# Lab 8 : Decision Tree and Bagging

## I. Prerequisites

**scikit-learn**: Machine Learning package in Python

To install the scikit-learn in the VirtualMachine,

1). Open the terminal

2). Type the following command. And enter *cis4340* when password is asked.

```
sudo pip install scikit-learn
```

## II. Data set and preprocessing

We use the UCI car data set in this tutorial. We dropped several rows, deleted the 'car_name' column and 'model' column, which left us 6 columns. Furthermore, we transferred the 'mpg' column as the target (mpg <= 23 as class 0, mpg > 23 as class 1).

1). Import necessary toolboxes

```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.ensemble import BaggingClassifier
from sklearn import cross_validation
from sklearn.externals.six import StringIO
```

2). Load the car data and preprocessing

```
data = pd.read_csv("http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data-original",
delim_whitespace = True, header=None, names = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration',
'model', 'origin', 'car_name'])

data = data.dropna()
data = data[~ data.cylinders.isin([3, 5, 7])]
data.cylinders = data.cylinders.astype(np.int)
```

3). Specify the samples matrix X with size [n_samples, n_features],
and array Y of integer values, size [n_samples], holding the class labels for the training samples.

```
X = data[['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration']]
Y = data['mpg']
Y[Y <=23 ] = 0
Y[Y > 23 ] = 1
```

## III. Building a Decision Tree Classifier

1). Split (X,Y) into train set (Xtrain, Ytrain), and test set (Xtest, Ytest).

```
Xtrain, Xtest, Ytrain, Ytest = cross_validation.train_test_split(X, Y, test_size=0.3, random_state=0)
```

**Q. How many examples are in the train and test sets?**

2). Grow a decision tree classifier using the train set (Xtrain, Ytrain).

```
clf = DecisionTreeClassifier(random_state = 0)
clf = clf.fit(Xtrain, Ytrain)
```

This process is growing a very large tree, where each leaf node is pure (contains training examples with the same class). Let us see how the tree looks like.

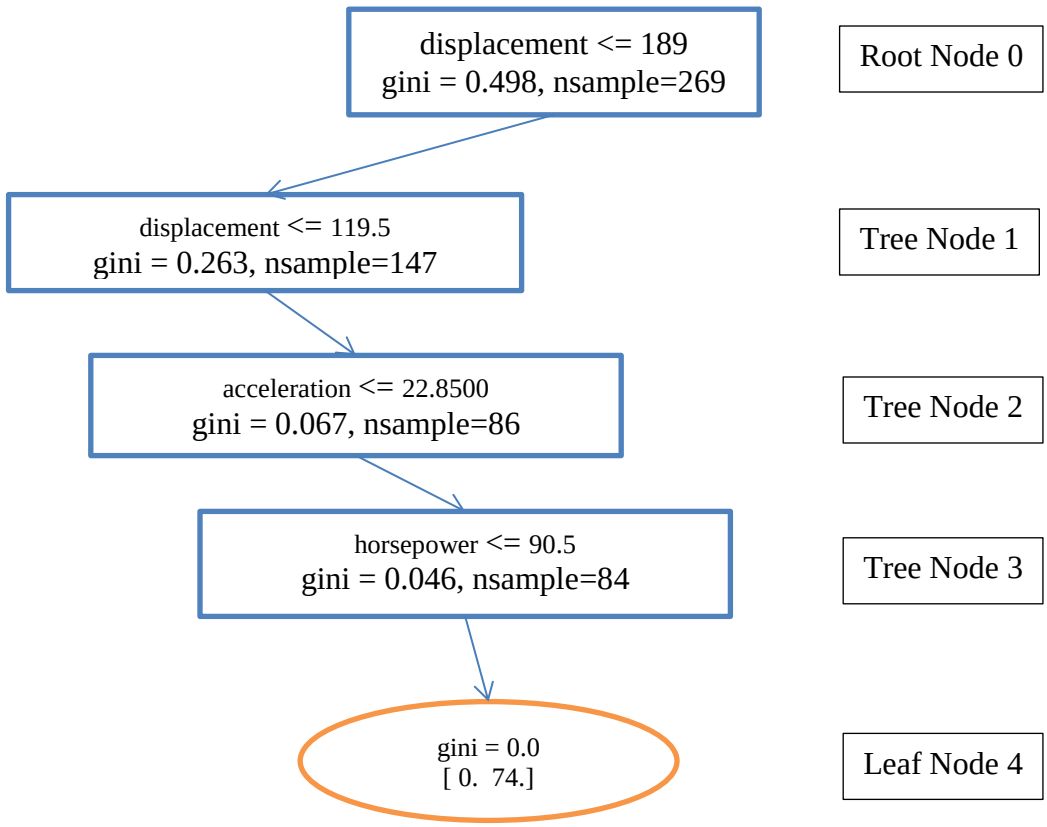
* Visualize the decision tree

```
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data, feature_names = ['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration'])
print(dot_data.getvalue())
```

You will see the text output of the decision tree:

```
0 [label="displacement <= 189.5000\ngini = 0.498832243888\nsamples = 269", shape="box"] ;
1 [label="displacement <= 119.5000\ngini = 0.263964089037\nsamples = 147", shape="box"] ;
0 -> 1 ;
2 [label="acceleration <= 22.8500\ngini = 0.0673336938886\nsamples = 86", shape="box"] ;
1 -> 2 ;
3 [label="horsepower <= 90.5000\ngini = 0.046485260771\nsamples = 84", shape="box"] ;
2 -> 3 ;
4 [label="gini = 0.0000\nsamples = 74\nvalue = [  0.  74.]", shape="box"] ;
3 -> 4 ;
```

It actually represents the following figure:




**Q. What is the question at the root of the tree?**
**Q. How deep it the tree?**

As we discussed in the class, the tree obtained in this way will be very accurate on the training data, but not very accurate

on the test data. Let us calculate the accuracy of the tree on the train and test data.

3). Predict labels on train and test data using the decision tree.

```
predict_train = clf.predict(Xtrain)
predict_test = clf.predict(Xtest)
```

4). Calculate the accuracy on training and test data.

```
accuracy_train = sum(predict_train == Ytrain) / float(len(Ytrain))
print(accuracy_train)
accuracy_test = sum(predict_test == Ytest) / float(len(Ytest))
print(accuracy_test)
```

**Q. Discuss the difference in accuracies. Is it consistent with what we discussed in the class?**

### IV. Build a Decision Tree with Early Stopping

1). One way to prevent overfitting is to stop growing the tree before it becomes too large. There is a parameter in the decision tree code that specifies the minimum size of leaf nodes (defined as the minimum number of training examples ending up in any leaf).

```
leaf = 5
clf = DecisionTreeClassifier(min_samples_leaf = leaf)
clf = clf.fit(Xtrain, Ytrain)
```

Note: if we do not specify min_samples_leaf, the default value is 1.

3). Predict labels on train and test data using the decision tree.

```
predict_train = clf.predict(Xtrain)
predict_test = clf.predict(Xtest)
```

4). Calculate the accuracy on training and test data.

```
accuracy_train = sum(predict_train == Ytrain) / float(len(Ytrain))
print(accuracy_train)
accuracy_test = sum(predict_test == Ytest) / float(len(Ytest))
print(accuracy_test)
```

**Q. Discuss the difference in accuracies. How does it compare with the first decision tree you built?**

### V. A Bagging Classifier with DT as the base Classifier

1). Train 100 DTs by samplilng 0.8 from (Xtrain, Ytrain) each time.

```
n_estimators = 100
leaf = 1
bag_trees = []
for i in range(n_estimators):
    ratio = 0.8
    rs = np.random.choice(len(Ytrain), np.floor(len(Ytrain) * ratio))
    Xtrain_rs, Ytrain_rs = Xtrain[rs,:], Ytrain[rs]
    clf = DecisionTreeClassifier(min_samples_leaf = leaf)
    clf = clf.fit(Xtrain_rs, Ytrain_rs)
    bag_trees.append(clf)
```

2). Get the prediction for each example in (Xtest) with each of the 100 DTs.

```
predictions = np.zeros((len(Ytest), n_estimators))
for i in range(n_estimators):
    predictions[:, i] = bag_trees[i].predict(Xtest)
```

3). Select the final label for each example which is voted by the majority of the 100 DTs.

```
predict_bagging = np.zeros((len(Ytest), ))
for i in range(len(Ytest)):
    most_vote = 1 if sum(predictions[i, :]) >= n_estimators/2 else 0
    predict_bagging[i] =  most_vote
```

4). Report the accuracy.

```
accuracy = sum(predict_bagging == Ytest) / float(len(Ytest))
print('Test accuracy with bagging  = ' + str(accuracy))
```

### VI. Tasks and Submission

**Task 0**. Answer all the questions appear in the tutorial

**Task 1**. For the early stopping parameter min_samples_leaf, set its values to 1, 5, 10. Report the training and testing accuracy for each value respectively.

**Task 2**. By default, DecisionTreeClassifer use *criterion = 'gini'*, which stands for Gini impurity to split a node in the tree. Repeat III with the *criterion = 'entropy'* . Report the train and test accuracy.

**Task 3**. Change the n_estimator in V and draw the trend of test accuracy when increasing the n_estimator.

**Task 4**. Download the bank_data.csv. The description is listed here . Your task is to predict the **pep** field. Preprocess your data set using Python, such that the string values are transferred to numerical values. Then, repeat the II, III, IV, V in the tutorial, report your accuracies and discoveries. The TA will assist you in preprocessing the data if necessary.

Submit your code and report through blackboard. You grading will heavily depends on your discoveries and writing details. Try to write as nice as possible and experiment as much as possible.