Decision Tree

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import plot_tree,export_text
import matplotlib.pyplot as plt
```

```
df = pd.read_csv("data/BikeRental/hour.csv")
df.head()
```

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0.0	3	13	16
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727	0.80	0.0	8	32	40
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727	0.80	0.0	5	27	32
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0.0	3	10	13
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0.0	0	1	1

```
cols_to_drop = ['instant', 'dteday','weekday', 'casual', 'registered']
df = df.drop(columns = cols_to_drop)
df.head()
```

	season	yr	mnth	hr	holiday	workingday	weathersit	temp	atemp	hum	windspeed	cnt
0	1	0	1	0	0	0	1	0.24	0.2879	0.81	0.0	16
1	1	0	1	1	0	0	1	0.22	0.2727	0.80	0.0	40
2	1	0	1	2	0	0	1	0.22	0.2727	0.80	0.0	32
3	1	0	1	3	0	0	1	0.24	0.2879	0.75	0.0	13
4	1	0	1	4	0	0	1	0.24	0.2879	0.75	0.0	1

```
medCount = df["cnt"].median()
medCount
```

```
142.0
```

We will covert the `cnt` column, which the number of hirings as categorical as the following:

```
if cnt >= median:
   cnt =  1
else
   cnt = 0
```

```
def cutAtMedian(val):
    return 1 if val >= medCount else 0
```

```
df["cnt"] = df["cnt"].apply(cutAtMedian)
```

```
# save this file as cleaned
df.to_csv("data/BikeRental/cleaned_hour.csv", index=False)
```

```
#loading dataset
def load_dataset(fname):
    data = pd.read_csv(fname, header=0, skip_blank_lines=True)
    dataset = data.values
    X = dataset[:,:-1]
    y = dataset[:,-1]
    cols = data.columns
    return X,y, cols
```

```
#Train Test split
X, y, cols = load_dataset('data/BikeRental/cleaned_hour.csv')
X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size=0.33,random_state=1)
print('Train', X_train.shape, y_train.shape)
print('Test', X_test.shape, y_test.shape)
```

```
Train (11643, 11) (11643,)
Test (5736, 11) (5736,)
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

```
#Create a decision tree classifier object
dt = DecisionTreeClassifier(criterion='entropy')
```

```
#Train the decision tree
dt = dt.fit(X_train, y_train)
#predict the response for the test data
yhat=dt.predict(X_test)
```

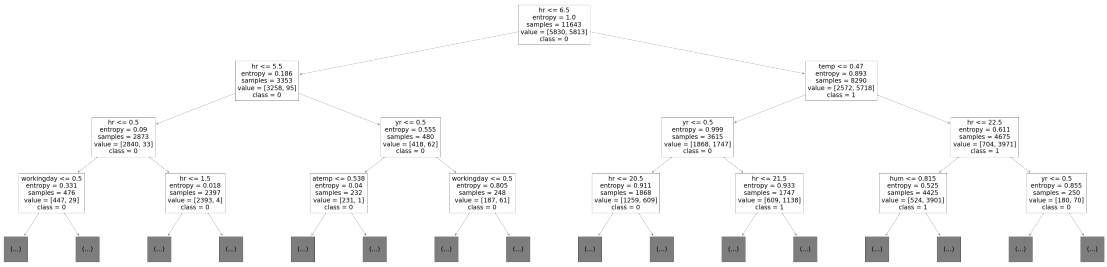
```
#Evaluate the model
accuracy=accuracy_score(y_test,yhat)
print("Accuracy: %.2f" %(accuracy*100))
```

```
Accuracy: 90.66
```

```
plt.figure(figsize=(80,20))
plot_tree(dt,
          feature_names=cols[:-1],
          class_names=['0','1'],
          max_depth=3
         )
```

```
[Text(2232.0, 978.48, 'hr <= 6.5\nentropy = 1.0\nsamples = 11643\nvalue =
[5830, 5813]\nclass = 0'),
 Text(1116.0, 761.0400000000001, 'hr <= 5.5\nentropy = 0.186\nsamples =
3353\nvalue = [3258, 95]\nclass = 0'),
 Text(558.0, 543.6, 'hr <= 0.5\nentropy = 0.09\nsamples = 2873\nvalue = [2840,
33]\nclass = 0'),
 Text(279.0, 326.1600000000001, 'workingday <= 0.5\nentropy = 0.331\nsamples =
476\nvalue = [447, 29]\nclass = 0'),
 Text(139.5, 108.72000000000003, '\n  (...)  \n'),
 Text(418.5, 108.72000000000003, '\n  (...)  \n'),
 Text(837.0, 326.1600000000001, 'hr <= 1.5\nentropy = 0.018\nsamples =
2397\nvalue = [2393, 4]\nclass = 0'),
 Text(697.5, 108.72000000000003, '\n  (...)  \n'),
 Text(976.5, 108.72000000000003, '\n  (...)  \n'),
 Text(1674.0, 543.6, 'yr <= 0.5\nentropy = 0.555\nsamples = 480\nvalue = [418,
62]\nclass = 0'),
 Text(1395.0, 326.1600000000001, 'atemp <= 0.538\nentropy = 0.04\nsamples =
232\nvalue = [231, 1]\nclass = 0'),
 Text(1255.5, 108.72000000000003, '\n  (...)  \n'),
 Text(1534.5, 108.72000000000003, '\n  (...)  \n'),
 Text(1953.0, 326.1600000000001, 'workingday <= 0.5\nentropy = 0.805\nsamples
= 248\nvalue = [187, 61]\nclass = 0'),
 Text(1813.5, 108.72000000000003, '\n  (...)  \n'),
```

```
 Text(2092.5, 108.72000000000003, '\n  (...)  \n'),
 Text(3348.0, 761.0400000000001, 'temp <= 0.47\nentropy = 0.893\nsamples =
8290\nvalue = [2572, 5718]\nclass = 1'),
 Text(2790.0, 543.6, 'yr <= 0.5\nentropy = 0.999\nsamples = 3615\nvalue =
[1868, 1747]\nclass = 0'),
 Text(2511.0, 326.1600000000001, 'hr <= 20.5\nentropy = 0.911\nsamples =
1868\nvalue = [1259, 609]\nclass = 0'),
 Text(2371.5, 108.72000000000003, '\n  (...)  \n'),
 Text(2650.5, 108.72000000000003, '\n  (...)  \n'),
 Text(3069.0, 326.1600000000001, 'hr <= 21.5\nentropy = 0.933\nsamples =
1747\nvalue = [609, 1138]\nclass = 1'),
 Text(2929.5, 108.72000000000003, '\n  (...)  \n'),
 Text(3208.5, 108.72000000000003, '\n  (...)  \n'),
 Text(3906.0, 543.6, 'hr <= 22.5\nentropy = 0.611\nsamples = 4675\nvalue =
[704, 3971]\nclass = 1'),
 Text(3627.0, 326.1600000000001, 'hum <= 0.815\nentropy = 0.525\nsamples =
4425\nvalue = [524, 3901]\nclass = 1'),
 Text(3487.5, 108.72000000000003, '\n  (...)  \n'),
 Text(3766.5, 108.72000000000003, '\n  (...)  \n'),
 Text(4185.0, 326.1600000000001, 'yr <= 0.5\nentropy = 0.855\nsamples =
250\nvalue = [180, 70]\nclass = 0'),
 Text(4045.5, 108.72000000000003, '\n  (...)  \n'),
 Text(4324.5, 108.72000000000003, '\n  (...)  \n')]
```




```
# hyperparameter tuning
dt.tree_.max_depth
```

```
25
```

```
acc = []
for max_d in range(1,dt.tree_.max_depth):
    dt=DecisionTreeClassifier(max_depth=max_d,random_state=2)
    dt.fit(X_train,y_train)
    train_accuracy = dt.score(X_train, y_train)
    val_accuracy = dt.score(X_test,y_test)
    acc.append(val_accuracy.round(2))
```

```
    print(f'max_depth: {max_d}\t \
            Training Accuracy: {train_accuracy:.2f} \
            Validation accuracy: {val_accuracy:.2f}')
```

```
max_depth: 1                Training Accuracy: 0.77             Validation
accuracy: 0.77
max_depth: 2                Training Accuracy: 0.80             Validation
accuracy: 0.81
max_depth: 3                Training Accuracy: 0.84             Validation
accuracy: 0.84
max_depth: 4                Training Accuracy: 0.85             Validation
accuracy: 0.85
max_depth: 5                Training Accuracy: 0.87             Validation
accuracy: 0.88
max_depth: 6                Training Accuracy: 0.88             Validation
accuracy: 0.88
max_depth: 7                Training Accuracy: 0.89             Validation
accuracy: 0.89
max_depth: 8                Training Accuracy: 0.91             Validation
accuracy: 0.89
max_depth: 9                Training Accuracy: 0.93             Validation
accuracy: 0.90
max_depth: 10               Training Accuracy: 0.95             Validation
accuracy: 0.91
max_depth: 11               Training Accuracy: 0.96             Validation
accuracy: 0.91
max_depth: 12               Training Accuracy: 0.97             Validation
accuracy: 0.91
max_depth: 13               Training Accuracy: 0.98             Validation
accuracy: 0.91
max_depth: 14               Training Accuracy: 0.99             Validation
accuracy: 0.91
max_depth: 15               Training Accuracy: 0.99             Validation
accuracy: 0.91
max_depth: 16               Training Accuracy: 0.99             Validation
accuracy: 0.91
max_depth: 17               Training Accuracy: 1.00             Validation
accuracy: 0.90
max_depth: 18               Training Accuracy: 1.00             Validation
accuracy: 0.91
max_depth: 19               Training Accuracy: 1.00             Validation
accuracy: 0.90
max_depth: 20               Training Accuracy: 1.00             Validation
accuracy: 0.90
max_depth: 21               Training Accuracy: 1.00             Validation
accuracy: 0.91
max_depth: 22               Training Accuracy: 1.00             Validation
```

```
accuracy: 0.91
max_depth: 23                Training Accuracy: 1.00            Validation
accuracy: 0.91
max_depth: 24                Training Accuracy: 1.00            Validation
accuracy: 0.91
```

```
max_i = np.argmax(acc)
print(f"The maximum accuracy is at the  depth: {max_i+1}
({acc[max_i]*100:.1f}%)")
```

```
The maximum accuracy is at the  depth: 10 (91.0%)
```