K-NN

Image 1

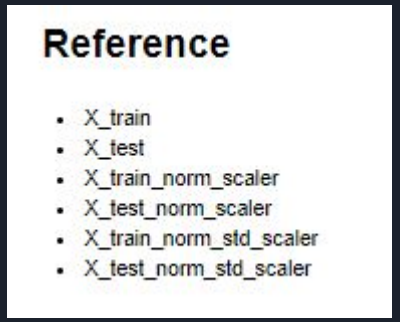


Image 2

In [9]:

```
# Design KNN model, for example wih k = 3 as an example
knn = KNeighborsClassifier(n_neighbors = 3)
# train the model
knn.fit(X_train, Y_train)
## Predict using X_test
y_pred = knn.predict(X_test)
print('Accuracy with norm: ', knn.score(X_test, Y_test))

# Normalized with scale
knn.fit(X_train_norm_scaler, Y_train)
y_pred_scale = knn.predict(X_test_norm_scaler)
print('Accuracy max-min norm: ', knn.score(X_test_norm_scaler, Y_test))

# Normalized with standard scale
knn.fit(X_train_norm_std_scaler, Y_train)
y_pred_std_scale = knn.predict(X_test_norm_std_scaler)
print('Accuracy std norm: ', knn.score(X_test_norm_std_scaler, Y_test))
```

```
Accuracy with norm:  0.7359307359307359
Accuracy max-min norm:  0.7532467532467533
Accuracy std norm:  0.7445887445887446
```

In [10]:

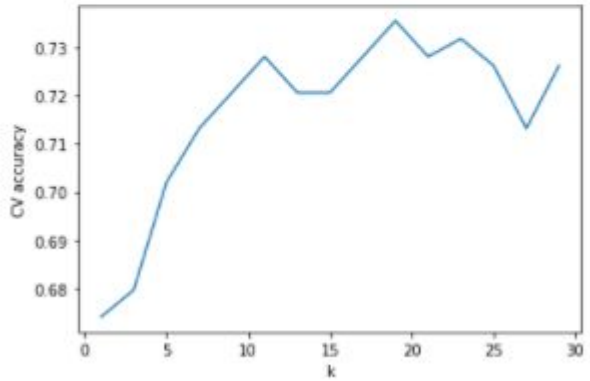
```
print('Without norm')
print(confusion_matrix(Y_test, y_pred))
print('')
print('Max-min norm')
print(confusion_matrix(Y_test, y_pred_scale))
print('')
print('Std norm')
print(confusion_matrix(Y_test, y_pred_std_scale))
```

```
Without norm
[[128  27]
 [ 34  42]]

Max-min norm
[[126  29]
 [ 28  48]]

Std norm
[[123  32]
 [ 27  49]]
```

K-NN

Image 3

```
K value =  1 Score [0.65363128 0.69832402 0.67039106]
K value =  3 Score [0.67039106 0.69832402 0.67039106]
K value =  5 Score [0.65921788 0.74860335 0.69832402]
K value =  7 Score [0.68715084 0.74860335 0.70391061]
K value =  9 Score [0.70391061 0.73184358 0.72625698]
K value =  11 Score [0.70391061 0.74860335 0.73184358]
K value =  13 Score [0.69832402 0.74301676 0.72067039]
K value =  15 Score [0.69832402 0.75977654 0.70391061]
K value =  17 Score [0.70391061 0.77653631 0.70391061]
K value =  19 Score [0.70949721 0.77653631 0.72067039]
K value =  21 Score [0.69273743 0.77094972 0.72067039]
K value =  23 Score [0.69273743 0.77653631 0.72625698]
K value =  25 Score [0.70391061 0.76536313 0.70949721]
K value =  27 Score [0.70949721 0.75977654 0.67039106]
K value =  29 Score [0.72067039 0.76536313 0.69273743]
```




```
The best k value is =  19
```

Image 4

No norm

```
In [16]:
kNN=KNeighborsClassifier(n_neighbors=np.array(k_values)[cv_k_scores.index(np.array(cv_k_scores).max())])
model=kNN.fit(X_train,Y_train)

In [17]:
y_pred=model.predict(X_test)

In [18]:
print(accuracy_score(Y_test, y_pred))
print(confusion_matrix(Y_test, y_pred))
```

```
0.7662337662337663
[[132  23]
 [ 31  45]]
```

With max-min scale norm and std_scale norm

```
In [19]:
kNN=KNeighborsClassifier(n_neighbors=np.array(k_values)[cv_k_scores_scale.index(np.array(cv_k_scores_scale).max())])
model=kNN.fit(X_train_norm_scaler,Y_train)
y_pred=model.predict(X_test_norm_scaler)
print('Max-min norm')
print(accuracy_score(Y_test, y_pred))
print(confusion_matrix(Y_test, y_pred))
```

```
Max-min norm
0.7142857142857143
[[124  31]
 [ 35  41]]
```

```
In [20]:
kNN=KNeighborsClassifier(n_neighbors=
                         np.array(k_values)[cv_k_scores_std_scale.index(np.array(cv_k_scores_std_scale).max())])
model=kNN.fit(X_train_norm_std_scaler,Y_train)
y_pred=model.predict(X_test_norm_std_scaler)
print('Std norm')
print(accuracy_score(Y_test, y_pred))
print(confusion_matrix(Y_test, y_pred))
```

```
Std norm
0.7359307359307359
[[127  28]
 [ 33  43]]
```

K-NN

Image 5

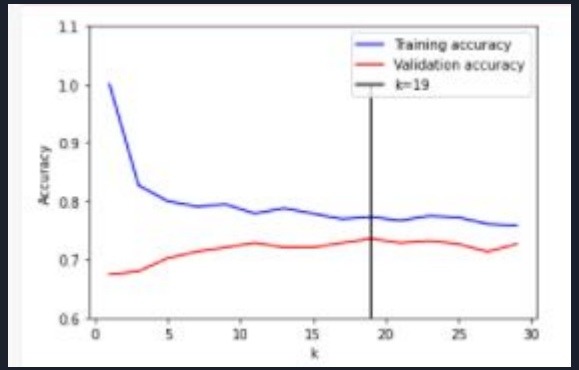


Image 6

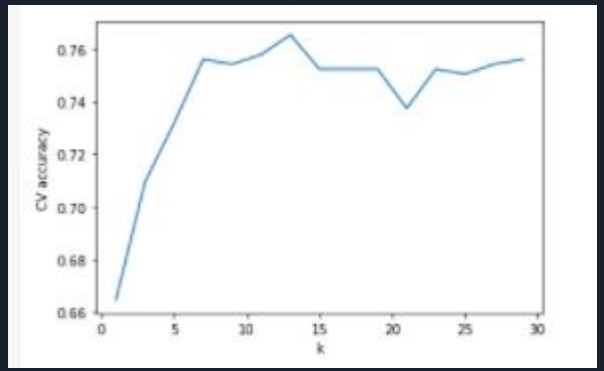
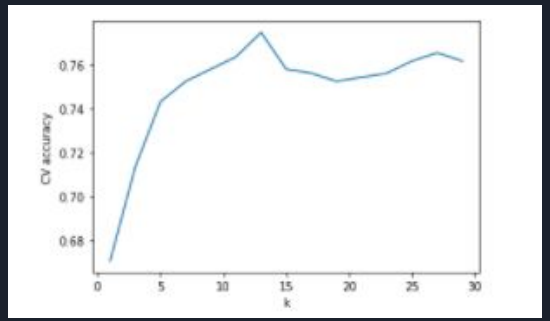


Image 7



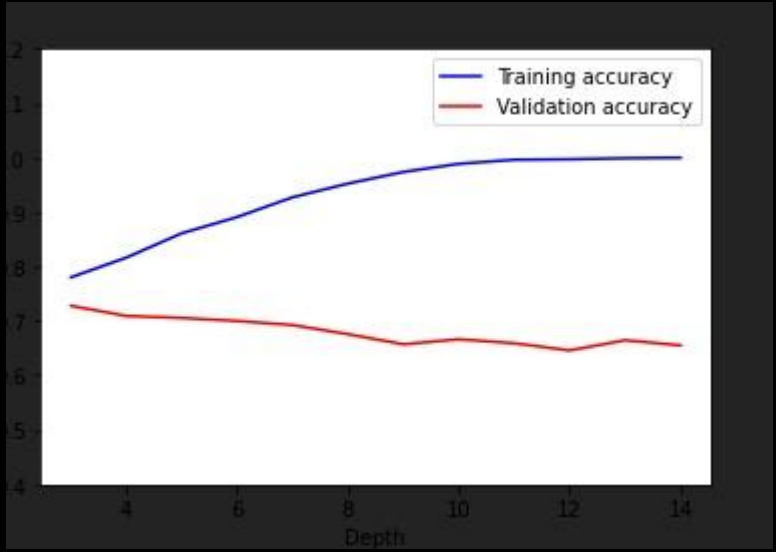
No norm

In [23]:

```python
# Train de model
my_model = dtree_model.best_estimator_
my_tree=my_model.fit(X_train, Y_train)

#Predict using X_test
y_predicted = my_model.predict(X_test)

# Results
print('Accuracy = ' , accuracy_score(Y_test, y_predicted))
print('')

def measures_from_conf_matrix(confusion_matrix):
    TP = confusion_matrix[0][0]
    FN = confusion_matrix[0][1]
    FP = confusion_matrix[1][0]
    TN = confusion_matrix[1][1]

    sensitivity = TP/(TP+FN)
    print('Sensitivity = ',sensitivity)
    specifity = TN/(TN+FP)
    print('Specifity = ',specifity)
    return

measures_from_conf_matrix(confusion_matrix(Y_test, y_predicted))

print('')
print(confusion_matrix(Y_test, y_predicted))
```

```
Accuracy =  0.7662337662337663

Sensitivity =  0.8129032258064516
Specifity =  0.6710526315789473
```

Decision Trees



Decision Trees

Max-min scale

```
param_grid = { 'criterion':['gini','entropy'],'max_depth': np.arange(3, 15), 'min_samples_leaf' : [50,
dtree_model=tree.DecisionTreeClassifier()
dtree_model = GridSearchCV(dtree_model, param_grid, cv=5) # Cross validation of 5
dtree_model=dtree_model.fit(X_train_norm_scaler, Y_train)
print(dtree_model.best_params_)


# Train de model
my_model = dtree_model.best_estimator_
my_tree=my_model.fit(X_train_norm_scaler, Y_train)
#Predict using X_test
y_predicted = my_model.predict(X_test_norm_scaler)
# Results
print('Accuracy = ' , accuracy_score(Y_test, y_predicted))
print('')
measures_from_conf_matrix(confusion_matrix(Y_test, y_predicted))

print('')
print(confusion_matrix(Y_test, y_predicted))
my_tree.get_depth()
```

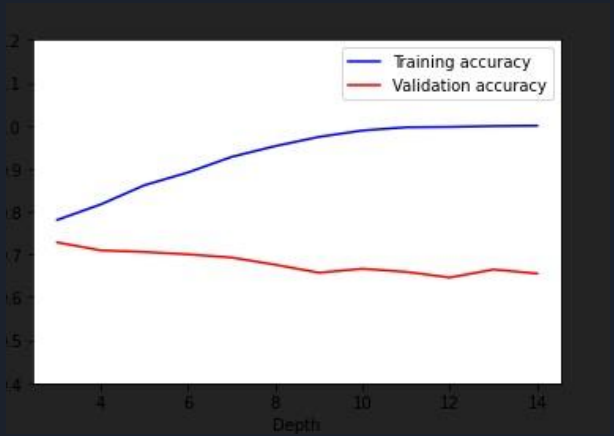
```
{'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 50}
Accuracy =  0.7186147186147186

Sensitivity =  0.7290322580645161
Specifity =  0.6973684210526315

[[113  42]
 [ 23  53]]


4
```

Std norm

```
param_grid = { 'criterion':['gini','entropy'],'max_depth': np.arange(3, 15), 'min_samples_leaf' : [50,
dtree_model=tree.DecisionTreeClassifier()
dtree_model = GridSearchCV(dtree_model, param_grid, cv=5) # Cross validation of 5
dtree_model=dtree_model.fit(X_train_norm_std_scaler, Y_train)
print(dtree_model.best_params_)


# Train de model
my_model = dtree_model.best_estimator_
my_tree=my_model.fit(X_train_norm_std_scaler, Y_train)
#Predict using X_test
y_predicted = my_model.predict(X_test_norm_std_scaler)

# Results
print('Accuracy = ' , accuracy_score(Y_test, y_predicted))
print('')

measures_from_conf_matrix(confusion_matrix(Y_test, y_predicted))

print('')
print(confusion_matrix(Y_test, y_predicted))

my_tree.get_depth()
```

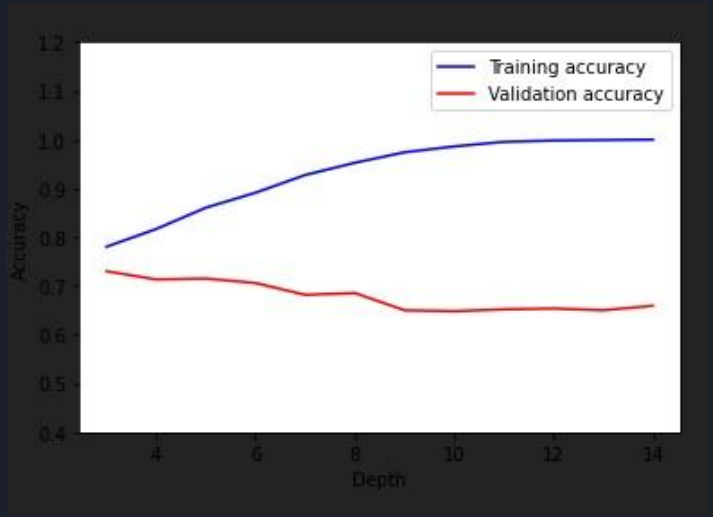
```
{'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 50}
Accuracy =  0.7532467532467533

Sensitivity =  0.7677419354838709
Specifity =  0.7236842105263158

[[119  36]
 [ 21  55]]


4
```

Decision Trees

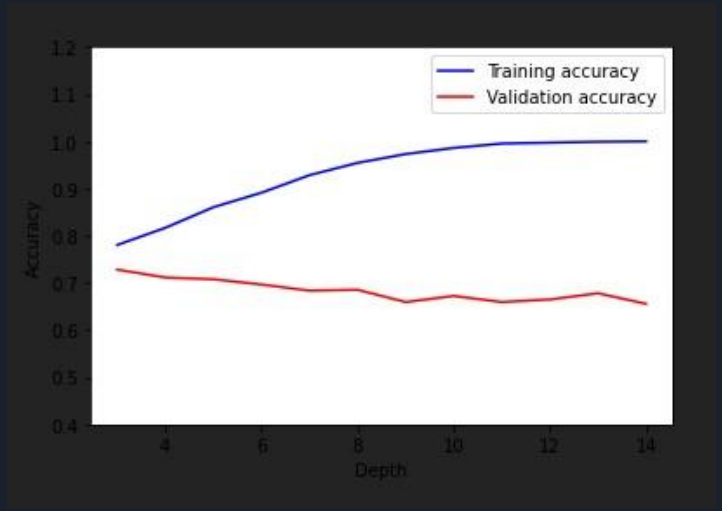


NON-NORMALIZED

MAX-MIN STD



STANDARDIZATION



MLP
Non-normalized

```python
from sklearn.neural_network import MLPClassifier

# MLP is sensitive to feature scaling.

# Starting from initial random weights, multi-layer perceptron (MLP) minimizes the loss function by
# repeatedly updating these weights. After computing the loss, a backward pass propagates it from
# the output layer to the previous layers, providing each weight parameter with an update value meant t

activation_functs = ['identity', 'logistic', 'tanh', 'relu']
learning_rates = [0.001, 0.005, 0.01, 0.05, 0.1]

# The number of hidden nodes should be less than twice the size of the nodes in the input layer.
# The number of hidden nodes should be 2/3 the size of input nodes, plus the size of the output node.

for i in activation_functs:
    for j in learning_rates:

        MLP = MLPClassifier(hidden_layer_sizes =(8,), activation = i, solver = 'sgd', random_state = 1,
                            learning_rate_init = j , shuffle = True ,max_iter = 400, validation_fraction =

        MLP = MLP.fit(X_train, Y_train)
        print('Activation function = ',i,', Learning rate = ', j , 'Score = ',MLP.score(X_test, Y_test)
```

```
Activation function =  identity , Learning rate =  0.001 Score =  0.670995670995671
Activation function =  identity , Learning rate =  0.005 Score =  0.6883116883116883
Activation function =  identity , Learning rate =  0.01 Score =  0.329004329004329
Activation function =  identity , Learning rate =  0.05 Score =  0.329004329004329
Activation function =  identity , Learning rate =  0.1 Score =  0.670995670995671
Activation function =  logistic , Learning rate =  0.001 Score =  0.670995670995671
Activation function =  logistic , Learning rate =  0.005 Score =  0.7142857142857143
Activation function =  logistic , Learning rate =  0.01 Score =  0.6926406926406926
Activation function =  logistic , Learning rate =  0.05 Score =  0.6233766233766234
Activation function =  logistic , Learning rate =  0.1 Score =  0.670995670995671
Activation function =  tanh , Learning rate =  0.001 Score =  0.6666666666666666
Activation function =  tanh , Learning rate =  0.005 Score =  0.670995670995671
Activation function =  tanh , Learning rate =  0.01 Score =  0.670995670995671
Activation function =  tanh , Learning rate =  0.05 Score =  0.670995670995671
Activation function =  tanh , Learning rate =  0.1 Score =  0.670995670995671
Activation function =  relu , Learning rate =  0.001 Score =  0.6666666666666666
Activation function =  relu , Learning rate =  0.005 Score =  0.670995670995671
Activation function =  relu , Learning rate =  0.01 Score =  0.670995670995671
Activation function =  relu , Learning rate =  0.05 Score =  0.670995670995671
Activation function =  relu , Learning rate =  0.1 Score =  0.670995670995671
```

MLP

MAX-MIN normalized

Max-min norm

```
# MLP is sensitive to feature scaling.

activation_functs = ['identity', 'logistic', 'tanh', 'relu']
learning_rates = [0.001, 0.005, 0.01, 0.05, 0.1]

for i in activation_functs:
    for j in learning_rates:

        MLP = MLPClassifier(hidden_layer_sizes =(8,), activation = i, solver = 'sgd', random_state = 1,
                        learning_rate_init = j , shuffle = True ,max_iter = 400, validation_fraction =

        MLP = MLP.fit(X_train_norm_scaler, Y_train)
        print('Activation function = ',i,', Learning rate = ', j , 'Score = ',
              MLP.score(X_test_norm_scaler, Y_test))
```

```
print('Best result = Activation function =  relu , Learning rate =  0.01 Score =  0.7748917748917749')
```

```
Best result = Activation function =  relu , Learning rate =  0.01 Score =  0.7748917748917749
```

MLP
Standardization

Std norm

```
# MLP is sensitive to feature scaling.

activation_functs = ['identity', 'logistic', 'tanh', 'relu']
learning_rates = [0.001, 0.005, 0.01, 0.05, 0.1]

for i in activation_functs:
    for j in learning_rates:

        MLP = MLPClassifier(hidden_layer_sizes =(8,), activation = i, solver = 'sgd', random_state = 1,
                        learning_rate_init = j , shuffle = True ,max_iter = 400, validation_fraction =

        MLP = MLP.fit(X_train_norm_std_scaler, Y_train)
        print('Activation function = ',i,', Learning rate = ', j , 'Score = ',
              MLP.score(X_test_norm_std_scaler, Y_test))
```

```
print('Best result = Activation function =  relu , Learning rate =  0.01 Score =  0.7748917748917749')
```

```
Best result = Activation function =  relu , Learning rate =  0.01 Score =  0.7748917748917749
```