CSPE65: MACHINE LEARNING TECHNIQUES AND PRACTICES

Assignment - 2

106119100 - Rajneesh Pandey

Problem Statements

1. Do the following for the given dataset:

   a. Visualize the dataset using a plot *(Library: Seaborn and Matplotlib)* and create Decision Tree Algorithm to its complete depth (visualize the tree after construction).

      - Calculate the following evaluation metrics → *Accuracy, Precision, Recall, F1 Score, Confusion Matrix* and discuss what you observe.

   b. Create Decision Tree Algorithm with hyperparameter tuning (visualize the tree after construction).

      - Calculate the following evaluation metrics → *Accuracy, Precision, Recall, F1 Score, Confusion Matrix* and discuss what you observe.

2. Do the following for the given dataset:

   a. Apply k-Nearest Neighbour algorithm on the given dataset *(Find the best value for "k" using the method explained in class)*.

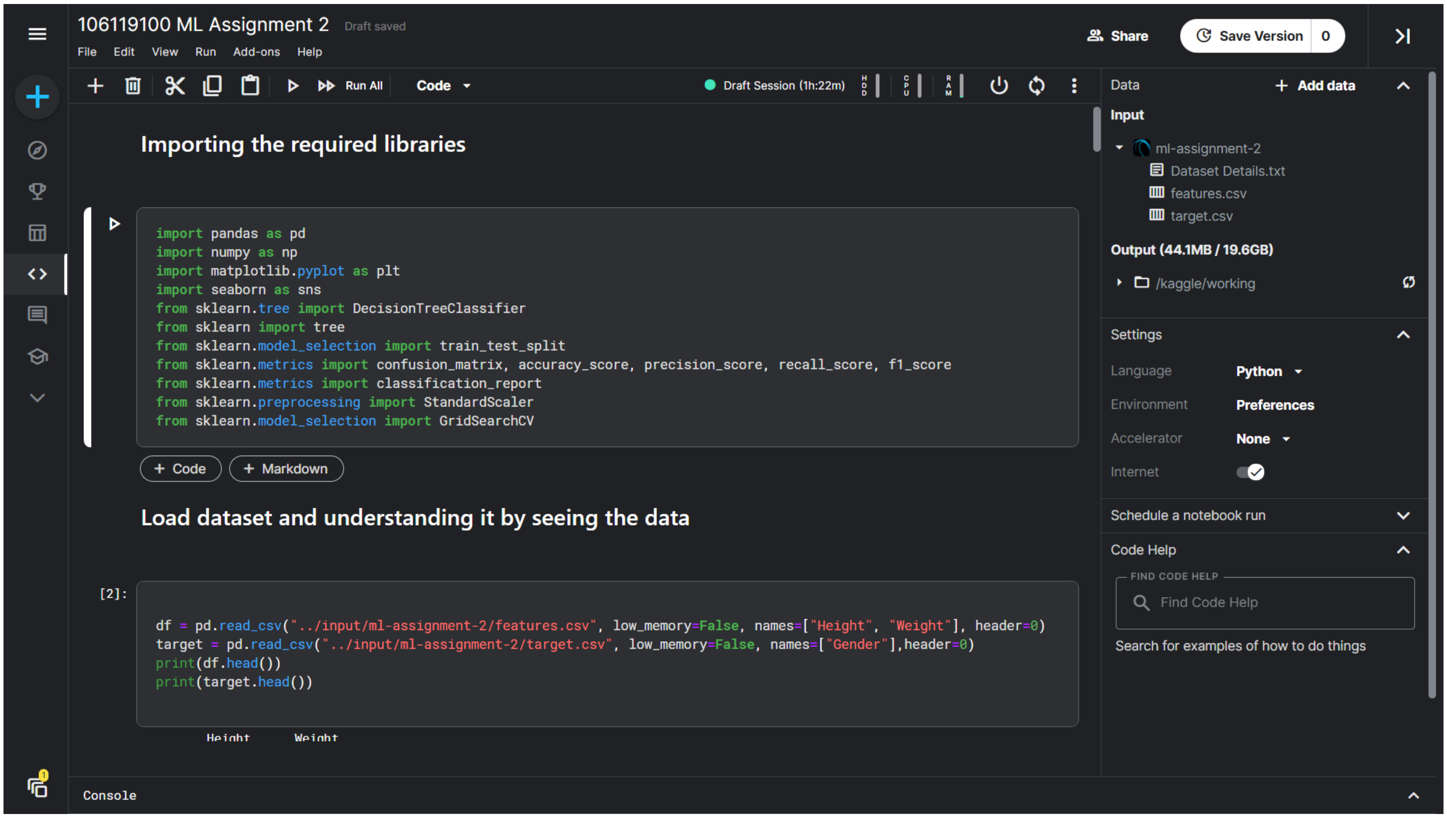
      - Calculate the following evaluation metrics → *Accuracy, Precision, Recall, F1 Score, Confusion Matrix* and discuss what you observe.

   b. Apply Min-Max Normalization on the given dataset and visualize it using a plot; Then, repeat Section "2. a" fully on the Normalized dataset.

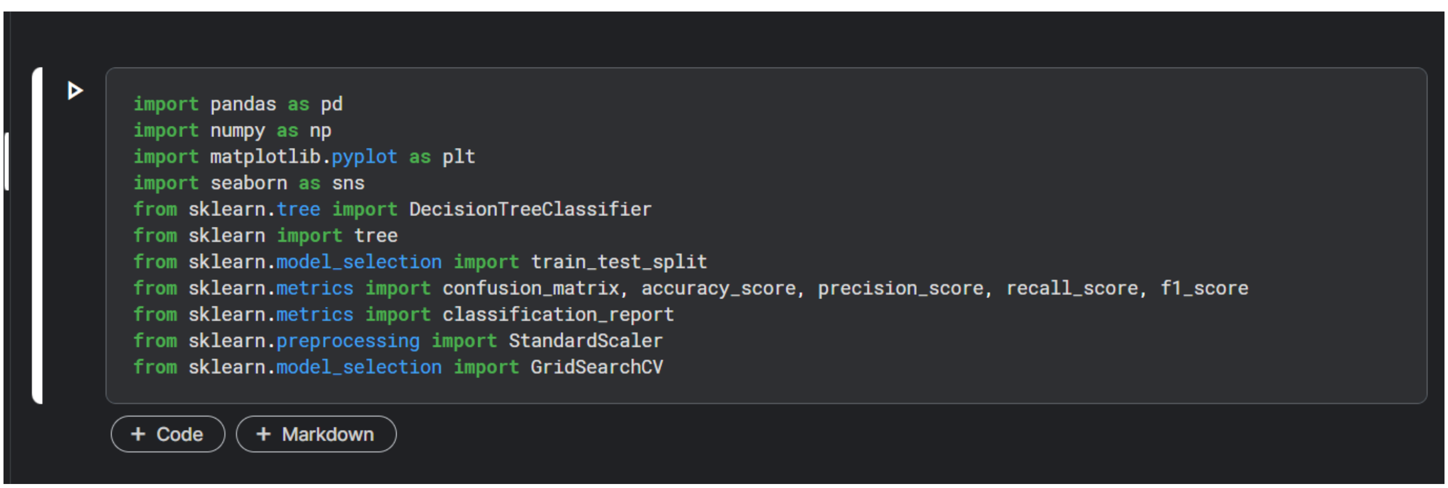
   c. Plot ROC curves and calculate the corresponding AUC values for Section "2. a" and "2. b" and discuss what you observe.

Code is written on Kaggle Notebook

Link of the notebook : https://www.kaggle.com/rajneesh1708/106119100-ml-assignment-2/edit



Importing the required libraries



Load dataset and understanding it by seeing the data

Load dataset and understanding it by seeing the data

```
df = pd.read_csv("../input/ml-assignment-2/features.csv", low_memory=False, names=["Height", "Weight"], header=0)
target = pd.read_csv("../input/ml-assignment-2/target.csv", low_memory=False, names=["Gender"],header=0)
print(df.head())
print(target.head())
```

```
         Height       Weight
0  1.170866e+06  114.834354
1  1.326528e+06   52.048800
2  1.436353e+06   56.271637
3  1.667446e+06  112.761949
4  1.606816e+06   74.433175
   Gender
0  Female
1  Female
2  Female
3    Male
4    Male
```

+ Code + Markdown

Checking for rows duplicate

Checking for rows duplicacy

```
[3]: any(df.duplicated())
```

```
[3]: False
```

Mapping the target variables, for Male as 1 and Female And checking the replacement done.

```
mapping_dict = {"Gender": {"Male": 1, "Female": 0}}
target.replace(mapping_dict, inplace=True)
print(target.dtypes)
target.head()
```

```
Gender    int64
dtype: object
```

[4]:

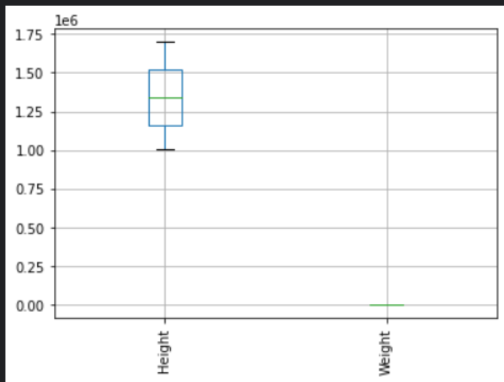
	Gender
0	0
1	0
2	0
3	1
4	1

+ Code + Markdown

Checking the outliers present in the dataset by using Box and Whiskers Plot

```
df.boxplot(rot=90)
```

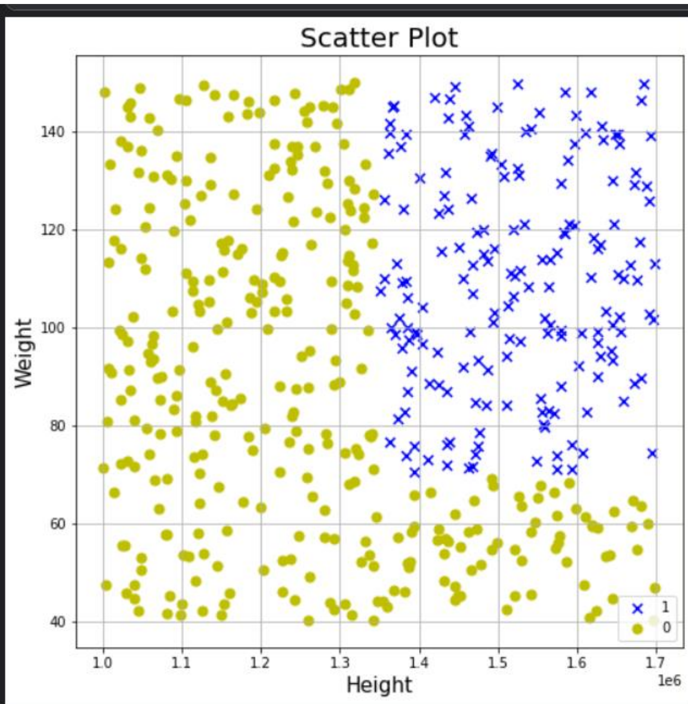
[5]: <AxesSubplot:>



+ Code + Markdown

Scatter Plot

```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Height', fontsize = 15)
ax.set_ylabel('Weight', fontsize = 15)
ax.set_title('Scatter Plot', fontsize = 20)
y_values = [1, 0]
colors = ['b', 'y']
markers = ['x', 'o']
for t, color, marker in zip(y_values,colors,markers):
    indicesToKeep = target['Gender'] == t
    ax.scatter(df.loc[indicesToKeep, 'Height'], df.loc[indicesToKeep, 'Weight'], c = color, s=50, marker=marker)
ax.legend(y_values)
ax.grid()
```



+ Code + Markdown

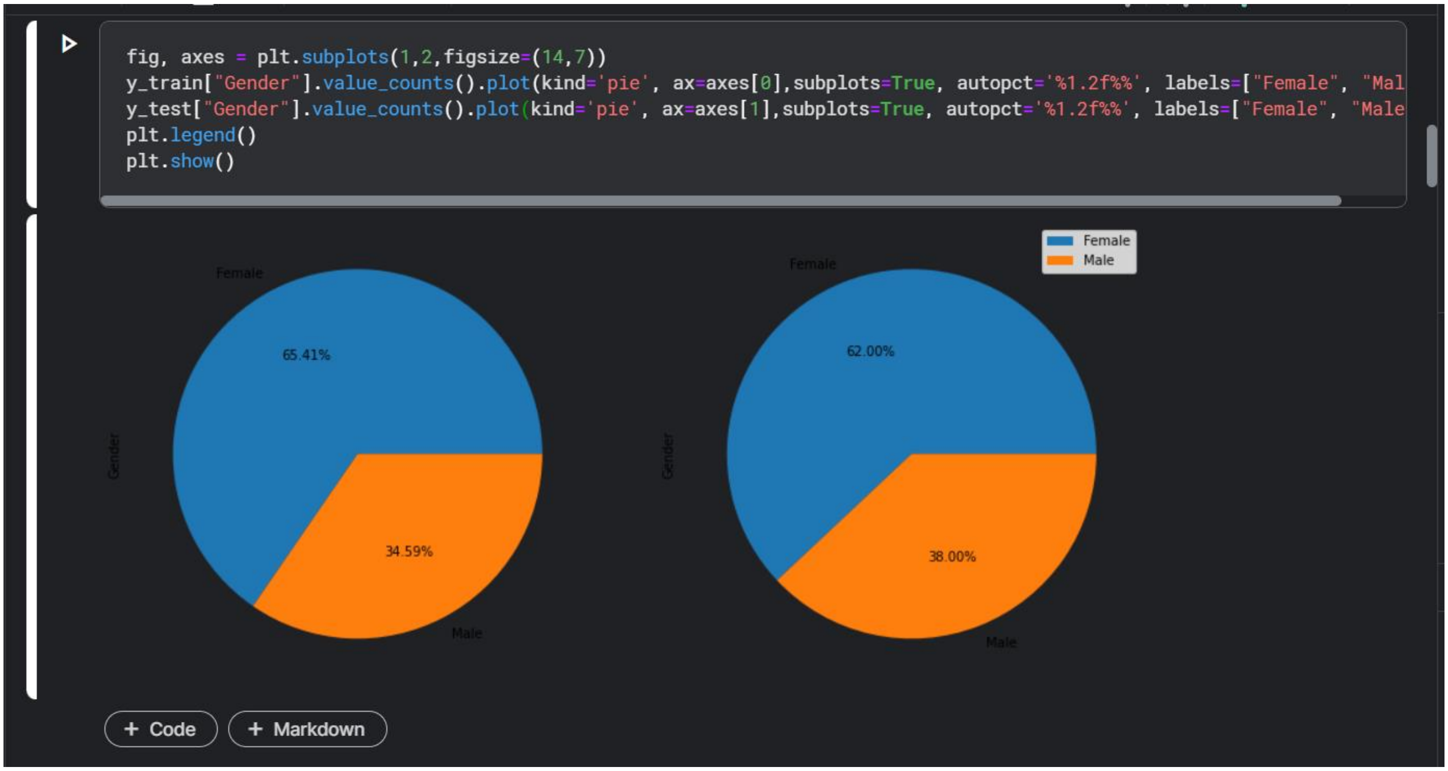
Split our dataset into training and testing dataset in the ratio of 80-20 respectively.

```
x_train, x_test, y_train, y_test = train_test_split(df, target, test_size=0.2, random_state=1)
print(len(x_train), len(x_test), len(y_train), len(y_test))
print(x_test.head(), y_test.head())
```

```
399 100 399 100
           Height      Weight
110  1.103653e+06  125.121096
147  1.521583e+06   45.075623
307  1.030528e+06   45.573982
326  1.427560e+06  115.348483
189  1.237990e+06   52.562252      Gender
110       0
147       0
307       0
326       1
189       0
```

+ Code + Markdown

Check for class imbalance i.e. we have data belonging to both the classes in training and testing dataset



As I can see from the plots, both the classes are split in appropriate proportions. So our model will not become biased after training.

Question 1A and 1B

Intialising the DecisionTreeClassifier

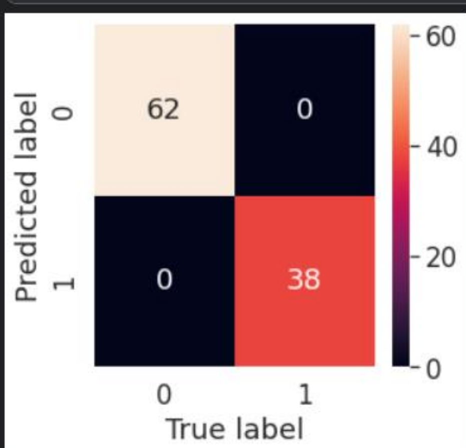
Question 1 A

+ Code + Markdown

```
[9]: model = DecisionTreeClassifier()
     fittedModel = model.fit(x_train, y_train)
     fittedModel
```

```
[9]: DecisionTreeClassifier()
```

```
predict = fittedModel.predict(x_test)
sns.set(font_scale=1.5)
fig, ax = plt.subplots(figsize=(4, 4))
ax = sns.heatmap(confusion_matrix(y_test, predict), annot=True, cbar=True)
plt.xlabel("True label")
plt.ylabel("Predicted label")
plt.show()
```



+ Code + Markdown

Printing the required evaluation metrics (1A)

```
[11]:
print("Confusion Matrix : \n",confusion_matrix(y_test, predict))
print("Accuracy Score : ",accuracy_score(y_test, predict))
print("Precision Score : ",precision_score(y_test, predict))
print("Recall Score : ",recall_score(y_test, predict))
print("F1 Score : ",f1_score(y_test, predict))
target_names = ['Female', 'Male']
print(classification_report(y_test, predict, target_names=target_names))
```

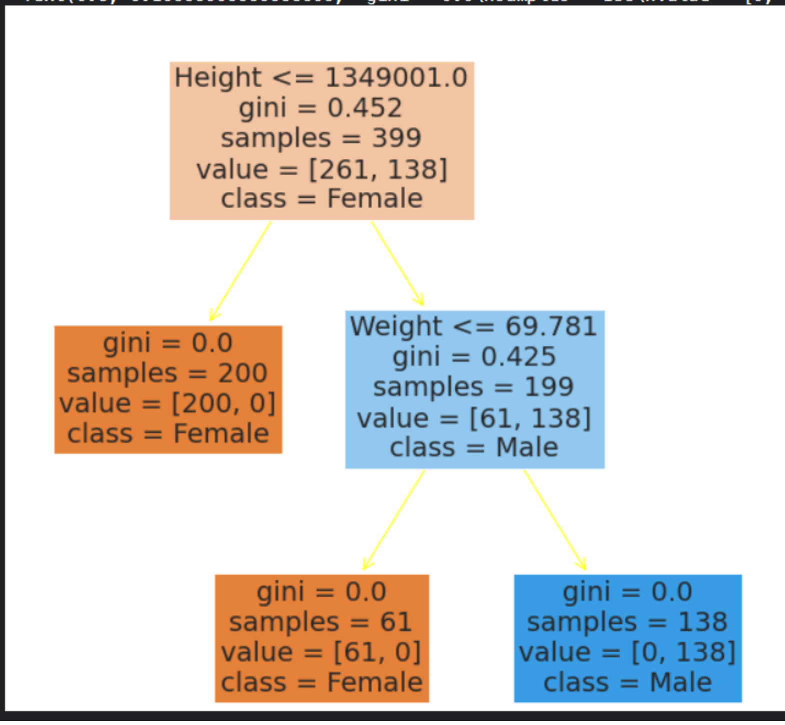
```
Confusion Matrix :
 [[62  0]
 [ 0 38]]
Accuracy Score :  1.0
Precision Score :  1.0
Recall Score :  1.0
F1 Score :  1.0
              precision    recall  f1-score   support

      Female       1.00      1.00      1.00        62
        Male       1.00      1.00      1.00        38

    accuracy                           1.00       100
   macro avg       1.00      1.00      1.00       100
weighted avg       1.00      1.00      1.00       100
```

As we can see from the evaluation metrics, the model has an accuracy of 100% even without hyperparameter tuning. Since this is a small and easy dataset, we were able to get a 100% accuracy. But this need not be true for complex datasets.

```
fig = plt.figure(figsize=(10,10))
out = tree.plot_tree(fittedModel, feature_names=["Height", "Weight"], class_names=["Female", "Male"], filled=True)
for outs_ in out:
    arrow = outs_.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor('yellow')
out
```

```
[12… [Text(0.4, 0.8333333333333334, 'Height <= 1349001.0\ngini = 0.452\nsamples = 399\nvalue = [261, 138]\nclass = Female'),
 Text(0.2, 0.5, 'gini = 0.0\nsamples = 200\nvalue = [200, 0]\nclass = Female'),
 Text(0.6, 0.5, 'Weight <= 69.781\ngini = 0.425\nsamples = 199\nvalue = [61, 138]\nclass = Male'),
 Text(0.4, 0.16666666666666666, 'gini = 0.0\nsamples = 61\nvalue = [61, 0]\nclass = Female'),
 Text(0.8, 0.16666666666666666, 'gini = 0.0\nsamples = 138\nvalue = [0, 138]\nclass = Male')]
```

```
 Text(0.6, 0.5, 'Weight <= 69.781\ngini = 0.425\nsamples = 199\nvalue = [61, 138]\nclass = Male'),
 Text(0.4, 0.16666666666666666, 'gini = 0.0\nsamples = 61\nvalue = [61, 0]\nclass = Female'),
 Text(0.8, 0.16666666666666666, 'gini = 0.0\nsamples = 138\nvalue = [0, 138]\nclass = Male')]
```



This is the obtained Decision tree
gridsearchCV is used to tune the hyperparameters.
'min_impurity_decrease' is an efficient parameter to prevent overfitting of the model.

Question 1 B

Question 1 B

+ Code + Markdown

[13]:

```python
# RandomForestClassifier hyperparameters
grid_params = {"max_depth": [None, 1, 2, 4, 8],"min_samples_split": list(np.arange(2, 70, 10)),
               "min_samples_leaf": np.arange(1, 20, 2),
               "min_impurity_decrease": [0,0.1,0.2],
               "max_features": [1, 2]}
grid_model = GridSearchCV(DecisionTreeClassifier(),param_grid=grid_params,verbose=True)
grid_model.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 2100 candidates, totalling 10500 fits
```

[13…

```
GridSearchCV(estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': [None, 1, 2, 4, 8],
                         'max_features': [1, 2],
                         'min_impurity_decrease': [0, 0.1, 0.2],
                         'min_samples_leaf': array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19]),
                         'min_samples_split': [2, 12, 22, 32, 42, 52, 62]},
             verbose=True)
```

+ Code + Markdown

[14]:

```python
grid_model.best_params_
```

[14…

```
{'max_depth': None,
 'max_features': 1,
 'min_impurity_decrease': 0,
 'min_samples_leaf': 5,
 'min_samples_split': 42}
```

[15]:

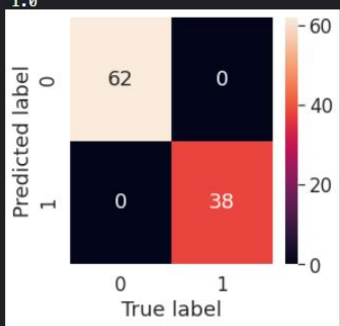
```python
model = DecisionTreeClassifier(max_depth=grid_model.best_params_["max_depth"],
 max_features=grid_model.best_params_["max_features"],
 min_samples_leaf=grid_model.best_params_["min_samples_leaf"],
 min_impurity_decrease = grid_model.best_params_["min_impurity_decrease"],
 min_samples_split=grid_model.best_params_["min_samples_split"])
fittedModel = model.fit(x_train, y_train)
```

+ Code + Markdown

Heat map

```
[16]: predict = fittedModel.predict(x_test)
      sns.set(font_scale=1.5)
      fig, ax = plt.subplots(figsize=(4, 4))
      ax = sns.heatmap(confusion_matrix(y_test, predict), annot=True, cbar=True)
      plt.xlabel("True label")
      plt.ylabel("Predicted label")
      print(accuracy_score(y_test, predict))
      print(precision_score(y_test, predict))
      print(recall_score(y_test, predict))
      print(f1_score(y_test, predict))
      plt.show()
```

```
1.0
1.0
1.0
1.0
```

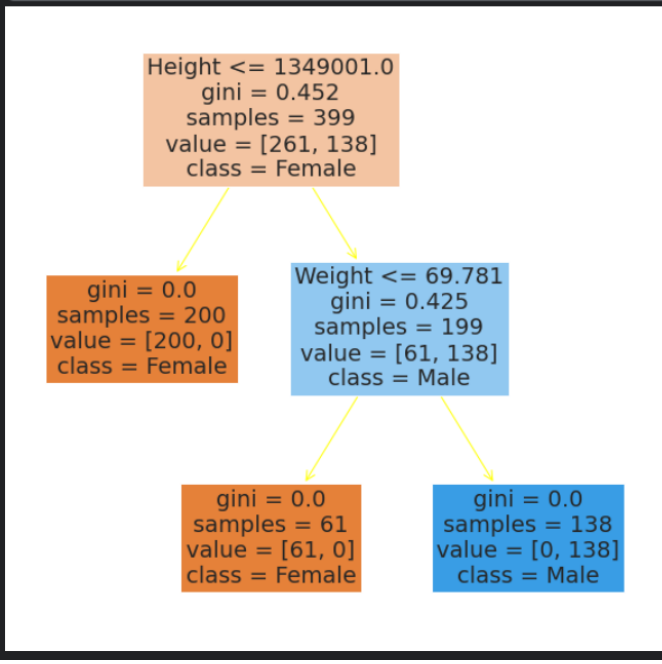



```
[17]: target_names = ['Female', 'Male']
      print(classification_report(y_test, predict, target_names=target_names))
```

```
              precision    recall  f1-score   support

      Female       1.00      1.00      1.00        62
        Male       1.00      1.00      1.00        38

    accuracy                           1.00       100
   macro avg       1.00      1.00      1.00       100
weighted avg       1.00      1.00      1.00       100
```

Here we got the same accuracy for both the trees.
But, for new complex real time data, this hyperparameter tuned decision tree may perform better

```
[18]: fig = plt.figure(figsize=(10,10))
      out = tree.plot_tree(fittedModel, feature_names=["Height", "Weight"], class_names=["Female", "Male"], filled=True)
      for o in out:
          arrow = o.arrow_patch
          if arrow is not None:
              arrow.set_edgecolor('yellow')
```



Reshape y_train and y_test to a one dimensional numpy array to suit KNN input format (avoid warnings)

```
[19]: y_train = y_train.values.ravel()
      y_train.shape

[19… (399,)
```

```
[20]: y_test = y_test.values.ravel()
      y_test.shape

[20… (100,)
```

Question 2A

Implementation of KNN

1. Import necesaary class from sklearn
2. Initialise model
3. Fit model on training dataset

Question 2 A

[21]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(x_train, y_train)
```

[21… KNeighborsClassifier()

Make predictions on test split

+ Code + Markdown

[22]:
```
y_pred = knn_model.predict(x_test)
```

Evaluating the model-confusion matrix and classification report

[23]:
```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[55  7]
 [ 3 35]]
              precision    recall  f1-score   support

           0       0.95      0.89      0.92        62
           1       0.83      0.92      0.88        38

    accuracy                           0.90       100
   macro avg       0.89      0.90      0.90       100
weighted avg       0.90      0.90      0.90       100
```

+ Code + Markdown

Selecting best K value (number of nearest neighbors)

[24]:

```python
errors = []
# Trying different K values
for i in range(1, 60):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train, y_train)
    pred_i = knn.predict(x_test)
    errors.append(np.mean(pred_i != y_test))
```

+ Code + Markdown

[25]:

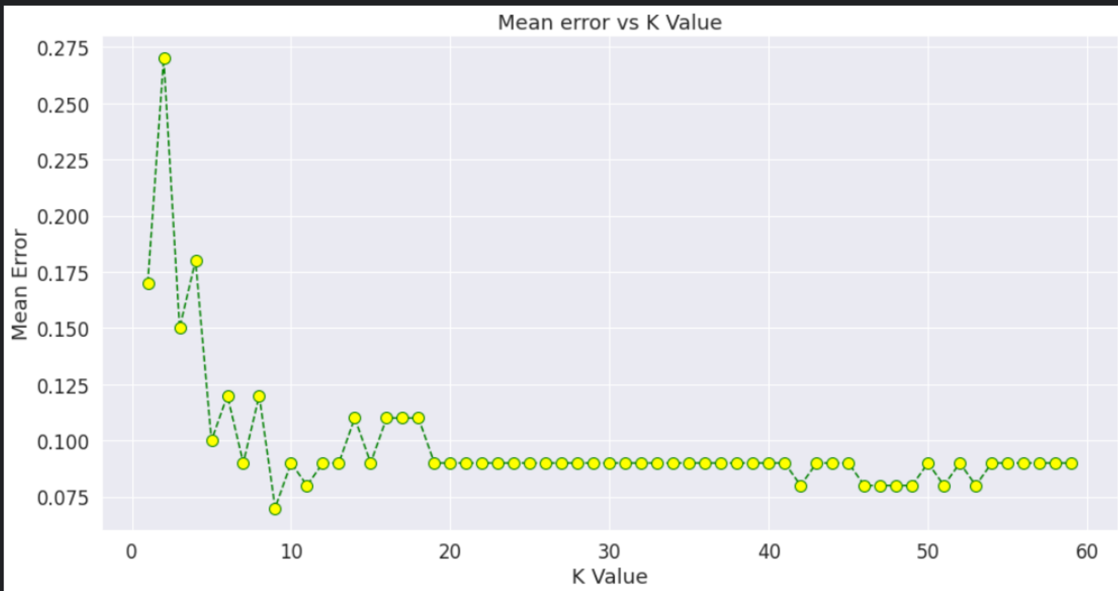
```python
#We print the values for each k value.
for i in range (1,60):
    print('Error is',errors[i-1],', When k is',i)
```

Run All Markdown Draft Session (1h:46m)

```
Error is 0.17 , When k is 1
Error is 0.27 , When k is 2
Error is 0.15 , When k is 3
Error is 0.18 , When k is 4
Error is 0.1 , When k is 5
Error is 0.12 , When k is 6
Error is 0.09 , When k is 7
Error is 0.12 , When k is 8
Error is 0.07 , When k is 9
Error is 0.09 , When k is 10
Error is 0.08 , When k is 11
Error is 0.09 , When k is 12
Error is 0.09 , When k is 13
Error is 0.11 , When k is 14
Error is 0.09 , When k is 15
Error is 0.11 , When k is 16
Error is 0.11 , When k is 17
Error is 0.11 , When k is 18
Error is 0.09 , When k is 19
Error is 0.09 , When k is 20
Error is 0.09 , When k is 21
Error is 0.09 , When k is 22
Error is 0.09 , When k is 23
Error is 0.09 , When k is 24
Error is 0.09 , When k is 25
Error is 0.09 , When k is 26
Error is 0.09 , When k is 27
Error is 0.09 , When k is 28
Error is 0.09 , When k is 29
Error is 0.09 , When k is 30
Error is 0.09 , When k is 31
Error is 0.09 , When k is 32
Error is 0.09 , When k is 33
Error is 0.09 , When k is 34
Error is 0.09 , When k is 35
Error is 0.09 , When k is 36
Error is 0.09 , When k is 37
Error is 0.09 , When k is 38
Error is 0.09 , When k is 39
Error is 0.09 , When k is 40
Error is 0.09 , When k is 41
Error is 0.08 , When k is 42
Error is 0.09 , When k is 43
Error is 0.09 , When k is 44
Error is 0.09 , When k is 45
Error is 0.08 , When k is 46
Error is 0.08 , When k is 47
Error is 0.08 , When k is 48
Error is 0.08 , When k is 49
Error is 0.09 , When k is 50
Error is 0.08 , When k is 51
Error is 0.09 , When k is 52
Error is 0.08 , When k is 53
Error is 0.09 , When k is 54
Error is 0.09 , When k is 55
Error is 0.09 , When k is 56
Error is 0.09 , When k is 57
Error is 0.09 , When k is 58
Error is 0.09 , When k is 59
```

```
[26]:
plt.figure(figsize=(16, 8))
plt.plot(range(1, 60), errors, color='green', linestyle='dashed', marker='o', markerfacecolor='yellow', markersize=10)
plt.title('Mean error vs K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

```
[26… Text(0, 0.5, 'Mean Error')
```



Creating a new model with n_neighbors parameter as 25

```
[27]:
knn_model = KNeighborsClassifier(n_neighbors=25)
knn_model.fit(x_train, y_train)
y_pred = knn_model.predict(x_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.85      0.92        62
           1       0.81      1.00      0.89        38

    accuracy                           0.91       100
   macro avg       0.90      0.93      0.91       100
weighted avg       0.93      0.91      0.91       100
```

+ Code + Markdown

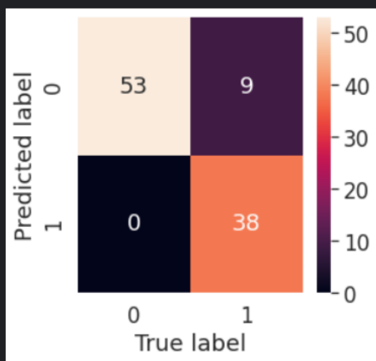
Confusion matrix:

Confusion matrix:

9 labels that are predicted as 0 but true label is 1

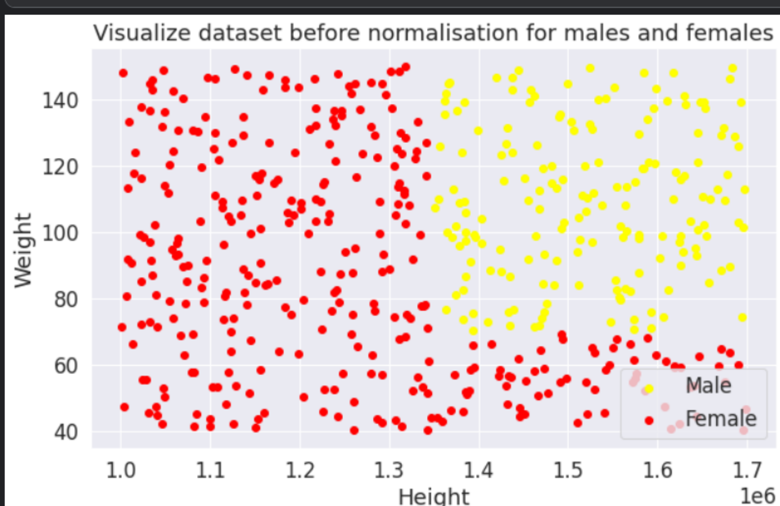
[28]:
```
sns.set(font_scale=1.5)
fig, ax = plt.subplots(figsize=(4, 4))
ax = sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cbar=True)
plt.xlabel("True label")
plt.ylabel("Predicted label")
```

[28…
```
Text(4.5, 0.5, 'Predicted label')
```



Visualising the dataset (data points under each target class) before normalization

[29]:
```
df_out = pd.merge(df,target['Gender'],how = 'right',left_index = True,
right_index = True)
plt.figure(figsize=(10, 6))
plt.scatter(df_out.Height[df_out.Gender==1],df_out.Weight[df_out.Gender==1], c="yellow")
plt.scatter(df_out.Height[df_out.Gender==0],df_out.Weight[df_out.Gender==0], c="red")
plt.title("Visualize dataset before normalisation for males and females")
plt.xlabel("Height")
plt.ylabel("Weight")
plt.legend(["Male", "Female"]);
```



```
[30]: x_train.head()
```

[30…

	Height	Weight
437	1.209246e+06	99.596811
273	1.431273e+06	53.645809
58	1.126067e+06	74.055455
379	1.365429e+06	99.802530
78	1.356239e+06	125.954853

+ Code + Markdown

Question 2 B

Min-max normalisation
Use same scalar we used for the train split to transform the test split

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_train = scaler.fit_transform(x_train)
scaled_test = scaler.transform(x_test)
```

+ Code + Markdown

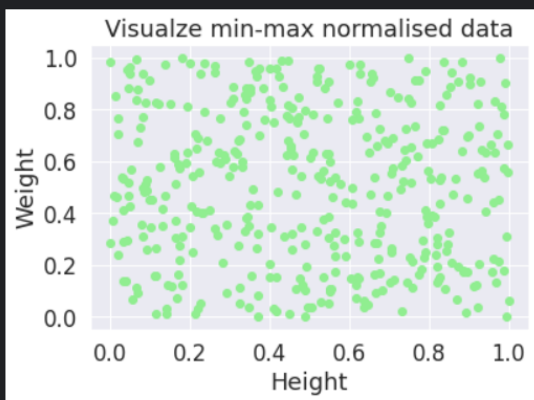
```
[32]: scaled_train[:5]
```

```
[32… array([[0.29818431, 0.54274625],
       [0.61676671, 0.12328948],
       [0.17883253, 0.30959585],
       [0.52228851, 0.54462412],
       [0.50910181, 0.78335165]])
```

Visualizing data points after normalization

```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
x = scaled_train[:,0]
y = scaled_train[:,1]
ax.scatter(x, y, color = 'lightgreen')
plt.title("Visualze min-max normalised data")
plt.xlabel("Height")
plt.ylabel("Weight")
```

```
[33… Text(0, 0.5, 'Weight')
```



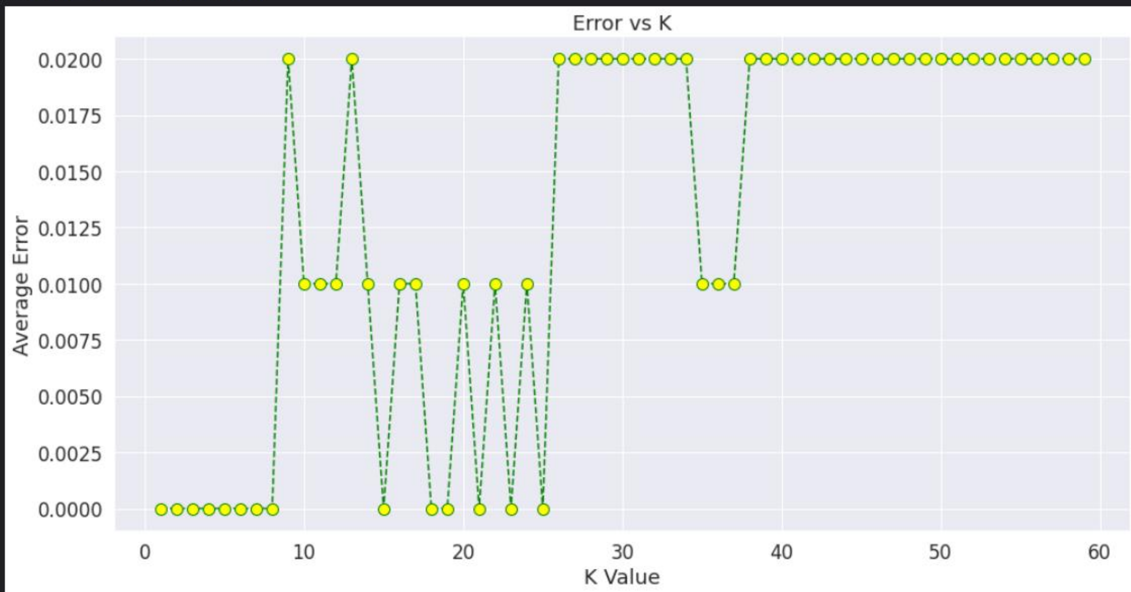
+ Code + Markdown

finding the optimal K value by plotting the error vs K value. But this time I use the normalised dataset to train the model

```
[34]:
errors = []
# Trying different K values
for i in range(1, 60):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(scaled_train, y_train)
    pred_i = knn.predict(scaled_test)
    errors.append(np.mean(pred_i != y_test))
```

```
plt.figure(figsize=(16, 8))
plt.plot(range(1, 60), errors, color='green', linestyle='dashed', marker='o',markerfacecolor='yellow', markersize=10)
plt.title('Error vs K')
plt.xlabel('K Value')
plt.ylabel('Average Error')
```

[35… Text(0, 0.5, 'Average Error')



+ Code + Markdown

```
[36]: #We print the values for each k value.
      for i in range(1,60):
          print('Error is',errors[i-1],', When k is',i)
```

```
Error is 0.0 , When k is 1
Error is 0.0 , When k is 2
Error is 0.0 , When k is 3
Error is 0.0 , When k is 4
Error is 0.0 , When k is 5
Error is 0.0 , When k is 6
Error is 0.0 , When k is 7
Error is 0.0 , When k is 8
Error is 0.02 , When k is 9
Error is 0.01 , When k is 10
Error is 0.01 , When k is 11
Error is 0.01 , When k is 12
Error is 0.02 , When k is 13
Error is 0.01 , When k is 14
Error is 0.0 , When k is 15
Error is 0.01 , When k is 16
Error is 0.01 , When k is 17
Error is 0.0 , When k is 18
Error is 0.0 , When k is 19
Error is 0.01 , When k is 20
Error is 0.0 , When k is 21
Error is 0.01 , When k is 22
Error is 0.0 , When k is 23
Error is 0.01 , When k is 24
Error is 0.0 , When k is 25
Error is 0.02 , When k is 26
Error is 0.02 , When k is 27
Error is 0.02 , When k is 28
Error is 0.02 , When k is 29
Error is 0.02 , When k is 30
Error is 0.02 , When k is 31
Error is 0.02 , When k is 32
Error is 0.02 , When k is 33
Error is 0.02 , When k is 34
Error is 0.01 , When k is 35
Error is 0.01 , When k is 36
Error is 0.01 , When k is 37
Error is 0.02 , When k is 38
Error is 0.02 , When k is 39
Error is 0.02 , When k is 40
Error is 0.02 , When k is 41
Error is 0.02 , When k is 42
Error is 0.02 , When k is 43
Error is 0.02 , When k is 44
Error is 0.02 , When k is 45
Error is 0.02 , When k is 46
Error is 0.02 , When k is 47
Error is 0.02 , When k is 48
Error is 0.02 , When k is 49
Error is 0.02 , When k is 50
Error is 0.02 , When k is 51
Error is 0.02 , When k is 52
Error is 0.02 , When k is 53
Error is 0.02 , When k is 54
Error is 0.02 , When k is 55
Error is 0.02 , When k is 56
Error is 0.02 , When k is 57
Error is 0.02 , When k is 58
Error is 0.02 , When k is 59
```

K=5 is the optimal value as obtained from the above graph.
Now I will create a new model with n_neighbors tuned to 5

[37]:
```python
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(scaled_train, y_train)
```

[37… KNeighborsClassifier()

Prediction on the test datset

[38]:
```python
y_pred = knn_model.predict(scaled_test)
```

\+ Code + Markdown

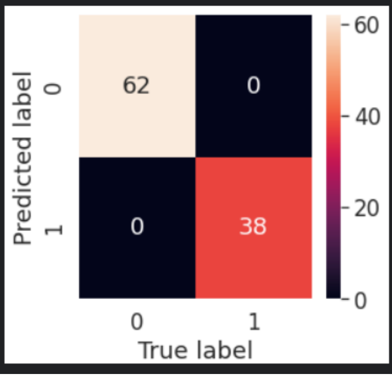
Classification Report and confusion matrix of the predicted values

[39]:
```python
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        62
           1       1.00      1.00      1.00        38

    accuracy                           1.00       100
   macro avg       1.00      1.00      1.00       100
weighted avg       1.00      1.00      1.00       100
```

[40]:
```python
sns.set(font_scale=1.5)
fig, ax = plt.subplots(figsize=(4, 4))
ax = sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cbar=True)
plt.xlabel("True label")
plt.ylabel("Predicted label")
```

[40… Text(4.5, 0.5, 'Predicted label')



Again Using gridsearch to find the best parameters (like for n_neighbors)

```
params = {'n_neighbors': np.arange(5, 60, 5)}
gs = GridSearchCV(KNeighborsClassifier(), param_grid=params, cv=5, verbose=True)
gs.fit(scaled_train, y_train)
```

```
Fitting 5 folds for each of 11 candidates, totalling 55 fits
[41… GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55])},
             verbose=True)
```

+ Code + Markdown

Got best value for n_neighbors as 5 which matches with the same value obtained from the K vs error plot above

```
gs.best_params_
```

```
[42… {'n_neighbors': 5}
```

+ Code + Markdown

Accuracy of 100% on the test split data is achieved

```
[43]: gs.score(scaled_test, y_test)
```

```
[43… 1.0
```

Question 2C

ROC curves and AUC

Question 2 C

[44]:
```python
from sklearn.metrics import plot_roc_curve
```

[45]:
```python
model_diff_neighbors = KNeighborsClassifier(n_neighbors=5)
model_diff_neighbors.fit(x_train, y_train)
plot_roc_curve(model_diff_neighbors, scaled_test, y_test)
```

[45…
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7fca64a9f550>
```

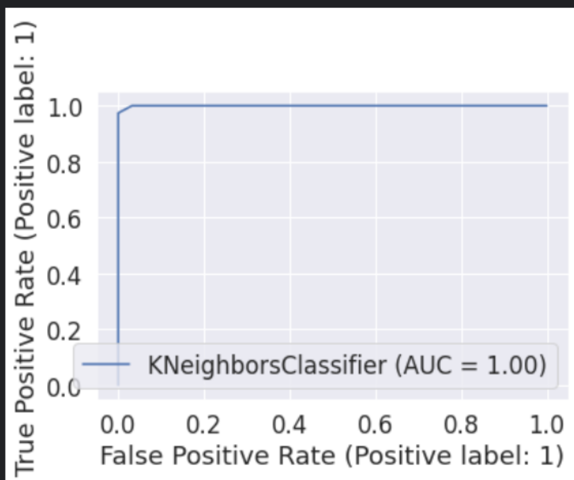
+ Code + Markdown

Got slightly bent roc curve (AUC not exactly one - rounded off in the plot) if I use different number of n_neighbors like 11 as the best hyperparameter is 5 for n_neighbors

```
[46]:
model_diff_neighbors = KNeighborsClassifier(n_neighbors=9)
model_diff_neighbors.fit(scaled_train, y_train)
plot_roc_curve(model_diff_neighbors, scaled_test, y_test)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_roc_curve
is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the
class methods: :meth:`sklearn.metric.RocCurveDisplay.from_predictions` or :meth:`sklearn.metric.RocCurveDispla
y.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```

```
[46…
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7fca64a66510>
```



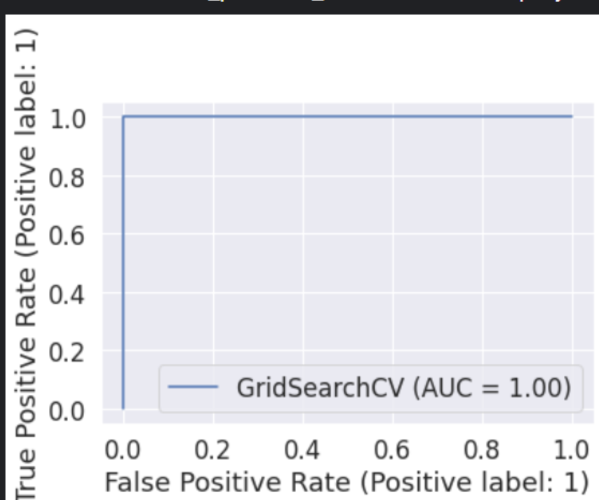
Perfect ROC curve

Perfect plot with well tuned hyperparameter(n_neighbors=5) and fitted on normalised data. AUC is one, hence this is the most optimum solution

```
[47]:
plot_roc_curve(gs, scaled_test, y_test)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_roc_curve
is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the
class methods: :meth:`sklearn.metric.RocCurveDisplay.from_predictions` or :meth:`sklearn.metric.RocCurveDispla
y.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```

```
[47…
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7fca649f8f90>
```



+ Code + Markdown

Thankyou