Decision tree

A decision tree is a flow chart like structure having root node, internal node and leaves. Every internal node has the test condition to choose the next node, branch is the outcome of the test. The path traversed from the root node to leaf node defines classification.

ID3 Algorithm

- ID3 (Iterative Dichotomiser 3) Decision tree is an algorithm used to generate a decision tree from a dataset.
- The ID3 algorithm begins with the original set S as the root node. On each iteration of the algorithm, it iterates through every unused attribute of the set S and calculates the entropy and the information gain of that attribute.
- It then selects the attribute which has the smallest entropy (or largest information gain) value. The set S is then split or partitioned by the selected attribute to produce subsets of the data.
- The algorithm continues to recurse on each subset, considering only attributes never selected before.

Code:

```
import os
import numpy as np
import pandas as pd
import numpy as np, pandas as pd
import matplotlib.pyplot as plt
from sklearn import tree, metrics

# reading dataset
data = pd.read_csv('D:/academics/MLpractice/glass.csv')
data.head()
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 214 entries, 0 to 213
Data columns (total 10 columns):
RI      214 non-null float64
Na      214 non-null float64
Mg      214 non-null float64
Al      214 non-null float64
Si      214 non-null float64
K       214 non-null float64
Ca      214 non-null float64
Ba      214 non-null float64
Fe      214 non-null float64
Type    214 non-null int64
dtypes: float64(9), int64(1)
memory usage: 16.8 KB
```

```
# factorize data
data['Type'],type_names = pd.factorize(data['Type'])
print(type_names)
print(data['Type'].unique())
```

```
data['RI'],_ = pd.factorize(data['RI'])
data['Na'],_ = pd.factorize(data['Na'])
data['Mg'],_ = pd.factorize(data['Mg'])
data['Al'],_ = pd.factorize(data['Al'])
data['Si'],_ = pd.factorize(data['Si'])
data['K'],_ = pd.factorize(data['K'])
data['Ca'],_ = pd.factorize(data['Ca'])
data['Ba'],_ = pd.factorize(data['Ba'])
data['Fe'],_ = pd.factorize(data['Fe'])
data.head()
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	0	0	0
2	2	2	2	2	2	2	2	0	0	0
3	3	3	3	3	3	3	3	0	0	0
4	4	4	4	4	4	4	4	0	0	0

```
data.info()
```

```
RangeIndex: 214 entries, 0 to 213
Data columns (total 10 columns):
RI      214 non-null int64
Na      214 non-null int64
Mg      214 non-null int64
Al      214 non-null int64
Si      214 non-null int64
K       214 non-null int64
Ca      214 non-null int64
Ba      214 non-null int64
Fe      214 non-null int64
Type    214 non-null int64
dtypes: int64(10)
memory usage: 16.8 KB
```

```
# splitting train and test data
from sklearn.model_selection import train_test_split
from sklearn import model_selection
from sklearn.metrics import confusion_matrix, accuracy_score, roc_curve,precision_score,
recall_score, f1_score, precision_recall_curve
X = data.iloc[:,:-1]
y = data.iloc[:,-1]
# split data randomly into 70% training and 30% test
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=0.3,
random_state=0)

# train the decision tree
dtree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
```

```
dtree.fit(X_train, y_train)

# use the model to make predictions with the test data
y_pred = dtree.predict(X_test)
# how did our model perform?
count_misclassified = (y_test != y_pred).sum()
print('Misclassified samples: {}'.format(count_misclassified))
accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy: {:.2f}'.format(accuracy))

print ("\n======== Confusion Matrix =========")
conf_matrix = confusion_matrix(y_test, y_pred)
print (conf_matrix)
print ("\n======== Precision and Recall Scores =========")
print ("Precision : ", precision_score(y_test, y_pred,average='micro'))
print ("Recall : ", recall_score(y_test, y_pred, average='micro'))
```

```
Misclassified samples: 7
Accuracy: 0.89

======== Confusion Matrix =========
[[21  0  0  0  0  0]
 [ 1 24  1  0  0  0]
 [ 0  1  6  0  0  0]
 [ 0  0  0  2  0  0]
 [ 1  0  0  1  0  0]
 [ 0  1  0  1  0  5]]

======== Precision and Recall Scores =========
Precision :  0.8923076923076924
Recall :  0.8923076923076924
```

```
# decision tree visualization
import graphviz
from sklearn.tree import export_graphviz
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
feature_names = X.columns

dot_data = tree.export_graphviz(dtree, out_file=None, filled=True, rounded=True,
                                class_names='type_names')
graph = graphviz.Source(dot_data)
graph
```

Final decision tree: