Classification

K-NN(K-Nearest Neighbors)

In [14]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
df=pd.read_csv("Datasets/Churn.csv")
df=df.dropna()
# df.info()
x=df.drop(["Unnamed: 0","customerID","Churn"],axis=1)
y=df['Churn']
x=pd.get_dummies(x)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=2
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train,y_train)
ypred=knn.predict(x_test)
print(ypred)
```

```
['Yes' 'No' 'No' ... 'No' 'Yes' 'No']
```

Overfitting->

- it accuers when a model learns the training data too well including noice and irrelevant details resulting in poor performance on new data
- high accuracey on traing data and poor accurecy on new and unseen data

Underfiting

- it happens when a model is to simple to capture the underline patterns in the training data resulting in poor performance on both traing as well as new data

KNN(K-Nearest Neighbors)

- based on its closenes or similarities in a given range(k) of neighbors the algorithm assigns the new data to a class or category in the dataset
- to calculate the closenes __ formula is used
- key point->always use odd number for k

Confusion Matrix (201%)

In [4]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
import pandas as pd
df=pd.read_csv("Datasets/Churn.csv")
df=df.dropna()
x=df.drop(["Unnamed: 0","customerID","Churn"],axis=1)
y=df['Churn']
x=pd.get_dummies(x)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=2
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train,y_train)
ypred=knn.predict(x_test)
conf_mat=confusion_matrix(y_test,ypred)
print(conf_mat)
```

```
[[909 124]
 [207 167]]
```

- **Confusion matrix is simple table that show the how well the calssification model by compraing its prediction to the actual result**
- its break down predction into 4 categorey
- 1.TP(True positive) ->correctly predicted possive outcome
- 2.TN(true negative) ->correctly predicted negative outcome
- 3.FP(false postive) ->incorrectly predicted a postive outcome(the actual outcome was negative) it is also kown as type-1 error
- 4.FN(False negative) ->incorrectly predicted a negative outcome(the actual outcome was postive) it is also kown as type-2 error

1.Acuracy ->it measure overall performance of the model

Acuracy=(TP+TN)/(TP+TN+FP+FN)

2.Precisoin ->it focus is on the quality of the models postive predictions(capicity of a model to correctly predict possitve instances

Precision=TP/(TP+FP)

3.Recall ->It show the propotion of true posstive detacted out of all the actual postive instances

recal = TP/(TP+FN)

4.specificity -> measures the ability of a model to correctly

identify negative instances

In [16]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
import pandas as pd
df=pd.read_csv("Datasets/Churn.csv")
df=df.dropna()
x=df.drop(["Unnamed: 0","customerID","Churn"],axis=1)
y=df['Churn']
x=pd.get_dummies(x)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.002,random_state
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train,y_train)
ypred=knn.predict(x_test)
print(ypred)
conf_mat=confusion_matrix(y_test,ypred)
print(conf_mat)

tn=conf_mat[0][0]
fp=conf_mat[0][1]
fn=conf_mat[1][0]
tp=conf_mat[1][1]
print("accuracy=",(tp+tn)/(tp+tn+fp+fn))
print("Precision=",(tp)/(tp+fp))
print("recall=",(tp)/(tp+fn))
print("specificity=",(tn)/(tn+fp))
```

```
['Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes'
 'No']
[[9 3]
 [2 1]]
accuracy= 0.6666666666666666
Precision= 0.25
recall= 0.3333333333333333
specificity= 0.75
```

In [24]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix,accuracy_score,precision_score,rec
from sklearn.model_selection import train_test_split
import pandas as pd
df=pd.read_csv("Datasets/Churn.csv")
df=df.dropna()
x=df.drop(["Unnamed: 0","customerID","Churn"],axis=1)
y=df['Churn']
x=pd.get_dummies(x)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.002,random_state
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train,y_train)
ypred=knn.predict(x_test)
print(ypred)
conf_mat=confusion_matrix(y_test,ypred)
print(conf_mat)
print("accuracy=",accuracy_score(y_test,ypred))
print("Precision=",precision_score(y_test,ypred,pos_label='Yes'))
print("recall=",recall_score(y_test,ypred,pos_label="Yes"))
print("specificity=",recall_score(y_test,ypred,pos_label="No"))
```

```
['Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes'
 'No']
[[9 3]
 [2 1]]
accuracy= 0.6666666666666666
Precision= 0.25
recall= 0.3333333333333333
specificity= 0.75
```

In [40]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix,accuracy_score,precision_score,rec
from sklearn.model_selection import train_test_split
import pandas as pd
df=pd.read_csv("Datasets\diabetes_unclean.csv")
df= df.dropna()
x=df.drop(["CLASS","ID","No_Pation"],axis=1)
# x = pd.get_dummies(df.drop(["CLASS", "ID", "No_Pation"], axis=1))
# print(x.dtypes)
y=df["CLASS"]
x=pd.get_dummies(x)
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.2,random_state=110
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(xtrain,ytrain)
ypred=knn.predict(xtest)
# print(ypred)
conf_mat=confusion_matrix(ytest,ypred)
print(conf_mat)
print("accuracy=",accuracy_score(ytest,ypred))
print("Precision=",precision_score(ytest,ypred,average='macro'))
print("recall=",recall_score(ytest,ypred,average="macro"))
print("specificity=",recall_score(ytest,ypred,average="macro"))
```

```
[[ 15   0   8   0]
 [  3   6   2   0]
 [  8   2 154   0]
 [  0   0   1   0]]
accuracy= 0.8793969849246231
Precision= 0.5650641025641026
recall= 0.5341632121854816
specificity= 0.5341632121854816

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:122
1: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [43]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix,accuracy_score,precision_score,rec
from sklearn.model_selection import train_test_split
import pandas as pd
df=pd.read_csv("Datasets\diabetes_unclean.csv")
df= df.dropna()
x=df.drop(["CLASS","ID","No_Pation"],axis=1)
y=df["CLASS"]
maxacc=0;
x=pd.get_dummies(x)
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.2,random_state=110
for i in range(1,11):
    knn=KNeighborsClassifier(n_neighbors=i)
    knn.fit(xtrain,ytrain)
    ypred=knn.predict(xtest)
    conf_mat=confusion_matrix(ytest,ypred)
    acc=accuracy_score(ytest,ypred)
    if acc>maxacc:
        maxacc=acc
        index=i
print(f"max acc {acc} for {index}")
```

```
max acc 0.8542713567839196 for 1
```

Decision Tree

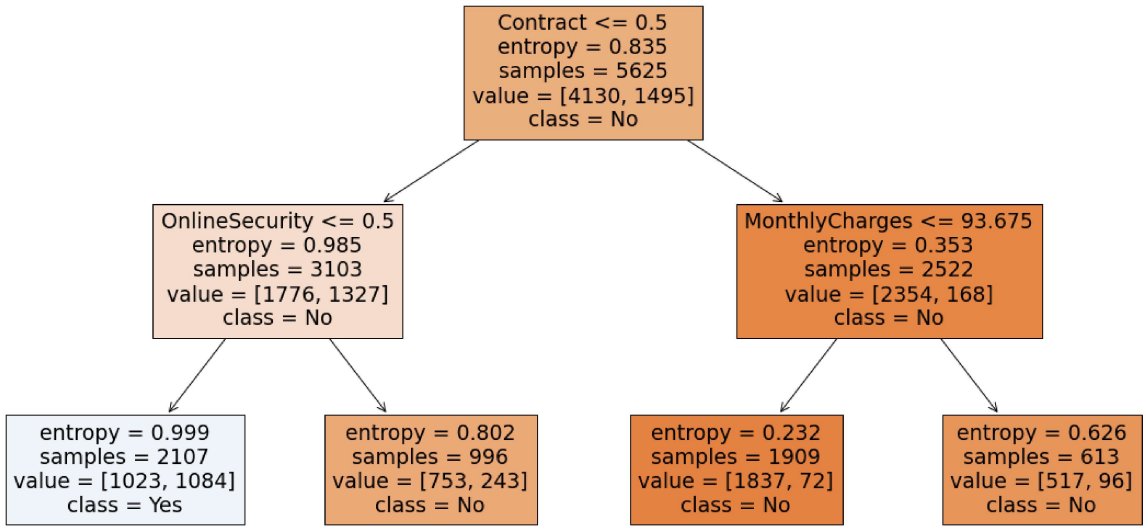
In [15]:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,
from sklearn.preprocessing import LabelEncoder

# Load dataset
df = pd.read_csv("csv/Churn.csv")

df.dropna(inplace=True)
# Drop unwanted columns
df.drop(["Unnamed: 0", "customerID"], axis=1, inplace=True)

# Convert target variable to numeric
df["Churn"] = df["Churn"].replace({"No": 0, "Yes": 1})

# Handle categorical variables
for column in df.columns:
    if df[column].dtype == 'object':
        df[column] = LabelEncoder().fit_transform(df[column])

# Split features and target
x = df.drop("Churn", axis=1)
y = df["Churn"]

# Split into train and test
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.2, random_sta

# Train Decision Tree
dt = DecisionTreeClassifier(criterion="entropy", max_depth=2)
dt.fit(xtrain, ytrain)

# Predict and evaluate
ypr = dt.predict(xtest)
print("Predictions:", ypr)
print("Accuracy:", accuracy_score(ytest, ypr))
print("Precision:", precision_score(ytest, ypr))
print("Recall:", recall_score(ytest, ypr))
print("Confusion Matrix:\n", confusion_matrix(ytest, ypr))

c = classification_report(ytest, ypr)
print(c)
```

```
Predictions: [1 0 0 ... 0 1 1]
Accuracy: 0.7299218194740583
Precision: 0.49427480916030536
Recall: 0.6925133689839572
Confusion Matrix:
 [[768 265]
 [115 259]]
              precision    recall  f1-score   support

           0       0.87      0.74      0.80      1033
           1       0.49      0.69      0.58       374

    accuracy                           0.73      1407
   macro avg       0.68      0.72      0.69      1407
weighted avg       0.77      0.73      0.74      1407
```

In [13]:

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(20,10))
plot_tree(dt, feature_names=x.columns, class_names=["No", "Yes"], filled=True)
plt.show()
```



P.b. 253 Encode Categorical variables using one hot encoding

- **pd.get_dummies(x)**
- **sensitivity = recall those are synonms**

In [ ]:

```

```

In [ ]:
```
1
```