Machine Learning

Lab Sheet 6

Decision Tree

S Abhishek

AM.EN.U4CSE19147

Colab Link

```
import numpy as np
import pandas as pd
import random
import math
from math import log2
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

```
import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('/car.csv')
df.head()
```

```
  vhigh vhigh.1  2 2.1  small   low  unacc
0  vhigh   vhigh  2   2  small   med  unacc
1  vhigh   vhigh  2   2  small  high  unacc
2  vhigh   vhigh  2   2    med   low  unacc
3  vhigh   vhigh  2   2    med   med  unacc
4  vhigh   vhigh  2   2    med  high  unacc
```

```
df.shape
```

```
(1727, 7)
```

```
df.info
```

```
<bound method DataFrame.info of       vhigh vhigh.1      2   2.1  small   low
unacc
0     vhigh   vhigh      2     2  small   med  unacc
1     vhigh   vhigh      2     2  small  high  unacc
2     vhigh   vhigh      2     2    med   low  unacc
3     vhigh   vhigh      2     2    med   med  unacc
4     vhigh   vhigh      2     2    med  high  unacc
...     ...     ...    ...   ...    ...   ...    ...
1722    low     low  5more  more    med   med   good
```

1723 low low 5more more med high vgood

1724 low low 5more more big low unacc

1725 low low 5more more big med good

1726 low low 5more more big high vgood

[1727 rows x 7 columns]>

```
df['vhigh'] = df['vhigh'].replace({'vhigh' :3 , 'high' : 2, 'med' : 1, 'low': 0})
df['vhigh.1'] = df['vhigh.1'].replace({'vhigh' :3 , 'high' : 2, 'med' : 1, 'low': 0})
df['low'] = df['low'].replace({'high' : 2, 'med' : 1, 'low': 0}
df['small'] = df['small'].replace({'big' : 2, 'med' : 1, 'small': 0})
df['2'] = df['2'].replace({'Smore': 5})
df['2.1'] = df['2.1'].replace({'more': 5})
```

1 - Split data into training and test set.

```
X = df.iloc[:,:-1].values
y = df.iloc[:,-1:].values
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2, stratify = y,
random_state=1)
```

```
X_train.shape, X_test.shape
```

((1381, 6), (346, 6))

2 - Calculate the entropy of a dataset

- Define a function to calculate the entropy of a dataset, S, based on the target variable, where pi is the probability of class i.

Entropy(S) = Σpilog(pi)

```
def calcEntropy(S):
  entrop = 0
  for i in range(S.iloc[:,-1].nunique()):
    prob = S.iloc[:,-1].value_counts()[i]/S.shape[0]
  entrop = entrop + (prob * np.log2(prob))
  return (entrop*-1)
calcEntropy(df)
```

0.17808888016359614

3 - Consider 'buying' attribute of car dataset.

- Find unique values in the dataset for 'buying' attribute.
- Find expected information gain when 'buying' attribute becomes known
- Gain(S,buying)=Entropy(S)−1/|||S|||Σ|Sv|Entropy(Sv)
- Where Sv is the subset of dataset with v value in buying attribute.

```
df['unacc'].value_counts()
df['unacc'] = df['unacc'].replace({'unacc' : 3 , 'acc' : 2, 'good' : 1, 'vgood': 0})
def calcEntropyColumn(col):
```

```
  cnt = np.bincount(col)
  prob = cnt / len(col)
  entropy = 0
  for i in prob:
    if i > 0:
      entropy += i * np.log2(i)
  return -entropy
def calcInformationGain(data, split_name, target_name) :
  entropy = calcEntropyColumn(data[target_name])
  values = data[split_name].unique()
  split = []
  for i in values:
    split.append(data[data[split_name] == i])
  sub = 0
  for subset in split:
    prob = (subset.shape[0] / data.shape[0])
    sub += prob * calcEntropyColumn(subset[target_name])
  return entropy - sub
calcInformationGain(df,'vhigh','unacc')
```

```
0.09635953253842389
```

4 - Repeat Q.3 for all attributes

- Find the attribute with maximum gain.

```
def highestInfoGain(columns):
  gains = {}
  for col in columns[0:-1]:
    information_gain = calcInformationGain(df, col, columns[-1])
  gains[col] = information_gain
  return max(gains, key=gains.get)
```

df

	vhigh	vhigh.1	2	2.1	small	low	unacc
0	3	3	2	2	0	1	3
1	3	3	2	2	0	2	3
2	3	3	2	2	1	0	3
3	3	3	2	2	1	1	3
4	3	3	2	2	1	2	3
...	...	...	...	..	...	...	...
1722	0	0	5more	5	1	1	1
1723	0	0	5more	5	1	2	0
1724	0	0	5more	5	2	0	3
1725	0	0	5more	5	2	1	1
1726	0	0	5more	5	2	2	0

[1727 rows x 7 columns]

```
df = df[df['2'] != '5more']
```

```
highestInfoGain(df.columns)
```

```
{"type":"string"}
```

```
df
```

```
     vhigh  vhigh.1  2 2.1  small  low  unacc
0        3        3  2   2      0    1      3
1        3        3  2   2      0    2      3
2        3        3  2   2      1    0      3
3        3        3  2   2      1    1      3
4        3        3  2   2      1    2      3
...    ...      ... ..  ..    ...  ...    ...
1695     0        0  4   5      1    1      1
1696     0        0  4   5      1    2      0
1697     0        0  4   5      2    0      3
1698     0        0  4   5      2    1      1
1699     0        0  4   5      2    2      0

[1295 rows x 7 columns]
```

5 - Decision Tree

- Use the predefined function to do the training using decision tree

```
X = df.iloc[:,:-1].values
```

```
y = df.iloc[:,-1:].values
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, y,
test_size=0.3,random_state=10)

model_entropy = DecisionTreeClassifier(criterion = "entropy", random_state =
100, max_depth=3, min_samples_leaf=5)

model_entropy.fit(X_train, Y_train);

DecisionTreeClassifier(criterion='entropy', max_depth=3,
min_samples_leaf=5,random_state=100)

y_pred = model_entropy.predict(X_test)

print("Accuracy :",metrics.accuracy_score(Y_test, y_pred))

print("Precision :", metrics.precision_score(Y_test, y_pred, average = 'weighted'))

print("Recall :", metrics.recall_score(Y_test, y_pred, average = 'weighted'))
```

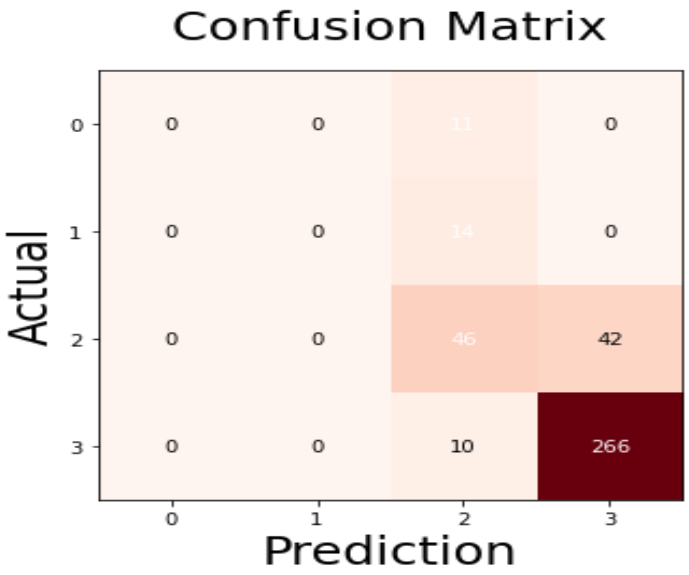
```
Accuracy : 0.8020565552699229

Precision : 0.7412312210941174

Recall : 0.8020565552699229
```

```
print(metrics.classification_report(Y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	11
1	0.00	0.00	0.00	14
2	0.57	0.52	0.54	88
3	0.86	0.96	0.91	276

accuracy			0.80	389
macro avg	0.36	0.37	0.36	389
weighted avg	0.74	0.80	0.77	389

```
conf_matrix = metrics.confusion_matrix(Y_test, y_pred)
fig, ax = plot_confusion_matrix(conf_matrix, figsize=(5, 5), cmap=plt.cm.Reds)
plt.xlabel('Prediction', fontsize=24)
plt.ylabel('Actual', fontsize=24)
plt.title('Confusion Matrix', fontsize=24)
plt.show()
```



6 - KNN vs Logitsic Regression

- Compare the results of Decision tree with KNN and Logistic regression.

KNN

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train,Y_train)

y_pred_KNN= knn.predict(X_test)

print("Accuracy :",metrics.accuracy_score(Y_test, y_pred_KNN))

print("Precision :",metrics.precision_score(Y_test, y_pred_KNN, average =
'weighted'))

print("Recall :",metrics.recall_score(Y_test, y_pred_KNN, average = 'weighted'))
```

Accuracy : 0.9305912596401028

Precision : 0.9322170015816597

Recall : 0.9305912596401028

Logistic Regression

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()

lr.fit(X_train, Y_train)

y_pred_LR = lr.predict(X_test)

print("Accuracy :",metrics.accuracy_score(Y_test, y_pred_LR))

print("Precision :",metrics.precision_score(Y_test, y_pred_LR, average =
'weighted'))
```

```
print("Recall :",metrics.recall_score(Y_test, y_pred_LR, average = 'weighted'))
```

Accuracy : 0.8483290488431876

Precision : 0.8352823261202744

Recall : 0.8483290488431876

- Decision trees cannot derive the significance of features, but Logistic Regression can.
- Decision tree is faster when compared to KNN's expensive real time execution.

Thankyou!!