Decision Tree for Classification of Breast Cancer Wisconsin Diagnostic

In [1]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,precision_score,recall_score
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
#conda install -c conda-forge pygraphviz
import graphviz
from sklearn import preprocessing

colname=['ID','Diagnosis']+['feature_%s' %i for i in range(1,31)]
#print(colname)
df = pd.read_csv('wdbc.data',header=None,names=colname)
```

1. After importing the data, let's check the if data successfully import.

In [2]:
```python
print(df.head(5))
```

```
         ID Diagnosis  feature_1  feature_2  feature_3  feature_4  feature_5  \
0    842302         M      17.99      10.38     122.80     1001.0    0.11840
1    842517         M      20.57      17.77     132.90     1326.0    0.08474
2  84300903         M      19.69      21.25     130.00     1203.0    0.10960
3  84348301         M      11.42      20.38      77.58      386.1    0.14250
4  84358402         M      20.29      14.34     135.10     1297.0    0.10030

   feature_6  feature_7  feature_8  ...  feature_21  feature_22  feature_23  \
0    0.27760     0.3001    0.14710  ...       25.38       17.33      184.60
1    0.07864     0.0869    0.07017  ...       24.99       23.41      158.80
2    0.15990     0.1974    0.12790  ...       23.57       25.53      152.50
3    0.28390     0.2414    0.10520  ...       14.91       26.50       98.87
4    0.13280     0.1980    0.10430  ...       22.54       16.67      152.20

   feature_24  feature_25  feature_26  feature_27  feature_28  feature_29  \
0      2019.0      0.1622      0.6656      0.7119      0.2654      0.4601
1      1956.0      0.1238      0.1866      0.2416      0.1860      0.2750
2      1709.0      0.1444      0.4245      0.4504      0.2430      0.3613
3       567.7      0.2098      0.8663      0.6869      0.2575      0.6638
4      1575.0      0.1374      0.2050      0.4000      0.1625      0.2364

   feature_30
0     0.11890
1     0.08902
2     0.08758
3     0.17300
4     0.07678

[5 rows x 32 columns]
```

In [3]:
```python
df.isnull().any()
```

Out[3]:
```
ID            False
Diagnosis     False
feature_1     False
feature_2     False
feature_3     False
feature_4     False
feature_5     False
feature_6     False
feature_7     False
feature_8     False
feature_9     False
feature_10    False
feature_11    False
feature_12    False
feature_13    False
feature_14    False
feature_15    False
feature_16    False
feature_17    False
feature_18    False
feature_19    False
feature_20    False
feature_21    False
feature_22    False
feature_23    False
feature_24    False
feature_25    False
feature_26    False
feature_27    False
feature_28    False
feature_29    False
feature_30    False
dtype: bool
```

1. Let's understand more about the data. We will start by getting to know the type of each column values. We see that the width and length column are represented using float64 and the name of the species uses object or string.

In [4]:
```python
df.dtypes
```

Out[4]:
```
ID              int64
Diagnosis      object
feature_1     float64
feature_2     float64
feature_3     float64
feature_4     float64
feature_5     float64
feature_6     float64
feature_7     float64
feature_8     float64
feature_9     float64
feature_10    float64
feature_11    float64
feature_12    float64
feature_13    float64
feature_14    float64
feature_15    float64
feature_16    float64
feature_17    float64
feature_18    float64
feature_19    float64
feature_20    float64
feature_21    float64
feature_22    float64
feature_23    float64
feature_24    float64
feature_25    float64
feature_26    float64
feature_27    float64
feature_28    float64
feature_29    float64
feature_30    float64
dtype: object
```

In [5]:
```python
len(df['ID'])
```

Out[5]:
```
569
```

1. Let's look at a quick summary of the data.

In [6]:
```python
df.describe()
```

Out[6]:

	ID	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	...	feature_21	feature_22	feature_23	feature_24	feature_
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	569.000000	569.000000	569.000000	569.000000	569.0000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	...	16.269190	25.677223	107.261213	880.583128	0.1323
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	...	4.833242	6.146258	33.602542	569.356993	0.0228
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	...	7.930000	12.020000	50.410000	185.200000	0.0711
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	...	13.010000	21.080000	84.110000	515.300000	0.1166
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	...	14.970000	25.410000	97.660000	686.500000	0.1313
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	...	18.790000	29.720000	125.400000	1084.000000	0.1460
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	...	36.040000	49.540000	251.200000	4254.000000	0.2226

8 rows × 31 columns

1. Everything checks out. First, I use Information Gain to construct a DT. But choose which features?

In [7]:
```python
#distribute M and B
dfm=df[df['Diagnosis']=='M']
dfb=df[df['Diagnosis']=='B']
dfm.describe()
```

Out[7]:

	ID	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	...	feature_21	feature_22	feature_23	feature_24	feature_25
count	2.120000e+02	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000	...	212.000000	212.000000	212.000000	212.000000	212.000000
mean	3.681805e+07	17.462830	21.604906	115.365377	978.376415	0.102898	0.145188	0.160775	0.087990	0.192909	...	21.134811	29.318208	141.370330	1422.286321	0.144845
std	1.378965e+08	3.203971	3.779470	21.854653	367.937978	0.012608	0.053987	0.075019	0.034374	0.027638	...	4.283569	5.434804	29.457055	597.967743	0.021870
min	8.670000e+03	10.950000	10.380000	71.900000	361.600000	0.073710	0.046050	0.023980	0.020310	0.130800	...	12.840000	16.670000	85.100000	508.100000	0.088220
25%	8.613450e+05	15.075000	19.327500	98.745000	705.300000	0.094010	0.109600	0.109525	0.064620	0.174050	...	17.730000	25.782500	119.325000	970.300000	0.130475
50%	8.953665e+05	17.325000	21.460000	114.200000	932.000000	0.102200	0.132350	0.151350	0.086280	0.189900	...	20.590000	28.945000	138.000000	1303.000000	0.143450
75%	8.911290e+06	19.590000	23.765000	129.925000	1203.750000	0.110925	0.172400	0.203050	0.103175	0.209850	...	23.807500	32.690000	159.800000	1712.750000	0.155975
max	9.112962e+08	28.110000	39.280000	188.500000	2501.000000	0.144700	0.345400	0.426800	0.201200	0.304000	...	36.040000	49.540000	251.200000	4254.000000	0.222600

8 rows × 31 columns

In [8]:
```python
dfb.describe()
```

Out[8]:

	ID	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	...	feature_21	feature_22	feature_23	feature_24	feature_25
count	3.570000e+02	357.000000	357.000000	357.000000	357.000000	357.000000	357.000000	357.000000	357.000000	357.000000	...	357.000000	357.000000	357.000000	357.000000	357.000000
mean	2.654382e+07	12.146524	17.914762	78.075406	462.790196	0.092478	0.080085	0.046058	0.025717	0.174186	...	13.379801	23.515070	87.005938	558.899440	0.124959
std	1.167397e+08	1.780512	3.995125	11.807438	134.287118	0.013446	0.033750	0.043442	0.015909	0.024807	...	1.981368	5.493955	13.527091	163.601424	0.020013
min	8.913000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	...	7.930000	12.020000	50.410000	185.200000	0.071170
25%	8.746620e+05	11.080000	15.150000	70.870000	378.200000	0.083060	0.055620	0.020310	0.015020	0.158000	...	12.080000	19.580000	78.270000	447.100000	0.110400
50%	9.089160e+05	12.200000	17.390000	78.180000	458.400000	0.090760	0.075290	0.037090	0.023440	0.171400	...	13.350000	22.820000	86.920000	547.400000	0.125400
75%	8.812816e+06	13.370000	19.760000	86.100000	551.100000	0.100700	0.097550	0.059990	0.032510	0.189000	...	14.800000	26.510000	96.590000	670.000000	0.137600
max	9.113205e+08	17.850000	33.810000	114.600000	992.100000	0.163400	0.223900	0.410800	0.085340	0.274300	...	19.820000	41.780000	127.100000	1210.000000	0.200600

8 rows × 31 columns

In [9]:
```python
#initialize
avg_m=np.zeros(30)
avg_b=np.zeros(30)
for i_feature in range(30):
    #obtain average value of feature for M and B
    avg_m[i_feature] = np.mean(dfm[lambda dfm: dfm.columns[i_feature+2]])
    avg_b[i_feature] = np.mean(dfb[lambda dfb: dfb.columns[i_feature+2]])
# find top 10 features with average value of M and B differs most
diff=abs(avg_m-avg_b)
diff_sorted = sorted(enumerate(diff),key=lambda x:x[1])
diff_sorted_index=[m[0] for m in diff_sorted]
print(diff_sorted_index[0:10])
```

```
[18, 9, 14, 19, 17, 11, 4, 15, 29, 16]
```

5.construct decision tree only from the 10 features and loop for 20 times

In [10]:
```python
X = df[['feature_18', 'feature_9', 'feature_14', 'feature_19','feature_17',
       'feature_11','feature_4','feature_15','feature_29','feature_16']].values
Y = df['Diagnosis'].values

dt1_accuracy_train=np.zeros(20)
dt1_accuracy_test=np.zeros(20)
dt1_precision=np.zeros(20)
dt1_recall=np.zeros(20)
dt2_accuracy_train=np.zeros(20)
dt2_accuracy_test=np.zeros(20)
dt2_precision=np.zeros(20)
dt2_recall=np.zeros(20)
for i_process in range(20):
    (X_train,X_test,Y_train,Y_test)=train_test_split(X, Y, random_state=2,test_size=0.3,stratify=Y)
    clf_1 = DecisionTreeClassifier(criterion='entropy')
    clf_1.fit(X_train, Y_train)
    Y_pre_train=clf_1.predict(X_train)
    Y_pre_test=clf_1.predict(X_test)
    dt1_accuracy_train[i_process]=accuracy_score(Y_train, Y_pre_train)
    dt1_accuracy_test[i_process]=accuracy_score(Y_test, Y_pre_test)
    dt1_precision[i_process]=precision_score(Y_test, Y_pre_test,pos_label='M')
    dt1_recall[i_process]=recall_score(Y_test, Y_pre_test,pos_label='M')

    dt2_depth=3
    last_recall=1
    while dt1_recall[i_process] > dt2_recall[i_process] or last_recall < dt2_recall[i_process]:
        last_recall=dt2_recall[i_process]
        dt2_depth+=1
        clf_2 = DecisionTreeClassifier(criterion='entropy',max_depth=dt2_depth)
        clf_2.fit(X_train, Y_train)
        Y_pre_train=clf_2.predict(X_train)
        Y_pre_test=clf_2.predict(X_test)
        dt2_accuracy_train[i_process]=accuracy_score(Y_train, Y_pre_train)
        dt2_accuracy_test[i_process]=accuracy_score(Y_test, Y_pre_test)
        dt2_precision[i_process]=precision_score(Y_test, Y_pre_test,pos_label='M')
        dt2_recall[i_process]=recall_score(Y_test, Y_pre_test,pos_label='M')

    dt2_recall[i_process]=last_recall

dt1_avg_accuracy_train=np.mean(dt1_accuracy_train)
dt1_avg_accuracy_test=np.mean(dt1_accuracy_test)
dt1_avg_precision=np.mean(dt1_precision)
dt1_avg_recall=np.mean(dt1_recall)
dt2_avg_accuracy_train=np.mean(dt2_accuracy_train)
dt2_avg_accuracy_test=np.mean(dt2_accuracy_test)
dt2_avg_precision=np.mean(dt2_precision)
dt2_avg_recall=np.mean(dt2_recall)
print([dt1_accuracy_train,dt2_accuracy_train])
print([dt1_accuracy_test,dt2_accuracy_test])
print([dt1_avg_accuracy_train,dt1_avg_accuracy_test,dt1_avg_precision,dt1_avg_recall])
print([dt2_avg_accuracy_train,dt2_avg_accuracy_test,dt2_avg_precision,dt2_avg_recall])
```

```
[array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1.]), array([1.        , 0.99748744, 0.99497487, 1.        , 0.99497487,
       0.99748744, 0.99748744, 1.        , 0.99748744, 0.99748744,
       0.99748744, 0.99748744, 0.99748744, 0.99748744, 0.99497487,
       0.99748744, 1.        , 0.99497487, 0.99748744, 0.99748744])]
[array([0.9122807 , 0.91812865, 0.90643275, 0.9122807 , 0.9005848 ,
       0.89473684, 0.89473684, 0.90643275, 0.88888889, 0.90643275,
       0.9005848 , 0.90643275, 0.90643275, 0.9122807 , 0.9122807 ,
       0.91812865, 0.9122807 , 0.90643275, 0.90643275, 0.90643275]), array([0.9122807 , 0.9005848 , 0.90643275, 0.9005848 , 0.9005848 ,
       0.90643275, 0.90643275, 0.9122807 , 0.9122807 , 0.9122807 ,
       0.9005848 , 0.90643275, 0.90643275, 0.90643275, 0.9005848 ,
       0.91812865, 0.91812865, 0.90643275, 0.90643275, 0.9122807 ])]
[1.0, 0.9064327485380117, 0.8811484600840431, 0.8671875]
[0.9974874371859297, 0.9076023391812866, 0.8784219174642562, 0.89765625]
```

6.construct the decision tree from all 30 features and replace depth by minimum leaf samples

In [11]:
```python
X = df[lambda df: df.columns[2:32]].values
Y = df['Diagnosis'].values

dt1_accuracy_train,dt1_accuracy_test,dt1_precision,dt1_recall =(np.zeros(20) for i in range(4))
dt2_accuracy_train,dt2_accuracy_test,dt2_precision,dt2_recall=(np.zeros(20) for i in range(4))
for i_process in range(20):
    (X_train,X_test,Y_train,Y_test)=train_test_split(X, Y, random_state=2,test_size=0.3,stratify=Y)
    clf_1 = DecisionTreeClassifier(criterion='entropy')
    clf_1.fit(X_train, Y_train)
    Y_pre_train=clf_1.predict(X_train)
    Y_pre_test=clf_1.predict(X_test)
    dt1_accuracy_train[i_process]=accuracy_score(Y_train, Y_pre_train)
    dt1_accuracy_test[i_process]=accuracy_score(Y_test, Y_pre_test)
    dt1_precision[i_process]=precision_score(Y_test, Y_pre_test,pos_label='M')
    dt1_recall[i_process]=recall_score(Y_test, Y_pre_test,pos_label='M')

    min_sample=2
    last_recall=1
    while dt1_recall[i_process] > dt2_recall[i_process] or last_recall < dt2_recall[i_process]:
        last_recall=dt2_recall[i_process]
        min_sample+=1
        clf_2 = DecisionTreeClassifier(criterion='entropy',min_samples_split=min_sample)
        clf_2.fit(X_train, Y_train)
        Y_pre_train=clf_2.predict(X_train)
        Y_pre_test=clf_2.predict(X_test)
        dt2_accuracy_train[i_process]=accuracy_score(Y_train, Y_pre_train)
        dt2_accuracy_test[i_process]=accuracy_score(Y_test, Y_pre_test)
        dt2_precision[i_process]=precision_score(Y_test, Y_pre_test,pos_label='M')
        dt2_recall[i_process]=recall_score(Y_test, Y_pre_test,pos_label='M')

    dt2_recall[i_process]=last_recall

dt1_avg_accuracy_train=np.mean(dt1_accuracy_train)
dt1_avg_accuracy_test=np.mean(dt1_accuracy_test)
dt1_avg_precision=np.mean(dt1_precision)
dt1_avg_recall=np.mean(dt1_recall)
dt2_avg_accuracy_train=np.mean(dt2_accuracy_train)
dt2_avg_accuracy_test=np.mean(dt2_accuracy_test)
dt2_avg_precision=np.mean(dt2_precision)
dt2_avg_recall=np.mean(dt2_recall)
print([dt1_accuracy_train,dt2_accuracy_train])
print([dt1_accuracy_test,dt2_accuracy_test])
print([dt1_avg_accuracy_train,dt1_avg_accuracy_test,dt1_avg_precision,dt1_avg_recall])
print([dt2_avg_accuracy_train,dt2_avg_accuracy_test,dt2_avg_precision,dt2_avg_recall])
```

```
[array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1.]), array([0.99497487, 0.98994975, 0.98743719, 0.9798995 , 0.99497487,
       1.        , 0.98994975, 0.99497487, 1.        , 0.94472362,
       0.98994975, 0.98994975, 0.99497487, 0.99497487, 0.98994975,
       0.94472362, 0.98994975, 0.99497487, 0.9798995 , 0.98994975])]
[array([0.9005848 , 0.92982456, 0.92397661, 0.92982456, 0.91812865,
       0.9122807 , 0.93567251, 0.91812865, 0.92397661, 0.92982456,
       0.91812865, 0.92397661, 0.90643275, 0.91812865, 0.9122807 ,
       0.93567251, 0.9122807 , 0.91812865, 0.92982456, 0.9122807 ]), array([0.9122807 , 0.92982456, 0.93567251, 0.93567251, 0.93567251,
       0.92397661, 0.93567251, 0.91812865, 0.92982456, 0.88304094,
       0.92397661, 0.92982456, 0.91812865, 0.92397661, 0.92982456,
       0.88304094, 0.92397661, 0.9122807 , 0.92397661, 0.92982456])]
[1.0, 0.92046783625731, 0.9053330039099803, 0.8796875]
[0.9868090452261307, 0.9219298245614036, 0.9039124400043848, 0.9]
```