KNN Classifier & Decision Tree

In [31]:
```
import pandas as pd
import numpy as np
from sklearn import tree
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_predict, cross_val_score
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.metrics import recall_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import balanced_accuracy_score
```

New Section

In [3]:
```
df_train= pd.read_csv("train.txt")
df_test= pd.read_csv("test.txt")
```

In [4]:
```
df_train = df_train.drop(['Sample code number'], axis=1)
df_set = df_set.drop(['Sample code number'], axis=1)
```

In [5]:
```
#remove rows with "?"
cpy= train_set.cpy()

rows=[]
for i in cpy.index:
  for j in cpy.columns:
    if cpy.iloc[i][j] == '?':
      rows.append(i)

cpynew= cpy.shape[0]+1 - len(rows)

cpy= cpy.drop(index= rows, axis=0)
```

In [6]:
```
new_df_train = cpy
new_df_train['Bare Nuclei'] = new_df_train['Bare Nuclei'].astype(int)
new_df_train
```

Out[6]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	5	1	1	1	2	1	3	1	1	2
1	5	4	4	5	7	10	3	2	1	2
2	3	1	1	1	2	2	3	1	1	2
3	6	8	8	1	3	4	3	7	1	2
4	4	1	1	3	2	1	3	1	1	2
...	...	...	...	...	...	...	...	...	...	...
639	1	1	3	1	2	1	2	1	1	2
640	3	1	1	3	2	1	2	1	1	2
641	1	1	1	1	2	1	1	1	1	2
642	5	2	2	2	2	1	1	1	2	2
643	3	1	1	1	2	1	3	1	1	2

644 rows × 10 columns

1.0 Apply the KNN classifier you made in problem 1 to predict the class labels of the testing data samples for the Wisconsin Breast Cancer Diagnostic Data Set. The training dataset is in file “breastcancer- wisconsin-train.txt”, the testing dataset is in the file “breast-cancer-wisconsin-test.txt”, and the data instruction is in the file “breast-cancer-wisconsin-instruction.txt”. For the KNN classifier,

In [7]:
```
train_data = new_train
test_data = test_set.drop(['Class'], axis=1)
```

In [8]:
```
x_train = new_train.drop(['Class'], axis=1)
y_train = new_train['Class']

x_test = test_set.drop(['Class'], axis=1)
y_test = test_set['Class']
```

In [9]:
```
# minkowski distance:

def minkowski_distance(a, b, r):
  lol= []
  for x, y in zip(a,b):
    lol.append(abs(x-y)**r)
  s= sum(lol)
  final = s**(1/r)
  return final
```

In [10]:
```
# find the mink distance between all the points:

def return_ranks(a):
  rank_df= pd.DataFrame(new_train['Class'])
  rank_df['Min distances'] = ''
  rank_df['Rank'] = ''
  liss= []
  for b in train_data.values:
    liss.append(minkowski_distance(a, b, 1))
  rank_df['Min distances'] = liss
  return rank_df
```

In [11]:
```
# testing the values on the test_data:

def KNN_algo(k, train_data, test_data):
  for vals in test_data.index:
  #create ranks or sort:
    final_rank_table_df = return_ranks(test_data.values[vals])
    final_rank_table_df['Rank'] = final_rank_table['Min distances'].rank(method='first', ascending =True)

    #the most repeated element among the k values and print:
    maj2=[]

    majority_df= list(final_rank_table[final_rank_table['Rank'] < (k+1)]['Class'])
    for pp in majority_df:
      maj2.append(majority.count(pp))

    m=max(maj2)
    for i in majority_df:
      if majority.count(i) ==m:
        cat= i
        break

    if i==2:
      print("The Class of this Data-point:",i,"Benign")
    elif i==4:
      print("The Class of this Data-point:",i,"Malignant")

KNN_algo(5, train_data, test_data)
```

```
The Class of this Data-point: 4 Malignant
The Class of this Data-point: 4 Malignant
The Class of this Data-point: 4 Malignant
The Class of this Data-point: 2 Benign
The Class of this Data-point: 2 Benign
The Class of this Data-point: 2 Benign
The Class of this Data-point: 2 Benign
The Class of this Data-point: 2 Benign
The Class of this Data-point: 2 Benign
The Class of this Data-point: 2 Benign
The Class of this Data-point: 2 Benign
The Class of this Data-point: 2 Benign
The Class of this Data-point: 4 Malignant
The Class of this Data-point: 4 Malignant
The Class of this Data-point: 2 Benign
The Class of this Data-point: 2 Benign
The Class of this Data-point: 2 Benign
The Class of this Data-point: 2 Benign
The Class of this Data-point: 2 Benign
The Class of this Data-point: 2 Benign
The Class of this Data-point: 2 Benign
The Class of this Data-point: 2 Benign
The Class of this Data-point: 2 Benign
The Class of this Data-point: 4 Malignant
The Class of this Data-point: 2 Benign
The Class of this Data-point: 2 Benign
The Class of this Data-point: 2 Benign
The Class of this Data-point: 2 Benign
The Class of this Data-point: 4 Malignant
The Class of this Data-point: 4 Malignant
The Class of this Data-point: 4 Malignant
The Class of this Data-point: 4 Malignant
The Class of this Data-point: 2 Benign
The Class of this Data-point: 2 Benign
The Class of this Data-point: 2 Benign
The Class of this Data-point: 4 Malignant
The Class of this Data-point: 2 Benign
The Class of this Data-point: 4 Malignant
The Class of this Data-point: 4 Malignant
```

1.1) For each KNN parameter setting of K and r, report classification accuracy, sensitivity, specificity, balanced accuracy, recall, precision, F1-score, and the confusion matrix.

In [12]:
```
model = KNeighborsClassifier(n_neighbors = 5, p = 1, metric = 'minkowski')
model.fit(x_train, y_train)

y_pred_knn = model.predict(x_test)
y_pred_knn
```

Out[12]:
```
array([4, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 4, 2, 2, 2, 2, 4, 4, 4, 4, 2, 2, 2, 4, 2, 4, 4])
```

In [13]:
```
def print_score(model, x_train, y_train, x_test, y_test, dataset):
    if dataset == 'Train':
        print ("Accuracy score: {0:.4f}\n".format(accuracy_score(y_train, model.predict(x_train))*100))
        print ("Classification Report:{}".format(classification_report(y_train, model.predict(x_train))))
        print ("Confusion Matrix: \n {}\n".format(confusion_matrix(y_train, model.predict(x_train))))

        res = cross_val_score(model, x_train, y_train, cv=10, scoring='accuracy')
        print("Average Accuracy: \t {0:.4f}".format(np.mean(res)))
        print("Accuracy SD: \t\t{0:.4f}".format(np.std(res)))

    elif dataset == 'Test':
        print ("Accuracy score: {0:.4f}\n".format(accuracy_score(y_test, model.predict(x_test))*100))
        print ("Classification Report: \n {}\n".format(classification_report(y_test, model.predict(x_test))))
        print ("Confusion Matrix: \n {}\n".format(confusion_matrix(y_test, model.predict(x_test))))

print("For Train Dataset:")
print_score(model, x_train, y_train, x_test, y_test, 'Train')
print()
print()
print("For Test Dataset:")
print_score(model, x_train, y_train, x_test, y_test, 'Test')

print(confusion_matrix(y_test , y_pred_knn))
print(classification_report(y_test ,y_pred_knn,target_names=["Benign","Malignant"]))
conf_mat=confusion_matrix(y_test ,y_pred_knn)
total1=sum(sum(conf_mat))
#from confusion matrix calculate accuracy
sensitivity1=conf_mat[0,0]/(conf_mat[0,0]+conf_mat[0,1])
print('Sensitivity : ',sensitivity1)
specificity1=conf_mat[1,1]/(conf_mat[1,0]+conf_mat[1,1])
print('Specificity : ',specificity1)
print("")
print("The balanced accuracy is :")
print(balanced_accuracy_score(y_test, y_pred_knn))
```

```
For Train Dataset:
Accuracy score: 97.5155

Classification Report:              precision    recall  f1-score   support

           2       0.98      0.98      0.98       419
           4       0.96      0.96      0.96       225

    accuracy                           0.98       644
   macro avg       0.97      0.97      0.97       644
weighted avg       0.98      0.98      0.98       644

Confusion Matrix:
 [[411   8]
 [  8 217]]

Average Accuracy:        0.9644
Accuracy SD:            0.0364


For Test Dataset:
Accuracy score: 97.4359

Classification Report:
               precision    recall  f1-score   support

           2       0.96      1.00      0.98        25
           4       1.00      0.93      0.96        14

    accuracy                           0.97        39
   macro avg       0.98      0.96      0.97        39
weighted avg       0.98      0.97      0.97        39


Confusion Matrix:
 [[25  0]
 [ 1 13]]

[[25  0]
 [ 1 13]]
              precision    recall  f1-score   support

      Benign       0.96      1.00      0.98        25
   Malignant       1.00      0.93      0.96        14

    accuracy                           0.97        39
   macro avg       0.98      0.96      0.97        39
weighted avg       0.98      0.97      0.97        39

Sensitivity :  1.0
Specificity :  0.9285714285714286

The balanced accuracy is :
0.9642857142857143
```

In [14]:
```
# 2nd model with k=5 and r=2

model_df = KNeighborsClassifier(n_neighbors = 5, p = 2, metric = 'minkowski')
model_df.fit(x_train, y_train)
```

Out[14]:
```
KNeighborsClassifier()
```

In [15]:
```
y_pred_Knn2= model2.predict(x_test)
y_pred_Knn2
```

Out[15]:
```
array([4, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 4, 4, 4, 4, 2, 2, 2, 4, 2, 4, 4])
```

In [16]:
```
print("Train set")
print("Accuracy score: {0:.4f}".format(accuracy_score(y_train, model2.predict(x_train))*100),"%\n")
print("Classification Report:{}".format(classification_report(y_train, model2.predict(x_train))))
print("Confusion Matrix: \n {}\n".format(confusion_matrix(y_train, model2.predict(x_train))))

res_df = cross_val_score(model, x_train, y_train, cv=10, scoring='accuracy')
print("Average Accuracy: \t {0:.4f}".format(np.mean(res_df)))
print("Accuracy SD: \t\t{0:.4f}".format(np.std(res_df)))
print()
print()
print("Test set")
print("Accuracy score: {0:.4f}".format(accuracy_score(y_test, model2.predict(x_test))*100),"%\n")
print("Classification Report: \n {}\n".format(classification_report(y_test, model2.predict(x_test))))
print("Confusion Matrix: \n {}\n".format(confusion_matrix(y_test, model2.predict(x_test))))

print(confusion_matrix(y_test , y_pred_Knn2))
print(classification_report(y_test ,y_pred_Knn2,target_names=["Benign","Malignant"]))
conf_mat=confusion_matrix(y_test ,y_pred_Knn2)
total1=sum(sum(conf_mat))
#from confusion matrix calculate accuracy
sensitivity1=conf_mat[0,0]/(conf_mat[0,0]+conf_mat[0,1])
print('Sensitivity : ',sensitivity1)
specificity1=conf_mat[1,1]/(conf_mat[1,0]+conf_mat[1,1])
print('Specificity : ',specificity1)
print("")
print("The balanced accuracy score is :")
print(balanced_accuracy_score(y_test, y_pred_knn2))
```

```
Train set
Accuracy score: 97.9814 %

Classification Report:              precision    recall  f1-score   support

           2       0.99      0.98      0.98       419
           4       0.96      0.98      0.97       225

    accuracy                           0.98       644
   macro avg       0.98      0.98      0.98       644
weighted avg       0.98      0.98      0.98       644

Confusion Matrix:
 [[410   9]
 [  4 221]]

Average Accuracy:        0.9644
Accuracy SD:            0.0364


Test set
Accuracy score: 94.8718 %

Classification Report:
               precision    recall  f1-score   support

           2       0.93      1.00      0.96        25
           4       1.00      0.86      0.92        14

    accuracy                           0.95        39
   macro avg       0.96      0.93      0.94        39
weighted avg       0.95      0.95      0.95        39


Confusion Matrix:
 [[25  0]
 [ 2 12]]

[[25  0]
 [ 2 12]]
              precision    recall  f1-score   support

      Benign       0.93      1.00      0.96        25
   Malignant       1.00      0.86      0.92        14

    accuracy                           0.95        39
   macro avg       0.96      0.93      0.94        39
weighted avg       0.95      0.95      0.95        39

Sensitivity :  1.0
Specificity :  0.8571428571428571

The balanced accuracy score is :
0.9285714285714286
```

1.2) Based on the balanced accuracy, which KNN parameter setting is better? Model (k=5,r=1) is preferred over model 2(k=5,r=2) as the accuracy of model is better than that of model 2.

1. Build a decision tree classifier using the Breast Cancer training dataset and evaluate the trained decision tree on the testing dataset. Try different settings of the decision tree model and report the classification performance measures. The parameter options include: 1) feature ranking criterion: “gini”, “entropy”; 2) The maximum number of features to select at random at each split: $\sqrt{n}$, $\log n$, and n, where n is the number of features. For example, there are 50 features and the maximum feature used at each split is 10. Then each time, it randomly selects 10 features and pick the best of the 10 to split. For the next node split, it will randomly select another 10 and so on. As a thumb-rule, $\sqrt{n}$ and $\log n$ could work well. This design is to avoid overfitting. Higher values may lead to overfitting. You can verify in this problem.

2.1) For each parameter setting, report classification accuracy, sensitivity, specificity, and balanced accuracy.

In [30]:
```
#GINI IMPURITY 1
tree_gin_df1=DecisionTreeClassifier(criterion = 'gini',splitter='random',max_features="auto",min_samples_split=10)
tree_gin_df1.fit(x_train,y_train)
y_pred_train_gin1_d1=tree_gin_df1.predict(x_train)
y_pred_test_gin_d1=tree_gin_df1.predict(x_test)

print("\tGINI 1:");
print("Train Set Accuracy:",accuracy_score(y_train,y_pred_train_gin1_d1))
print("Test Set Accuracy:",accuracy_score(y_test,y_pred_test_gin_d1))
print("Balanced accuracy:",balanced_accuracy_score(y_test, y_pred_test_gin_d1))
```

```
        GINI 1:
Train Set Accuracy: 0.9611801242236024
Test Set Accuracy: 0.9487179487179487
Balanced accuracy: 0.9285714285714286
```

In [22]:
```
#GINI IMPURITY 2
tree_gin_d2=DecisionTreeClassifier(criterion = 'gini',splitter='random',max_features="sqrt",min_samples_split=10)
tree_gin_d2.fit(x_train,y_train)
y_pred_train_gin1_d2=tree_gin_d2.predict(x_train)
y_pred_test_gin_d2=tree_gin_d2.predict(x_test)

print("\tGINI 2:");
print("Train Set Accuracy:",accuracy_score(y_train,y_pred_train_gin1_d2))
print("Test Set Accuracy:",accuracy_score(y_test,y_pred_test_gin_d2))
print("Balanced accuracy:",balanced_accuracy_score(y_test, y_pred_test_gin_d2))
```

```
        GINI 2:
Train Set Accuracy: 0.9767080745341615
Test Set Accuracy: 0.9230769230769231
Balanced accuracy: 0.8928571428571428
```

In [23]:
```
#GINI IMPURITY 3
tree_gin_d3= DecisionTreeClassifier(criterion = 'gini',splitter='random',max_features="log2",min_samples_split=10)
tree_gin_d3.fit(x_train,y_train)

y_pred_train_gin1_d3= tree_gin_d3.predict(x_train)
y_pred_test_gin_d3= tree_gin_d3.predict(x_test)


print("\tGINI 3:");
print("Train Set Accuracy:",accuracy_score(y_train,y_pred_train_gin1_d3))
print("Test Set Accuracy:",accuracy_score(y_test,y_pred_test_gin_d3))
print("Balanced accuracy:",balanced_accuracy_score(y_test, y_pred_test_gin_d3))
```

```
        GINI 3:
Train Set Accuracy: 0.967391304347826
Test Set Accuracy: 0.9230769230769231
Balanced accuracy: 0.8928571428571428
```

In [24]:
```
# The Entropy Impurity for 1

tree_ent_d1=DecisionTreeClassifier(criterion='entropy',splitter='random',max_features="auto",min_samples_split=10)
tree_ent_d1.fit(x_train,y_train)

tree_ent_d1.fit(x_train,y_train)
y_pred_train_ent_d1=tree_ent_d1.predict(x_train);
y_pred_test_ent_d1=tree_ent_d1.predict(x_test);


print("\tENTROPY 1:");
print("Train Set Accuracy:",accuracy_score(y_train,y_pred_train_ent_d1))
print("Test Set Accuracy:",accuracy_score(y_test,y_pred_test_ent_d1))
print("Balanced accuracy:",balanced_accuracy_score(y_test, y_pred_test_ent_d1))
```

```
        ENTROPY 1:
Train Set Accuracy: 0.9658385093167702
Test Set Accuracy: 0.9743589743589743
Balanced accuracy: 0.9642857142857143
```

In [26]:
```
# The Entropy Impurity for 2

tree_ent_d2=DecisionTreeClassifier(criterion='entropy',splitter='random',max_features="sqrt",min_samples_split=10)
tree_ent_d2.fit(x_train,y_train)

tree_ent_d2.fit(x_train,y_train)
y_pred_train_ent_d2=tree_ent_d2.predict(x_train);
y_pred_test_ent_d2=tree_ent_d2.predict(x_test);


print("\tENTROPY 2:");
print("Train Set Accuracy:",accuracy_score(y_train,y_pred_train_ent_d2))
print("Test Set Accuracy:",accuracy_score(y_test,y_pred_test_ent_d2))
print("Balanced accuracy:",balanced_accuracy_score(y_test, y_pred_test_ent_d2))
```

```
        ENTROPY 2:
Train Set Accuracy: 0.9627329192546584
Test Set Accuracy: 0.9487179487179487
Balanced accuracy: 0.9285714285714286
```

In [28]:
```
# the Entropy Impurity for 3

tree_ent_d3=DecisionTreeClassifier(criterion='entropy',splitter='random',max_features="log2",min_samples_split=10)
tree_ent_d3.fit(x_train,y_train)

tree_ent_d3.fit(x_train,y_train)
y_pred_train_ent_d3=tree_ent_d3.predict(x_train);
y_pred_test_ent_d3=tree_ent_d3.predict(x_test);


print("\tENTROPY 3:");
print("Train Set Accuracy:",accuracy_score(y_train,y_pred_train_ent_d3))
print("Test Set Accuracy:",accuracy_score(y_test,y_pred_test_ent_d3))
print("Balanced accuracy:",balanced_accuracy_score(y_test, y_pred_test_ent_d3))
```

```
        ENTROPY 3:
Train Set Accuracy: 0.9751552795031055
Test Set Accuracy: 0.9487179487179487
Balanced accuracy: 0.9285714285714286
```

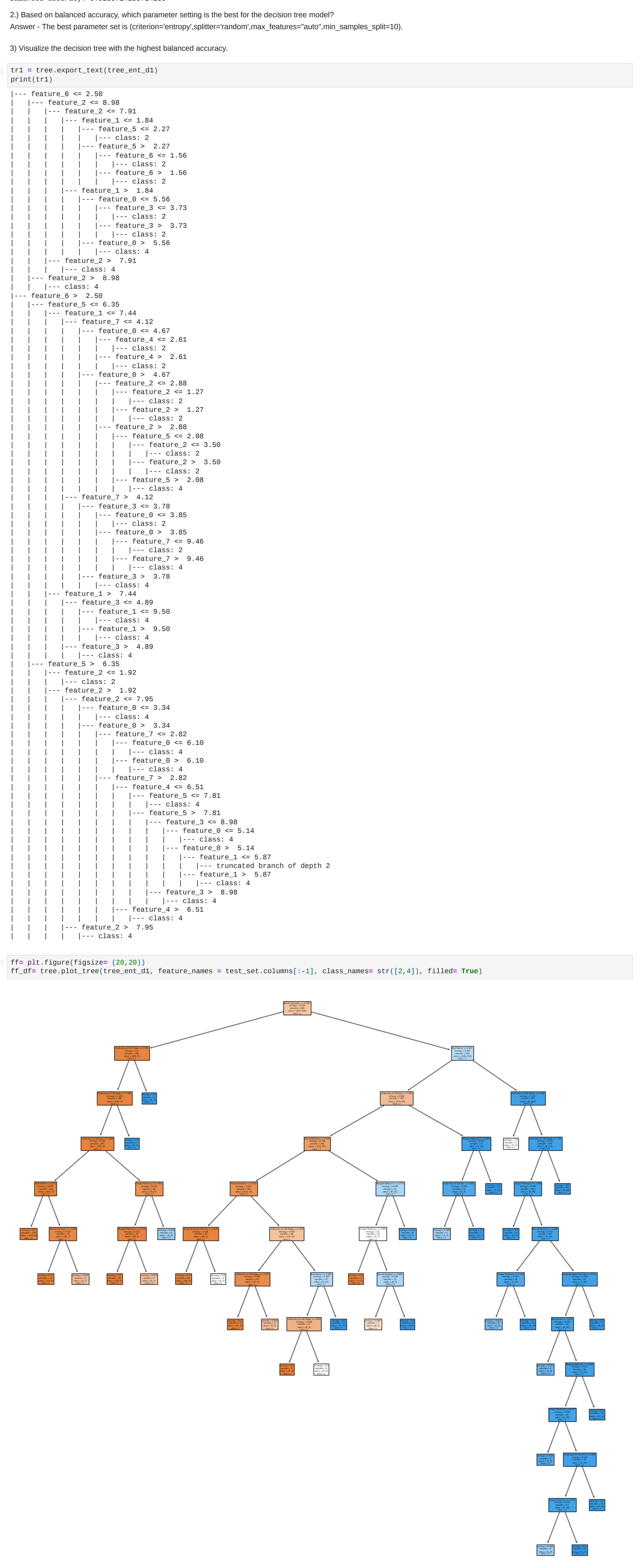
2.) Based on balanced accuracy, which parameter setting is the best for the decision tree model?
Answer - The best parameter set is (criterion='entropy',splitter='random',max_features="auto",min_samples_split=10).

3) Visualize the decision tree with the highest balanced accuracy.

In [33]:
```
tr1 = tree.export_text(tree_ent_d1)
print(tr1)
```

```
|--- feature_6 <= 2.50
|   |--- feature_2 <= 8.98
|   |   |--- feature_2 <= 7.91
|   |   |   |--- feature_1 <= 1.84
|   |   |   |   |--- feature_5 <= 2.27
|   |   |   |   |   |--- class: 2
|   |   |   |   |--- feature_5 >  2.27
|   |   |   |   |   |--- feature_6 <= 1.56
|   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |--- feature_6 >  1.56
|   |   |   |   |   |   |--- class: 2
|   |   |   |--- feature_1 >  1.84
|   |   |   |   |--- feature_0 <= 5.56
|   |   |   |   |   |--- feature_3 <= 3.73
|   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |--- feature_3 >  3.73
|   |   |   |   |   |   |--- class: 2
|   |   |   |   |--- feature_0 >  5.56
|   |   |   |   |   |--- class: 4
|   |   |--- feature_2 >  7.91
|   |   |   |--- class: 4
|   |--- feature_2 >  8.98
|   |   |--- class: 4
|--- feature_6 >  2.50
|   |--- feature_5 <= 6.35
|   |   |--- feature_1 <= 7.44
|   |   |   |--- feature_7 <= 4.12
|   |   |   |   |--- feature_0 <= 4.67
|   |   |   |   |   |--- feature_4 <= 2.61
|   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |--- feature_4 >  2.61
|   |   |   |   |   |   |--- class: 2
|   |   |   |   |--- feature_0 >  4.67
|   |   |   |   |   |--- feature_2 <= 2.88
|   |   |   |   |   |   |--- feature_2 <= 1.27
|   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |--- feature_2 >  1.27
|   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |--- feature_2 >  2.88
|   |   |   |   |   |   |--- feature_5 <= 2.08
|   |   |   |   |   |   |   |--- feature_2 <= 3.50
|   |   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |   |--- feature_2 >  3.50
|   |   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |--- feature_5 >  2.08
|   |   |   |   |   |   |   |--- class: 4
|   |   |   |--- feature_7 >  4.12
|   |   |   |   |--- feature_3 <= 3.78
|   |   |   |   |   |--- feature_0 <= 3.85
|   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |--- feature_0 >  3.85
|   |   |   |   |   |   |--- feature_7 <= 9.46
|   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |--- feature_7 >  9.46
|   |   |   |   |   |   |   |--- class: 4
|   |   |   |   |--- feature_3 >  3.78
|   |   |   |   |   |--- class: 4
|   |   |--- feature_1 >  7.44
|   |   |   |--- feature_3 <= 4.89
|   |   |   |   |--- feature_1 <= 9.50
|   |   |   |   |   |--- class: 4
|   |   |   |   |--- feature_1 >  9.50
|   |   |   |   |   |--- class: 4
|   |   |   |--- feature_3 >  4.89
|   |   |   |   |--- class: 4
|   |--- feature_5 >  6.35
|   |   |--- feature_2 <= 1.92
|   |   |   |--- class: 2
|   |   |--- feature_2 >  1.92
|   |   |   |--- feature_2 <= 7.95
|   |   |   |   |--- feature_0 <= 3.34
|   |   |   |   |   |--- class: 4
|   |   |   |   |--- feature_0 >  3.34
|   |   |   |   |   |--- feature_7 <= 2.82
|   |   |   |   |   |   |--- feature_0 <= 6.10
|   |   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |   |--- feature_0 >  6.10
|   |   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |--- feature_7 >  2.82
|   |   |   |   |   |   |--- feature_4 <= 6.51
|   |   |   |   |   |   |   |--- feature_5 <= 7.81
|   |   |   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |   |   |--- feature_5 >  7.81
|   |   |   |   |   |   |   |   |--- feature_3 <= 8.98
|   |   |   |   |   |   |   |   |   |--- feature_0 <= 5.14
|   |   |   |   |   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |   |   |   |   |--- feature_0 >  5.14
|   |   |   |   |   |   |   |   |   |   |--- feature_1 <= 5.87
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |   |--- feature_1 >  5.87
|   |   |   |   |   |   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |   |   |   |--- feature_3 >  8.98
|   |   |   |   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |   |--- feature_4 >  6.51
|   |   |   |   |   |   |   |--- class: 4
|   |   |   |--- feature_2 >  7.95
|   |   |   |   |--- class: 4
```

In [34]:
```
ff= plt.figure(figsize= (20,20))
ff_df= tree.plot_tree(tree_ent_d1, feature_names = test_set.columns[:-1], class_names= str([2,4]), filled= True)
```



In [ ]: