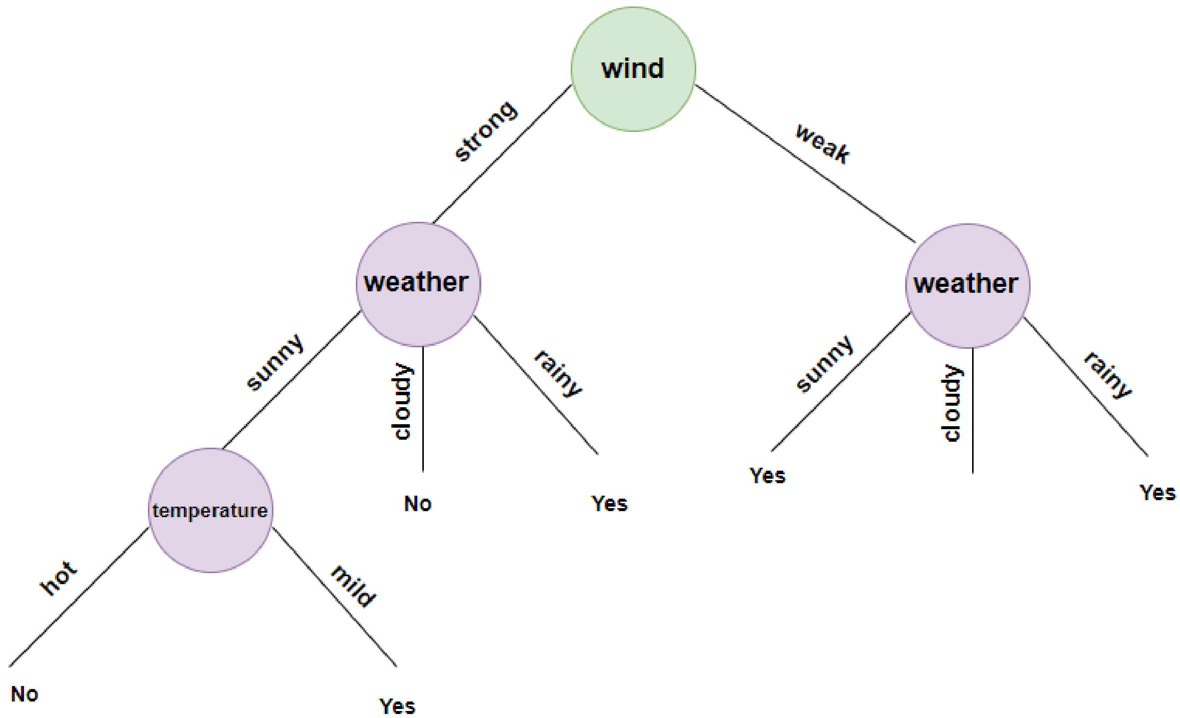
Wind = weak**, humidity**

- Info(high) = i[2,1] = - (2/3)*log2(2/3) - (1/3)*log2(1/3) = 0.918
- Info(normal) = i[1,0] = 0

Entropy(Wind = weak, weather) = (3/4)* 0.918 + 0 = 0.689

IG (Wind = weak, humidity) = 0.811 - 0.689 = 0.122

the weather have the heighest IG So, I will choose it



Info(cloudy) = i[1,1] = 1

Wind = weak**, weather =** cloudy**, Temperature**

- Info(hot) = i[1,1] = 1
- Info(mild) = i[0,0] = 0
- Info(cold) = i[0,0] = 0

Entropy(**Wind =** weak**, weather =** cloudy**, Temperature**) = 2/2 = 1

IG (**Wind =** weak**, weather =** cloudy**, Temperature**) = 1 - 1 = 0

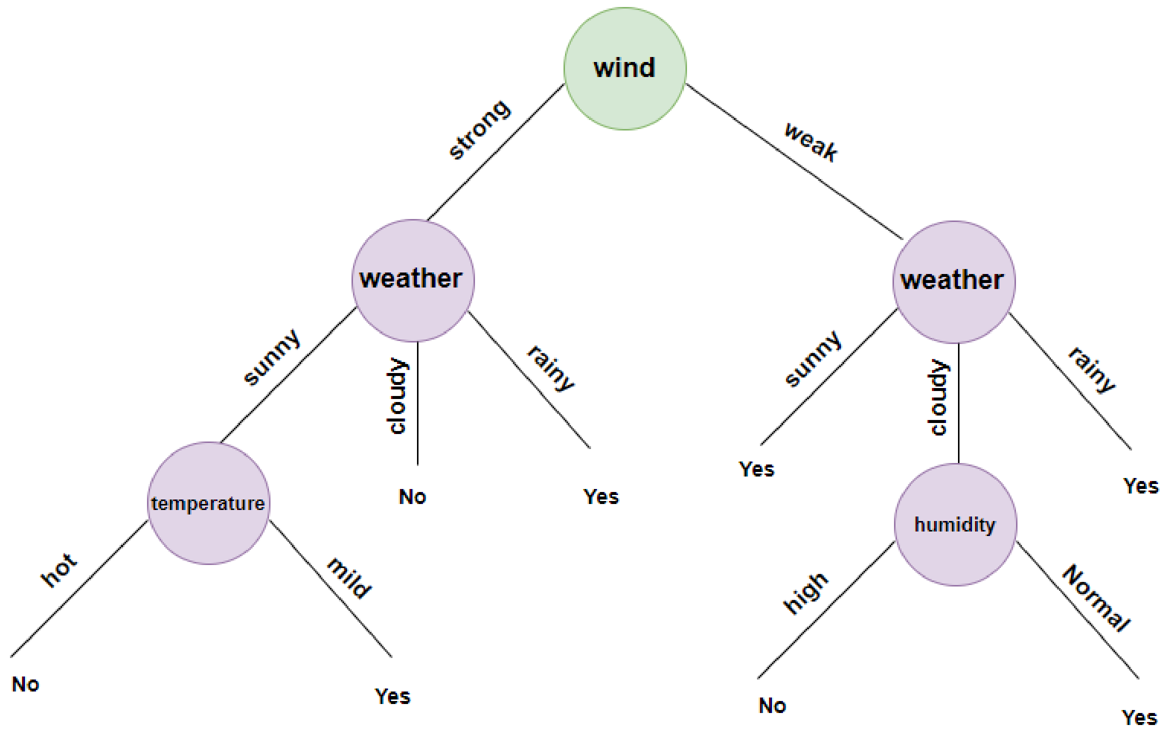
Wind = weak**, weather =** cloudy**, humidity**

- Info(high) = i[0,1] = 0
- Info(normal) = i[1,0] = 0

Entropy(**Wind =** weak**, weather =** cloudy**, humidity**) = 0

IG (**Wind =** weak**, weather =** cloudy**, humidity**) = 1 - 0 = 1

the **humidity** have the heighest IG So, I will choose it



If wind = strong & weather = rainy then yes

If wind = strong & weather = cloudy then no

If wind = strong & weather = sunny &temperature = hot then no

If wind = strong & weather = sunny &temperature = mild then yes

If wind = weak & weather = sunny then yes

If wind = weak & weather = rainy then yes

If wind = weak & weather = cloudy &humidity = normal then yes

If wind = weak & weather = cloudy &humidity = high then yes

Q1 (advantages and disadvantages of Gini Index and Information Gain):

Gini Index:

Advantages:

- It deals with inequality. So, it can judge the distribution pattern better.
- The definition of the Gini index is sufficiently simple to be comparable over other features.
- Works fine in larger distributions.

Disadvantages:

- Sample Bias: The validity of the Gini index can be dependent on sample size. For instance, small samples show less value, while large samples show higher values of the Gini index. This can be explained with the possibility of distribution divergence in a large specimen.
- Data Inaccuracy: The Gini index is sometimes prone to random and systematic data errors. So, in case there is any inaccurate data, it can create problems with the index value.
- Degeneracy: In some exceptional cases, the Gini index value can be the same for different distributions. So, it creates degeneracy, which is unavoidable.
- Structural Changes: A significant issue with this descriptor is that it doesn’t take count for structural changes. Population changes and other structural changes can divert the pattern of distribution.

Information Gain:

Advantages:

- Leafs with a small number of instances are assigned less weight.
- It favors dividing data into bigger but homogeneous groups. This approach is usually more stable and also chooses the most impactful features close to the root of the tree.

Disadvantages:

- It tends to choose attributes with more values. In some cases, such attributes may not provide much valuable information.
- Natural bias of information gain: it favours attributes with many possible values.
- An attributes (variable) with many distinct values, the information gain fails to accurately discriminate among the attributes.
- It doesn't work good for attributes with large number of distinct values (over fitting issue).

Programming Questions:

Required Libraries:

```
import pandas as pd, numpy as np, random, math, warnings, itertools
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from mlxtend.plotting import plot_decision_regions
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, plot_confusion_matrix, classif
ication_report, confusion_matrix
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, V
otingClassifier
from sklearn.datasets import make_circles
from sklearn.base import clone
from sklearn.tree import DecisionTreeClassifier, plot_tree
from matplotlib.colors import ListedColormap
warnings.filterwarnings('ignore')
```

Data set:

```
rs = 123
X, y = make_circles(300, noise=0.1, random_state=rs)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=rs)
```

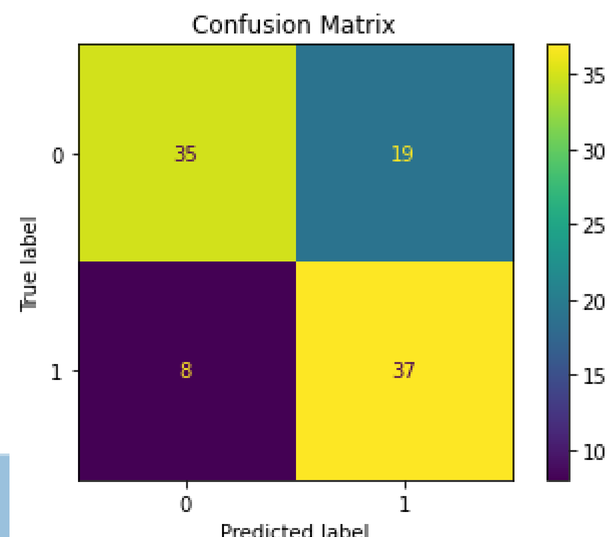
Decision Tree:

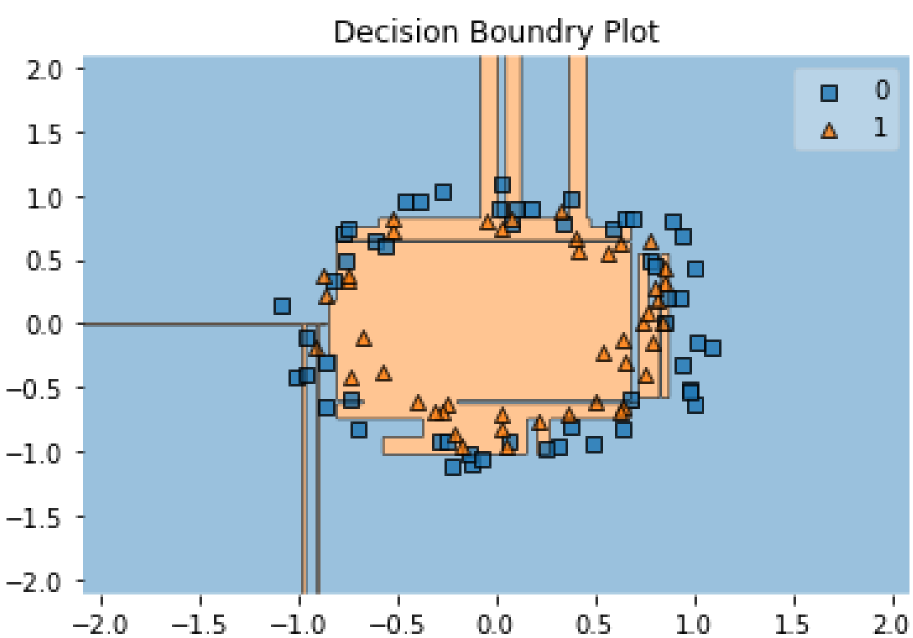
Q4: Apply decision tree to classify testing set, get the accuracy of the result, and plot the decision boundary.

```
model = DecisionTreeClassifier()
name = type(model).__name__
model.fit(X_train, y_train)

# Model Evaluation
y_predict = model.predict(X_test)
accVal = accuracy_score(y_predict, y_test) * 100
print('Using {} Algorithm'.format(name))
```

```
print('============================')
print('Accuracy: {}%'.format(accVal))
print('Classification Report: \n', classification_report(y_predict, y_test
))
plot_confusion_matrix(model, X_test, y_test, values_format='d')
plt.title('Confusion Matrix')
plt.show()

# Decision Boundary
plot_decision_regions(X_test, y_test, model)
plt.title('Decision Boundry Plot')
plt.show()

# Tree Plotting
plt.figure(figsize=(20,20))
plot_tree(model, fontsize=10)
plt.title('Decision Tree Plot', fontsize=18)
plt.show()
```

```
Using Decision Tree Classifier Algorithm
============================
Accuracy: 72.72727272727273%
Classification Report: 
              precision    recall  f1-score   support

           0       0.65      0.81      0.72        43
           1       0.82      0.66      0.73        56

    accuracy                           0.73        99
   macro avg       0.74      0.74      0.73        99
weighted avg       0.75      0.73      0.73        99
```





Bagging

Q5: Using decision tree as base-estimator and write bagging algorithm from scratch, set the number of estimators as 2, 5, 15, 20 respectively, and generate the results accordingly (i.e., accuracy and decision boundary).

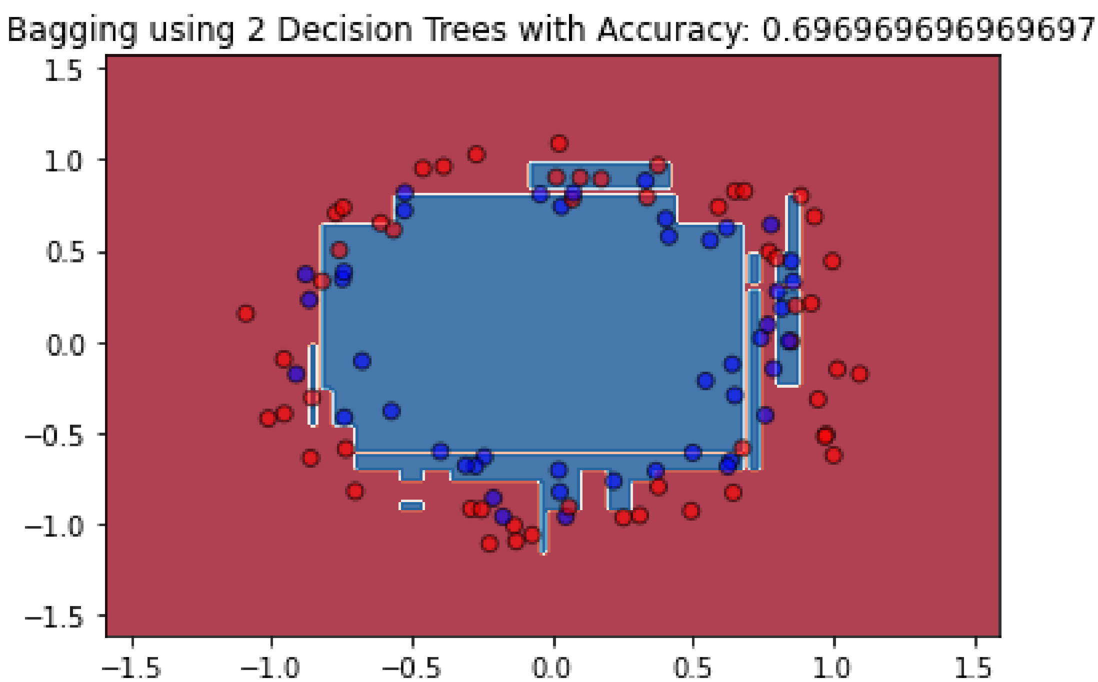
```
class BaggingClassifier:
    def __init__(self, estimator = DecisionTreeClassifier(), n_estimators = 2, sample_size = 1.0):
        self.estimator_ = estimator
        self.n_estimators_ = n_estimators
        self.estimators_accs_ = []
        self.sample_size_ = sample_size
    # End init

    def fit(self, X, y):
        self.estimators_ = [clone(self.estimator_) for i in range(self.n_estimators_)]
        for estimator in self.estimators_:
            train_size = X.shape[0]
            n_samples = round(self.sample_size_ * train_size)
            samples = np.random.choice([i for i in range(train_size)], n_samples, replace=True)
            Xtrain = X[samples, :]
            Ytrain = y[samples]
            estimator.fit(Xtrain, Ytrain)
        # End For
    # End of Func

    def predict(self, X, y):
        y_predicts = []
        real_preds = []
        acc_val = 0

        for estimator in self.estimators_:
            prediction = estimator.predict(X)
```

```
            acc_val = accuracy_score(prediction, y)
            self.estimators_accs_.append(acc_val)
            y_predicts.append(prediction)
        # End For

        y_predicts = np.array(y_predicts).T

        for pred in y_predicts:
            unique_values, count_values = np.unique(pred, return_counts=Tr
ue)
            real_preds.append(unique_values[np.argmax(count_values)])
        # End For

        return np.array(real_preds)
    # End of Func

    def predict(self, X):
        y_predicts = []
        real_preds = []
        acc_val = 0

        for estimator in self.estimators_:
            prediction = estimator.predict(X)
            y_predicts.append(prediction)
        # End For

        y_predicts = np.array(y_predicts).T

        for pred in y_predicts:
            unique_values, count_values = np.unique(pred, return_counts=Tr
ue)
            real_preds.append(unique_values[np.argmax(count_values)])
        # End For

        return np.array(real_preds)
```

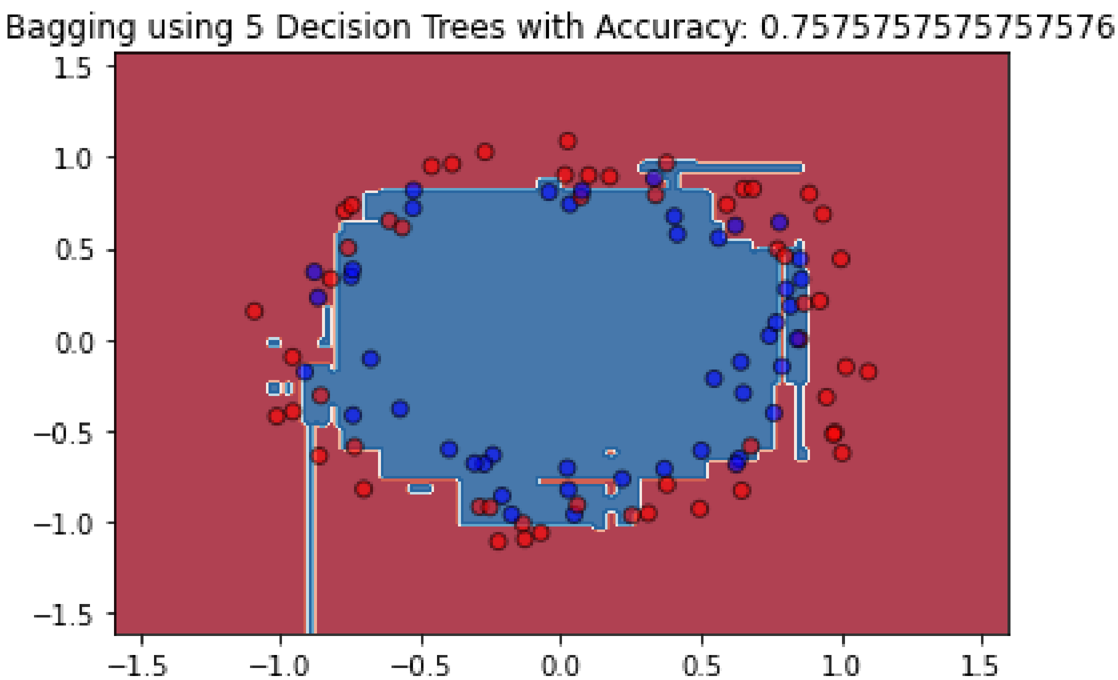
```
    # End of Func

    def plt_decision_boundry(self,X, y, n_cols=4):
        # Plotting Figures
        n_rows = math.ceil(self.n_estimators_ / n_cols)
        gs = gridspec.GridSpec(n_rows, n_cols)
        fig = plt.figure(figsize=(n_cols * 5, n_rows * 5))

        j = 1
        for clf, acc_val, grd in zip(self.estimators_, self.estimators_acc
s_, itertools.product([0, 1, 2, 3], repeat=2)):
            ax = plt.subplot(gs[grd[0], grd[1]])
            fig = plot_decision_regions(X=X, y=y, clf=clf, legend=2)
            plt.title('DT trained using Bag({}) acc: {}%'.format(j, round(
acc_val * 100, 3)))
            j += 1
        # End For
        plt.show()
    # End of Func
# End Class

def plotEstimator(teX, teY, estimator, title=''):
    h = .02
    x_min, x_max = teX[:, 0].min() - .5, teX[:, 0].max() + .5
    y_min, y_max = teX[:, 1].min() - .5, teX[:, 1].max() + .5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_ma
x, h))
    cm = plt.cm.RdBu
    cm_bright = ListedColormap(['#FF0000', '#0000FF'])
    Z = estimator.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=cm, alpha=0.8)
    plt.scatter(teX[:, 0], teX[:, 1], c=teY, cmap=cm_bright, edgecolors='k
', alpha=0.6)
    # plt.legend()
```

```
    plt.title(title)
    plt.show()
```

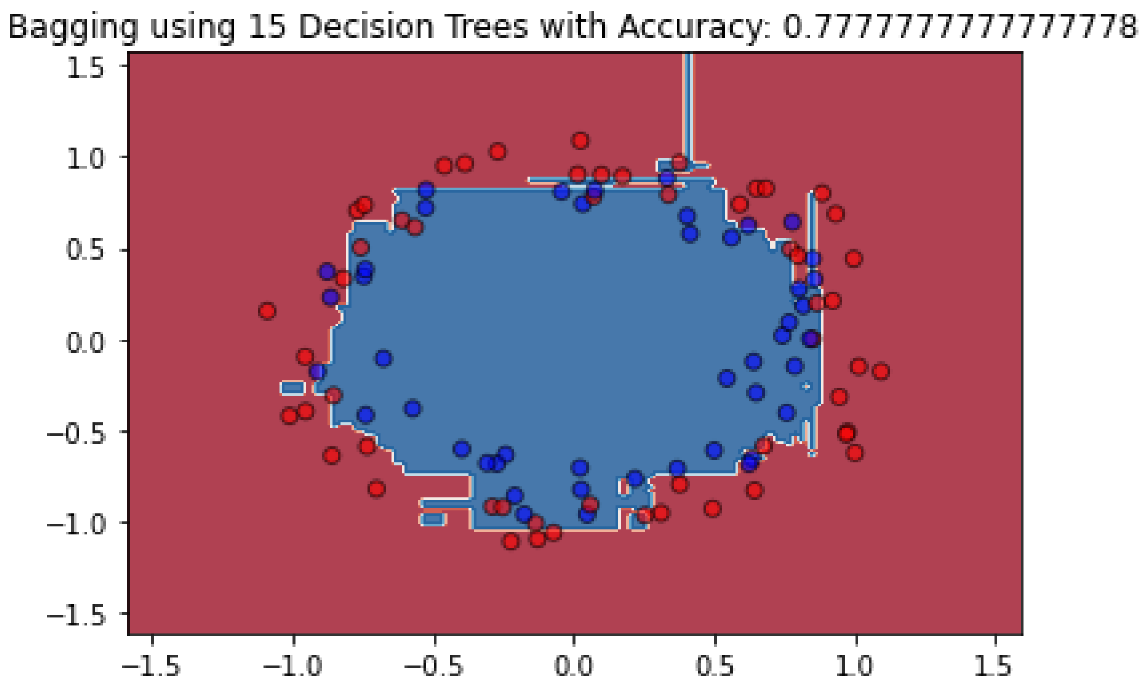
```
n_ests = 2
cls = BaggingClassifier(n_estimators = n_ests)
cls.fit(X_train, y_train)
ypred = cls.predict(X_test)
acc = accuracy_score(ypred, y_test)
title = "Bagging using {} Decision Trees with Accuracy: {}".format(n_ests, acc)
plotEstimator(X_test, y_test, cls, title=title)
```



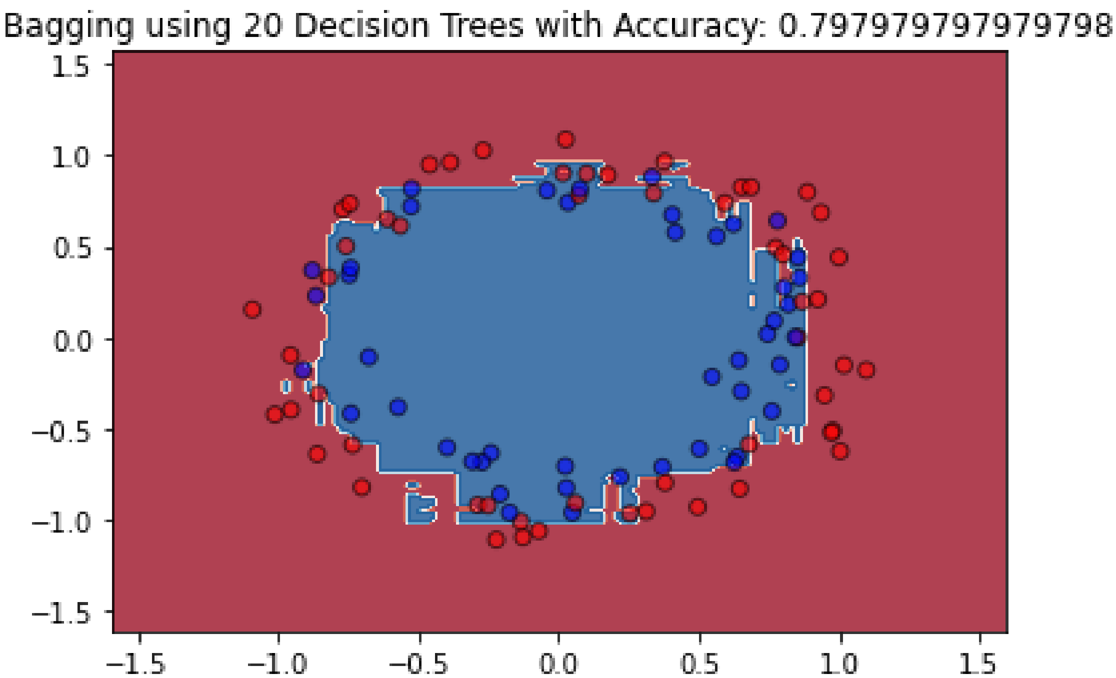
```
n_ests = 5
cls = BaggingClassifier(n_estimators = n_ests)
cls.fit(X_train, y_train)
ypred = cls.predict(X_test)
acc = accuracy_score(ypred, y_test)
title = "Bagging using {} Decision Trees with Accuracy: {}".format(n_ests, acc)
plotEstimator(X_test, y_test, cls, title=title)
```



```
n_ests = 15
cls = BaggingClassifier(n_estimators = n_ests)
cls.fit(X_train, y_train)
ypred = cls.predict(X_test)
acc = accuracy_score(ypred, y_test)
title = "Bagging using {} Decision Trees with Accuracy: {}".format(n_ests, acc)
plotEstimator(X_test, y_test, cls, title=title)
```



```
n_ests = 20
cls = BaggingClassifier(n_estimators = n_ests)
cls.fit(X_train, y_train)
ypred = cls.predict(X_test)
acc = accuracy_score(ypred, y_test)
title = "Bagging using {} Decision Trees with Accuracy: {}".format(n_ests, acc)
plotEstimator(X_test, y_test, cls, title=title)
```



Q6: Explain why bagging can reduce the variance and mitigate the over-fitting problem.

Bagging uses complex base models and tries to “smooth out” their predictions.

- It trains a large number of “strong” learners in parallel.
- A strong learner is a model that’s relatively unconstrained.
- Bagging then combines all the strong learners together in order to “smooth out” their predictions.

Bagging decreases variance through:

- Building more advanced models of complex data sets.
- Specifically, the bagging approach creates subsets which are often overlapping to model the data in a more involved way.

Boosting:

Q7: There are 2 important hyper-parameters in AdaBoost, i.e., the number of estimators (ne), and learning rate (lr). Please plot 12subfigures as the following table's setup. Each figure should plot the decision boundary and each of their title should be the same format as {n_estimaotrs}, {learning_rate}, {accuracy}.

```
# Training Phase
n_estimators = [10, 50, 100, 200]
learning_rates = [0.1, 1, 2]
models = []
y_pred_vals = []
plt_titles = []
acc_vals = []

for lr in learning_rates:
    for ne in n_estimators:

        # Model Training
        mod-
el = AdaBoostClassifier(n_estimators=ne, learning_rate=lr, random_state=0)
        name = type(model).__name__
        model.fit(X_train, y_train)
        models.append(model)

        # Model Evaluation
        y_predict = model.predict(X_test)
        accVal = round(accuracy_score(y_predict, y_test) * 100, 3)
        plt_title = 'ne: {} , lr: {}, acc: {}'.format(ne, lr, accVal)
        plt_titles.append(plt_title)
        y_pred_vals.append(y_predict)
        acc_vals.append(accVal)

# Plotting Figures
gs = gridspec.GridSpec(len(learning_rates), len(n_estimators))
fig = plt.figure(figsize=(16, 13))
```

```
for clf, lab, grd in zip(models, plt_titles, itertools.product([0, 1, 2, 3
], repeat=2)):
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=X, y=y, clf=clf, legend=2)
    plt.title(lab)
plt.show()
```