Machine Learning

Lab Sheet 3

K Fold Cross Validation Methods

S Abhishek

AM.EN.U4CSE19147

Collab Link

1 - KNN Implementation From Scratch

Data Set Visualization

Pregnancies: Number of times pregnant

Glucose: Plasma glucose concentration over 2 hours in an oral glucose tolerance test

BloodPressure: Diastolic blood pressure (mm Hg)

SkinThickness: Triceps skin fold thickness (mm)

Insulin: 2-Hour serum insulin (mu U/ml)

BMI: Body mass index (weight in kg/(height in m)2)

DiabetesPedigreeFunction: Diabetes pedigree function (a function which scores likelihood of diabetes based on family history)

Age: Age (years)

Outcome: Class variable (0 if non-diabetic, 1 if diabetic)

Importing the Data Set

```
import pandas as pd
import numpy as np
```

```
# Loading Data Set

df = pd.read_csv('/content/Diabetes.csv')

df.head()
```

```
   Pregnancies  Glucose  BloodPressure  ...  DiabetesPedigreeFunction  Age
Outcome
0            6      148             72  ...                     0.627   50
1
1            1       85             66  ...                     0.351   31
0
2            8      183             64  ...                     0.672   32
1
3            1       89             66  ...                     0.167   21
0
4            0      137             40  ...                     2.288   33
1

[5 rows x 9 columns]
```

Cleaning the Data Set

```
# Check if any column has any 0 values

df.eq(0).any()
```

```
Pregnancies                 True
Glucose                     True
BloodPressure               True
SkinThickness               True
Insulin                     True
BMI                         True
DiabetesPedigreeFunction    False
Age                         False
Outcome                     True
dtype: bool
```

```
# Columns with 0's in it

# We don't mind about the Pregnancy attribute though it has 0 value in it
since it's mean for non pregnant samples

Columns_with_0 = ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']

for column in Columns_with_0:

  df[column] = df[column].replace(0, np.NaN) # Replace 0's to NaN
```

```
  mean = int(df[column].mean(skipna=True)) # Find the mean of the column
skipping the Null values

  df[column] = df[column].replace(np.NaN, mean) # Replace NaN with the mean
value

# Check if any column has any 0 values

df.eq(0).any()
```

```
Pregnancies                 True
Glucose                    False
BloodPressure              False
SkinThickness              False
Insulin                    False
BMI                        False
DiabetesPedigreeFunction   False
Age                        False
Outcome                     True
dtype: bool
```

Splitting the Data Set

```
# Here the Target Label is Outcome
# We split the Feature and the Target

from sklearn.model_selection import train_test_split

X = df.iloc[:,:-1].values

Y = df.iloc[:,-1:].values

# Splitting dataset into train and test set

X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size = 0.2,
random_state = 0 )
```

Standardization

- Standardization comes into picture when features of input data set have large differences between their ranges, or simply when they are measured in different measurement units.
- These differences in the ranges of initial features causes trouble to many machine learning models.

- For example, for the models that are based on distance computation, if one of the features has a broad range of values, the distance will be governed by this particular feature.

```
from sklearn.preprocessing import StandardScaler

st_scaler = StandardScaler()

X_train = st_scaler.fit_transform(X_train)

X_test = st_scaler.transform(X_test)
```

KNN

```
# K Nearest Neighbors Classification

from scipy.stats import mode

class KNN() :

  def __init__( self, K ) :

    self.K = K


  def fit( self, X_train, Y_train ) : # Function to store training set

    self.X_train, self.Y_train = X_train, Y_train

    self.m, self.n = X_train.shape # No of Rows & Columns in Training Data
Set


  def predict( self, X_test ) : # Function for prediction

    self.X_test = X_test

    self.m_test, self.n = X_test.shape # No of Rows & Columns in Test Data
Set

    Y_predict = np.zeros( self.m_test )

    for i in range( self.m_test ) :

      neighbors = self.find_neighbors( self.X_test[i] ) # Find the K nearest
neighbors from current test example

      # most frequent class in K neighbors
```

```
      Y_predict[i] = mode( neighbors )[0][0]

    return Y_predict


  def find_neighbors( self, x ) : # Function to find the K nearest neighbors
to current test example

    # Calculate all the Euclidean distances between current test example x
and training set X_train

    euclidean_distances = np.zeros( self.m )

    for i in range( self.m ) :

      euclidean_distances[i] = self.euclidean( x, self.X_train[i] )

      Y_train_sorted = self.Y_train[euclidean_distances.argsort()] # Sort
Y_train according to euclidean_distance_array and store it in Y_train_sorted

    return Y_train_sorted[:self.K]


  def euclidean( self, x, x_train ) : # Function to calculate euclidean
distance

    return np.sqrt( np.sum( np.square( x - x_train ) ) )
```

Training the Model

```
# Training the Model on the Train Data Set

model = KNN( K = 5 )

model.fit( X_train, Y_train )
```

Predicting Test Data Set

```
# Prediction on test set

Y_pred = model.predict( X_test )
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix

print(confusion_matrix(Y_test,Y_pred))
```

```
[[87 20]
 [17 30]]
```

F1 Score

```
from sklearn.metrics import f1_score

print(f1_score(Y_test,Y_pred))
```

```
0.6185567010309279
```

Accuracy Score

```
from sklearn.metrics import accuracy_score

print(accuracy_score(Y_test,Y_pred))
```

```
0.7597402597402597
```

2 - KNN Automized Implementation

Importing the Dataset

```
import pandas as pd
import numpy as np

# Loading Data Set

df = pd.read_csv('/content/Diabetes.csv')

df.head()
```

```
   Pregnancies  Glucose  BloodPressure  ...  DiabetesPedigreeFunction  Age
Outcome
0            6      148             72  ...                     0.627   50
1
1            1       85             66  ...                     0.351   31
0
2            8      183             64  ...                     0.672   32
1
3            1       89             66  ...                     0.167   21
0
```

```
4            0      137             40  ...                     2.288   33
1

[5 rows x 9 columns]
```

Cleaning the Data Set

```
# Check if any column has any 0 values

df.eq(0).any()
```

```
Pregnancies                 True
Glucose                     True
BloodPressure               True
SkinThickness               True
Insulin                     True
BMI                         True
DiabetesPedigreeFunction    False
Age                         False
Outcome                     True
dtype: bool
```

```
# Columns with 0's in it

# We don't mind about the Pregnancy attribute though it has 0 value in it
since it's mean for non pregnant samples

Columns_with_0 = ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']

for column in Columns_with_0:

  df[column] = df[column].replace(0, np.NaN) # Replace 0's to NaN

  mean = int(df[column].mean(skipna=True)) # Find the mean of the column
skipping the Null values

  df[column] = df[column].replace(np.NaN, mean) # Replace NaN with the mean
value

# Check if any column has any 0 values

df.eq(0).any()
```

```
Pregnancies                 True
Glucose                    False
BloodPressure              False
SkinThickness              False
Insulin                    False
BMI                        False
```

```
DiabetesPedigreeFunction    False
Age                         False
Outcome                      True
dtype: bool
```

Splitting the Data Set

```
# Here the Target Label is Outcome
# We split the Feature and the Target

from sklearn.model_selection import train_test_split

X = df.iloc[:,:-1].values

Y = df.iloc[:,-1:].values

# Splitting dataset into train and test set

X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size = 0.2,
random_state = 0 )
```

Standardization

```
from sklearn.preprocessing import StandardScaler

st_scaler = StandardScaler()

X_train = st_scaler.fit_transform(X_train)

X_test = st_scaler.transform(X_test)
```

KNN

Training the Model

```
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier( n_neighbors = 5 )
model.fit( X_train, Y_train.ravel() )

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

Predicting Test Data Set

```
# Prediction on test set

Y_pred = model.predict( X_test )
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix

print(confusion_matrix(Y_test,Y_pred))
```

```
[[87 20]
 [17 30]]
```

F1 Score

```
from sklearn.metrics import f1_score

print(f1_score(Y_test,Y_pred))
```

```
0.6185567010309279
```

Accuracy Score

```
from sklearn.metrics import accuracy_score

print(accuracy_score(Y_test,Y_pred))
```

```
0.7597402597402597
```

3 - K Fold Cross Validation Methods

Simple K Fold

```
from sklearn.model_selection import KFold

accuracy1 = []

kf = KFold(n_splits=5, random_state=None)

for train_index, test_index in kf.split(X):

    #print("Train:", train_index, "\nValidation:",test_index)

    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]
```

```
    # Standardization
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Training the Model
    model = KNeighborsClassifier( n_neighbors = 5 )
    model.fit( X_train, Y_train.ravel() )

    # Predicting Test Data Set
    Y_pred = model.predict( X_test )

    # Confusion Matrix
    print("\n\nConfusion Matrix\n\n", confusion_matrix(Y_test,Y_pred), end =
"\n")

    # F1 Score
    print("\nF1 Score : ", f1_score(Y_test,Y_pred), end = "\n")

    # Accuracy Score
    accuracy1.append(accuracy_score(Y_test, Y_pred))
    print("\nAccuracy Score : ", accuracy_score(Y_test,Y_pred))
```

```
Confusion Matrix

 [[83 17]
 [24 30]]

F1 Score :  0.594059405940594

Accuracy Score :  0.7337662337662337


Confusion Matrix

 [[73 17]
 [27 37]]

F1 Score :  0.6271186440677967

Accuracy Score :  0.7142857142857143


Confusion Matrix
```

```
 [[81 16]
 [26 31]]

F1 Score :  0.5961538461538461

Accuracy Score :  0.7272727272727273


Confusion Matrix

 [[90 24]
 [14 25]]

F1 Score :  0.5681818181818182

Accuracy Score :  0.7516339869281046


Confusion Matrix

 [[78 21]
 [19 35]]

F1 Score :  0.6363636363636364

Accuracy Score :  0.738562091503268
```

Startified K Fold

```
from sklearn.model_selection import StratifiedKFold

accuracy2 = []

skf = StratifiedKFold(n_splits=5, random_state=None)

for train_index, test_index in kf.split(X):

    #print("Train:", train_index, "\nValidation:",test_index)

    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]

    # Standardization
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Training the Model
```

```
    model = KNeighborsClassifier( n_neighbors = 5 )
    model.fit( X_train, Y_train.ravel() )

    # Predicting Test Data Set
    Y_pred = model.predict( X_test )

    # Confusion Matrix
    print("\n\nConfusion Matrix\n\n", confusion_matrix(Y_test,Y_pred), end =
"\n")

    # F1 Score
    print("\nF1 Score : ", f1_score(Y_test,Y_pred), end = "\n")

    # Accuracy Score
    accuracy2.append(accuracy_score(Y_test, Y_pred))
    print("\nAccuracy Score : ", accuracy_score(Y_test,Y_pred))
```

```
Confusion Matrix

 [[83 17]
 [24 30]]

F1 Score :  0.594059405940594

Accuracy Score :  0.7337662337662337


Confusion Matrix

 [[73 17]
 [27 37]]

F1 Score :  0.6271186440677967

Accuracy Score :  0.7142857142857143


Confusion Matrix

 [[81 16]
 [26 31]]

F1 Score :  0.5961538461538461

Accuracy Score :  0.7272727272727273
```

```
Confusion Matrix

 [[90 24]
 [14 25]]

F1 Score :  0.5681818181818182

Accuracy Score :  0.7516339869281046


Confusion Matrix

 [[78 21]
 [19 35]]

F1 Score :  0.6363636363636364

Accuracy Score :  0.738562091503268
```

Repeated K Fold

```
from sklearn.model_selection import RepeatedKFold

accuracy3 = []

kf = RepeatedKFold(n_splits=5, n_repeats=10, random_state=None)

for train_index, test_index in kf.split(X):

    #print("Train:", train_index, "\nValidation:",test_index)

    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]

    # Standardization
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Training the Model
    model = KNeighborsClassifier( n_neighbors = 5 )
    model.fit( X_train, Y_train.ravel() )

    # Predicting Test Data Set
    Y_pred = model.predict( X_test )

    # Confusion Matrix
```

```
    print("\n\nConfusion Matrix\n\n", confusion_matrix(Y_test,Y_pred), end =
"\n")

    # F1 Score
    print("\nF1 Score : ", f1_score(Y_test,Y_pred), end = "\n")

    # Accuracy Score
    accuracy3.append(accuracy_score(Y_test, Y_pred))
    print("\nAccuracy Score : ", accuracy_score(Y_test,Y_pred))
```

```
Confusion Matrix

 [[83 14]
 [19 38]]

F1 Score :  0.6972477064220183

Accuracy Score :  0.7857142857142857


Confusion Matrix

 [[83 20]
 [23 28]]

F1 Score :  0.5656565656565657

Accuracy Score :  0.7207792207792207


Confusion Matrix

 [[83 19]
 [24 28]]

F1 Score :  0.5656565656565657

Accuracy Score :  0.7207792207792207


Confusion Matrix

 [[85 16]
 [14 38]]

F1 Score :  0.7169811320754716
```

```
Accuracy Score :  0.803921568627451


Confusion Matrix

 [[75 22]
 [25 31]]

F1 Score :  0.5688073394495413

Accuracy Score :  0.6928104575163399


Confusion Matrix

 [[87 20]
 [19 28]]

F1 Score :  0.5894736842105263

Accuracy Score :  0.7467532467532467


Confusion Matrix

 [[72 22]
 [23 37]]

F1 Score :  0.6218487394957983

Accuracy Score :  0.7077922077922078


Confusion Matrix

 [[93 17]
 [18 26]]

F1 Score :  0.5977011494252873

Accuracy Score :  0.7727272727272727


Confusion Matrix

 [[77 15]
 [24 37]]
```

F1 Score : 0.654867256637168

Accuracy Score : 0.7450980392156863

Confusion Matrix

 [[77 20]
 [19 37]]

F1 Score : 0.6548672566371682

Accuracy Score : 0.7450980392156863

Confusion Matrix

 [[81 11]
 [25 37]]

F1 Score : 0.6727272727272728

Accuracy Score : 0.7662337662337663

Confusion Matrix

 [[79 24]
 [25 26]]

F1 Score : 0.5148514851485149

Accuracy Score : 0.6818181818181818

Confusion Matrix

 [[79 16]
 [23 36]]

F1 Score : 0.6486486486486487

Accuracy Score : 0.7467532467532467

Confusion Matrix

```
 [[88 18]
 [25 22]]

F1 Score :  0.5057471264367817

Accuracy Score :  0.7189542483660131


Confusion Matrix

 [[76 28]
 [19 30]]

F1 Score :  0.5607476635514018

Accuracy Score :  0.6928104575163399


Confusion Matrix

 [[85 18]
 [15 36]]

F1 Score :  0.6857142857142857

Accuracy Score :  0.7857142857142857


Confusion Matrix

 [[82 17]
 [20 35]]

F1 Score :  0.6542056074766355

Accuracy Score :  0.7597402597402597


Confusion Matrix

 [[81 16]
 [29 28]]

F1 Score :  0.5544554455445544

Accuracy Score :  0.7077922077922078
```

```
Confusion Matrix

 [[79 22]
 [21 31]]

F1 Score :  0.5904761904761905

Accuracy Score :  0.7189542483660131


Confusion Matrix

 [[83 17]
 [15 38]]

F1 Score :  0.7037037037037037

Accuracy Score :  0.7908496732026143


Confusion Matrix

 [[87 21]
 [15 31]]

F1 Score :  0.6326530612244897

Accuracy Score :  0.7662337662337663


Confusion Matrix

 [[85 15]
 [21 33]]

F1 Score :  0.6470588235294118

Accuracy Score :  0.7662337662337663


Confusion Matrix

 [[81 20]
 [21 32]]

F1 Score :  0.6095238095238096

Accuracy Score :  0.7337662337662337
```

```
Confusion Matrix

 [[77 22]
 [22 32]]

F1 Score :  0.5925925925925926

Accuracy Score :  0.7124183006535948


Confusion Matrix

 [[83  9]
 [29 32]]

F1 Score :  0.627450980392157

Accuracy Score :  0.7516339869281046


Confusion Matrix

 [[86 16]
 [19 33]]

F1 Score :  0.6534653465346534

Accuracy Score :  0.7727272727272727


Confusion Matrix

 [[90 18]
 [18 28]]

F1 Score :  0.6086956521739131

Accuracy Score :  0.7662337662337663


Confusion Matrix

 [[78 21]
 [21 34]]

F1 Score :  0.6181818181818182
```

```
Accuracy Score :  0.7272727272727273


Confusion Matrix

 [[83 14]
 [26 30]]

F1 Score :  0.6

Accuracy Score :  0.738562091503268


Confusion Matrix

 [[78 16]
 [28 31]]

F1 Score :  0.5849056603773585

Accuracy Score :  0.7124183006535948


Confusion Matrix

 [[92 20]
 [18 24]]

F1 Score :  0.5581395348837208

Accuracy Score :  0.7532467532467533


Confusion Matrix

 [[82 18]
 [21 33]]

F1 Score :  0.6285714285714287

Accuracy Score :  0.7467532467532467


Confusion Matrix

 [[79 16]
 [25 34]]
```

F1 Score : 0.6238532110091743

Accuracy Score : 0.7337662337662337

Confusion Matrix

 [[73 16]
 [24 40]]

F1 Score : 0.6666666666666666

Accuracy Score : 0.738562091503268

Confusion Matrix

 [[84 20]
 [17 32]]

F1 Score : 0.6336633663366337

Accuracy Score : 0.7581699346405228

Confusion Matrix

 [[84 16]
 [20 34]]

F1 Score : 0.6538461538461539

Accuracy Score : 0.7662337662337663

Confusion Matrix

 [[83 19]
 [23 29]]

F1 Score : 0.5799999999999998

Accuracy Score : 0.7272727272727273

Confusion Matrix

```
 [[77 23]
 [21 33]]

F1 Score :  0.6

Accuracy Score :  0.7142857142857143


Confusion Matrix

 [[83 18]
 [19 33]]

F1 Score :  0.6407766990291263

Accuracy Score :  0.7581699346405228


Confusion Matrix

 [[80 17]
 [23 33]]

F1 Score :  0.6226415094339622

Accuracy Score :  0.738562091503268


Confusion Matrix

 [[78 23]
 [26 27]]

F1 Score :  0.5242718446601942

Accuracy Score :  0.6818181818181818


Confusion Matrix

 [[82 18]
 [25 29]]

F1 Score :  0.5742574257425743

Accuracy Score :  0.7207792207792207
```

```
Confusion Matrix

 [[80 18]
 [15 41]]

F1 Score :  0.7130434782608696

Accuracy Score :  0.7857142857142857


Confusion Matrix

 [[76 19]
 [25 33]]

F1 Score :  0.6

Accuracy Score :  0.7124183006535948


Confusion Matrix

 [[90 16]
 [16 31]]

F1 Score :  0.6595744680851063

Accuracy Score :  0.7908496732026143


Confusion Matrix

 [[70 21]
 [23 40]]

F1 Score :  0.6451612903225806

Accuracy Score :  0.7142857142857143


Confusion Matrix

 [[98 13]
 [22 21]]

F1 Score :  0.5454545454545455

Accuracy Score :  0.7727272727272727
```

```
Confusion Matrix

 [[76 25]
 [15 38]]

F1 Score :  0.6551724137931034

Accuracy Score :  0.7402597402597403


Confusion Matrix

 [[81 11]
 [30 31]]

F1 Score :  0.6019417475728156

Accuracy Score :  0.7320261437908496


Confusion Matrix

 [[80 25]
 [19 29]]

F1 Score :  0.5686274509803922

Accuracy Score :  0.7124183006535948
```

Accuracy Check

```
print("Mean Accuracy of K Fold : ", np.mean(accuracy1))

print("Mean Accuracy of Stratified K Fold : ", np.mean(accuracy2))

print("Mean Accuracy of Repeated K Fold : ", np.mean(accuracy3))
```

```
Mean Accuracy of K Fold :  0.7331041507512095
Mean Accuracy of Stratified K Fold :  0.7331041507512095
Mean Accuracy of Repeated K Fold :  0.7411382734912146
```

- Comparing the Mean Accuracy of all 3 K Fold Cross Validation Methods, the accuracy of **Repeated K Fold** is High.

ROC AUC Score Method 1

AUC - ROC curve is a performance measurement for classification problem at various thresholds settings.

ROC is a probability curve and AUC represents degree or measure of separability.

It tells how much model is capable of distinguishing between classes.

Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s.

By analogy, Higher the AUC, better the model is at distinguishing between patients with disease and no disease.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20,
stratify = Y)

# Standardization
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Training the Model
model = KNeighborsClassifier( n_neighbors = 5 )
model.fit(X_train, Y_train.ravel())

# The function predict_proba() returns a numpy array of two columns.
# The first column is the probability that target=0 and the second column is
the probability that target = 1

Y_pred = model.predict_proba(X_test)

# Select the probability estimates of the positive class
fpr, tpr, threshold = roc_curve(Y_test, Y_pred[:, 1])

# Predicting Test Data Set
#Y_pred = model.predict(X_test)

#print("Accuracy : ", accuracy_score(Y_test, Y_pred))
# print("False Postive Rate (FPR) :", fpr)
# print("True Positive Rate (TPR) :", tpr)
# print("Threshold :",threshold)

print("ROC AUC Score :", auc(fpr, tpr))

ROC AUC Score : 0.7674074074074073
```

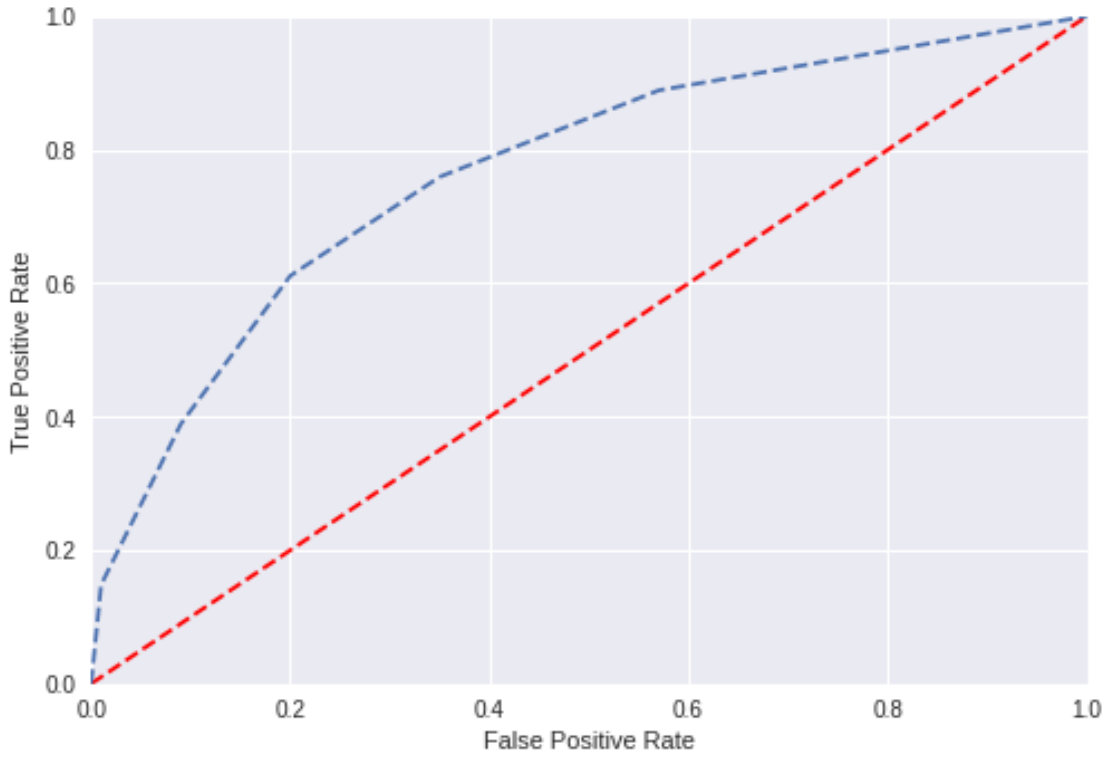
ROC AUC Score Method 2

```
from sklearn.metrics import roc_auc_score

# The function predict_proba() returns a numpy array of two columns.
# The first column is the probability that target=0 and the second column is
the probability that target = 1

Y_pred = model.predict_proba(X_test)

roc_auc_score = roc_auc_score(Y_test, Y_pred[:,1])

print("ROC AUC Score using Sklearn :", roc_auc_score)
```

```
ROC AUC Score using Sklearn : 0.7674074074074073
```

Plot ROC Curve

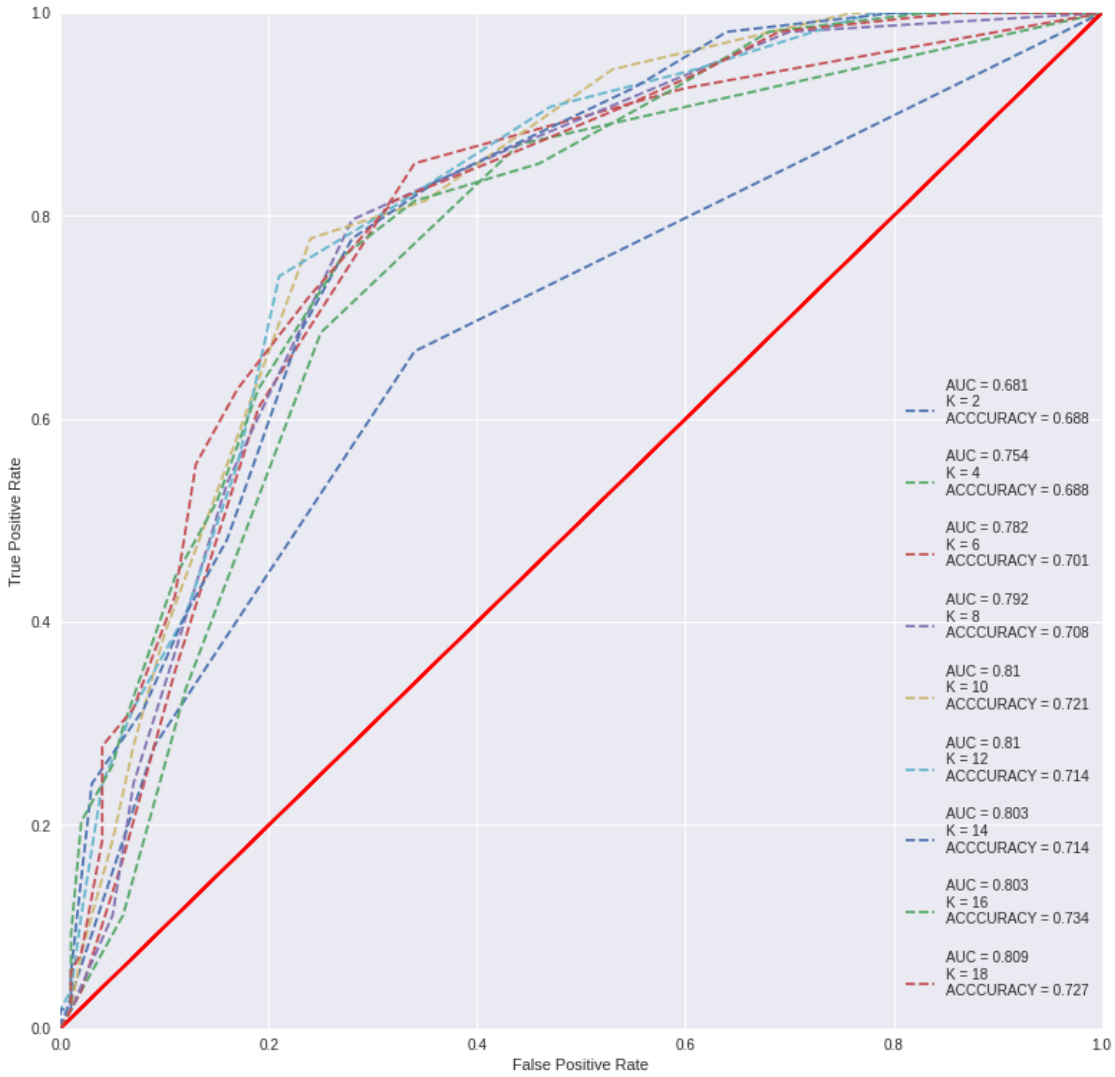
```
import matplotlib.pyplot as plt

plt.style.use('seaborn')

plt.plot([0, 1], [0, 1],'r--') # Set baseline model as diagonal

plt.xlim([0, 1]) # Set x axis limits
plt.ylim([0, 1]) # Set y axis limits

plt.xlabel('False Positive Rate') # X Label
plt.ylabel('True Positive Rate') # Y Label

plt.plot(fpr, tpr, linestyle='--', label = 'AUC = %0.2f' %roc_auc_score)
plt.show()
```



ROC Curve for Various N

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20,
stratify = Y)

# Standardization
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

plt.figure(figsize=(13,13))

for i in range(2,20,2):

  # Training the Model
  model = KNeighborsClassifier( n_neighbors = i )
  model.fit(X_train, Y_train.ravel())

  # The function predict_proba() returns a numpy array of two columns.
  # The first column is the probability that target=0 and the second column
is the probability that target = 1
```

```
    Y_pred = model.predict_proba(X_test)

    # Select the probability estimates of the positive class
    fpr, tpr, threshold = roc_curve(Y_test, Y_pred[:, 1])

    roc_auc = roc_auc_score(Y_test, Y_pred[:, 1])

    # Predicting Test Data Set
    Y_pred = model.predict(X_test)

    plt.plot([0, 1], [0, 1],'r-')
    plt.xlim([0, 1])
    plt.ylim([0, 1])

    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')

    plt.plot(fpr, tpr, linestyle='--', label = 'AUC = {}\nK = {}\nACCCURACY =
{}\n'.format(roc_auc.round(3),i,accuracy_score(Y_test, Y_pred).round(3)))
    plt.legend()

plt.show()
```



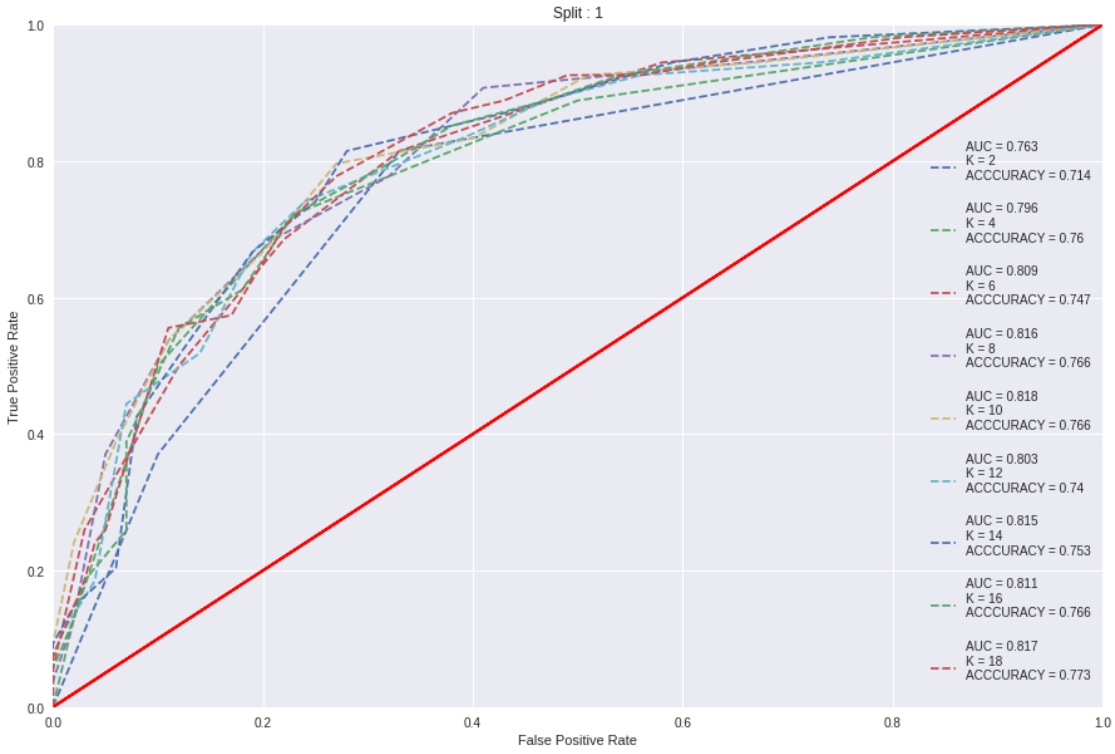
ROC Curve for Simple K Fold

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score

# Splitting dataset into train and test set

X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size = 0.2,
stratify = None, shuffle=False)

# Standardization Of Data Set

st_scaler = StandardScaler()
```

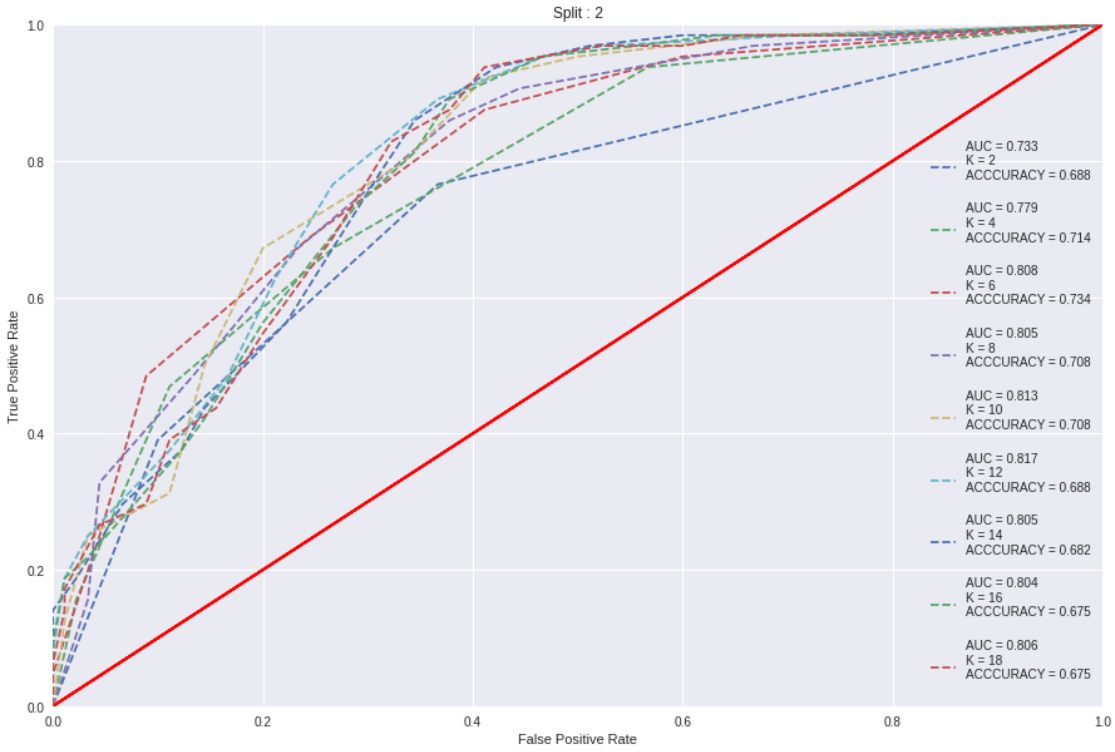
```
X_train = st_scaler.fit_transform(X_train)
X_test = st_scaler.transform(X_test)

kf = KFold(n_splits=5, random_state=None)

j = 1

for train_index, test_index in kf.split(X):

    #print("Train:", train_index, "\nValidation:",test_index)

    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]

    # Standardization
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Training the Model
    model = KNeighborsClassifier( n_neighbors = 5 )
    model.fit( X_train, Y_train.ravel() )

    plt.figure()
    plt.figure(figsize=(15,10))

    for i in range(2,20,2):

      # Training the Model
      model = KNeighborsClassifier( n_neighbors = i )
      model.fit(X_train, Y_train.ravel())

      # The function predict_proba() returns a numpy array of two columns.
      # The first column is the probability that target=0 and the second
column is the probability that target = 1

      Y_pred = model.predict_proba(X_test)

      # Select the probability estimates of the positive class
      fpr, tpr, threshold = roc_curve(Y_test, Y_pred[:, 1])

      roc_auc = roc_auc_score(Y_test, Y_pred[:, 1])

      # Predicting Test Data Set
      Y_pred = model.predict(X_test)

      plt.title('Split : {}'.format(j))
      plt.plot([0, 1], [0, 1],'r-')
```

```
        plt.xlim([0, 1])
        plt.ylim([0, 1])

        plt.ylabel('True Positive Rate')
        plt.xlabel('False Positive Rate')

        plt.plot(fpr, tpr, linestyle='--', label = 'AUC = {}\nK = {}\nACCCURACY
= {}\n'.format(roc_auc.round(3),i,accuracy_score(Y_test, Y_pred).round(3)))
        plt.legend()

    plt.show()
    j += 1
```

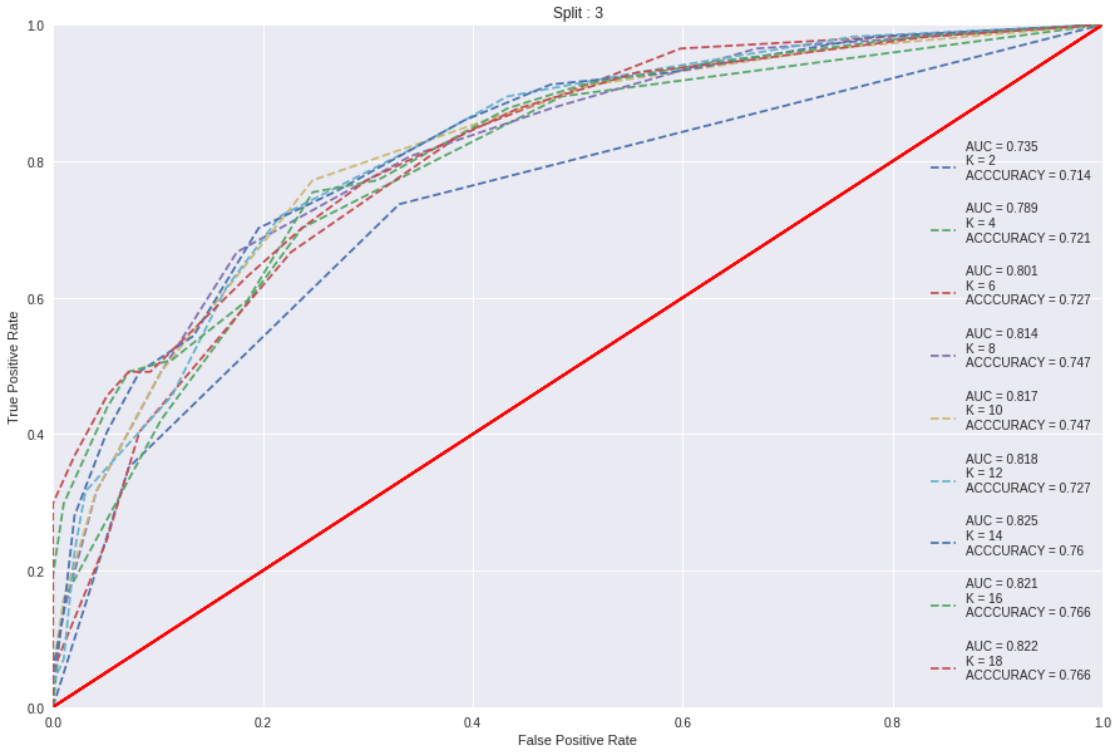
```
<Figure size 576x396 with 0 Axes>
```



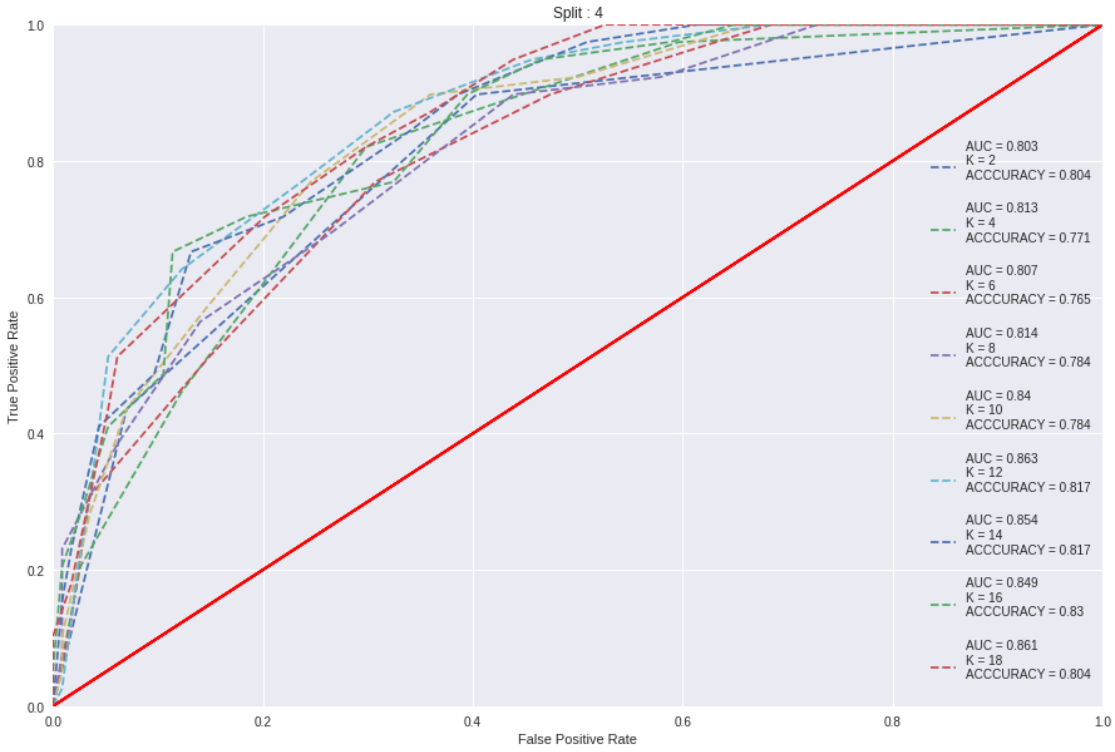

```
<Figure size 576x396 with 0 Axes>
```



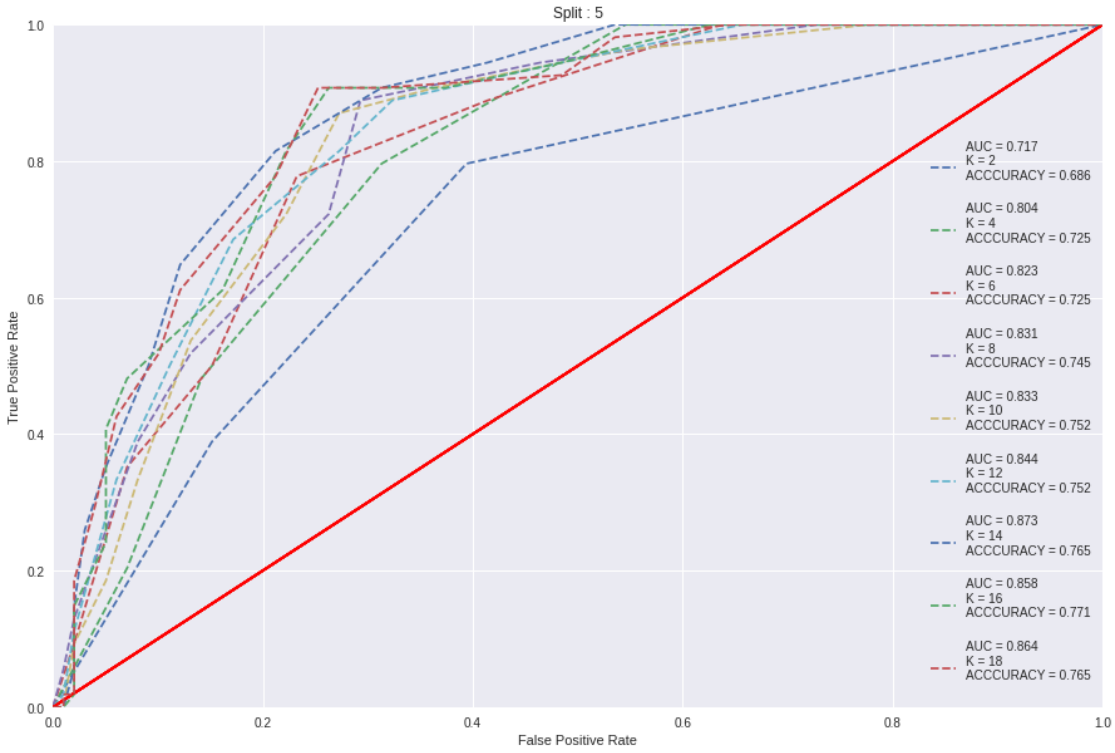
<Figure size 576x396 with 0 Axes>



<Figure size 576x396 with 0 Axes>



<Figure size 576x396 with 0 Axes>

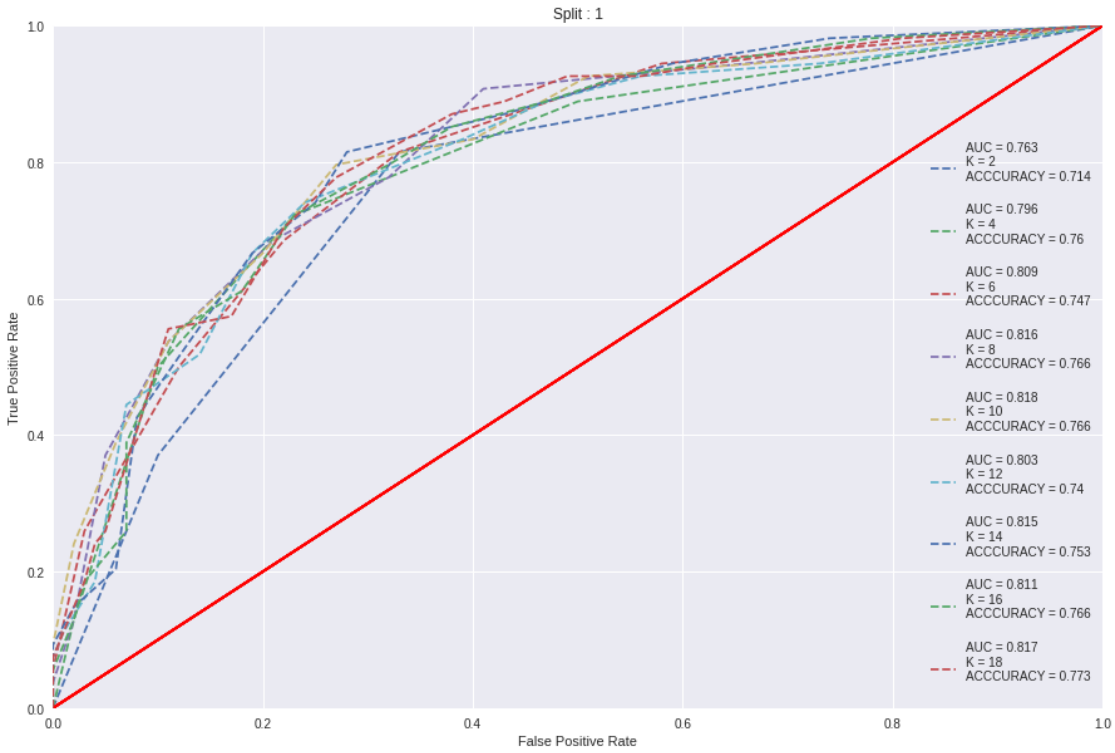


ROC Curve for Stratified K Fold

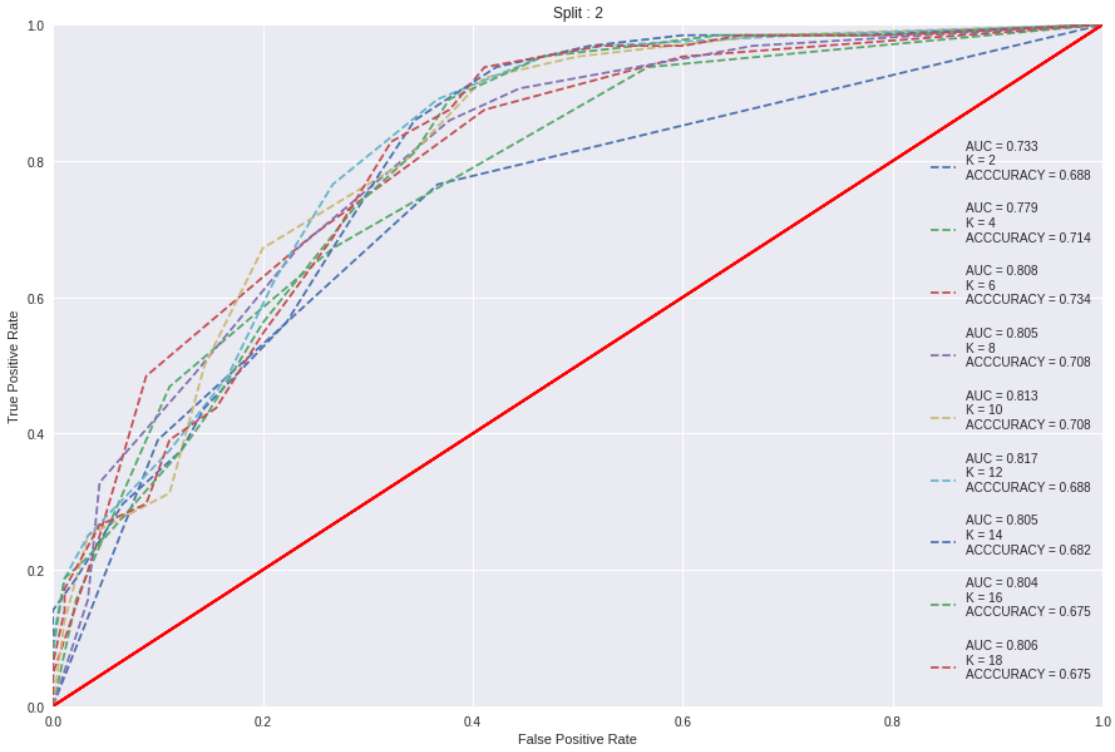
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score

skf = StratifiedKFold(n_splits=5, random_state=None)

j = 1

for train_index, test_index in kf.split(X):

    #print("Train:", train_index, "\nValidation:",test_index)

    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]

    # Standardization
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Training the Model
    model = KNeighborsClassifier( n_neighbors = 5 )
    model.fit( X_train, Y_train.ravel() )

    plt.figure()
    plt.figure(figsize=(15,10))

    for i in range(2,20,2):

      # Training the Model
      model = KNeighborsClassifier( n_neighbors = i )
      model.fit(X_train, Y_train.ravel())

      # The function predict_proba() returns a numpy array of two columns.
      # The first column is the probability that target=0 and the second
column is the probability that target = 1

      Y_pred = model.predict_proba(X_test)

      # Select the probability estimates of the positive class
      fpr, tpr, threshold = roc_curve(Y_test, Y_pred[:, 1])

      roc_auc = roc_auc_score(Y_test, Y_pred[:, 1])

      # Predicting Test Data Set
      Y_pred = model.predict(X_test)
```

```
        plt.title('Split : {}'.format(j))
        plt.plot([0, 1], [0, 1],'r-')
        plt.xlim([0, 1])
        plt.ylim([0, 1])

        plt.ylabel('True Positive Rate')
        plt.xlabel('False Positive Rate')

        plt.plot(fpr, tpr, linestyle='--', label = 'AUC = {}\nK = {}\nACCCURACY
= {}\n'.format(roc_auc.round(3),i,accuracy_score(Y_test, Y_pred).round(3)))
        plt.legend()

    plt.show()
    j += 1
```

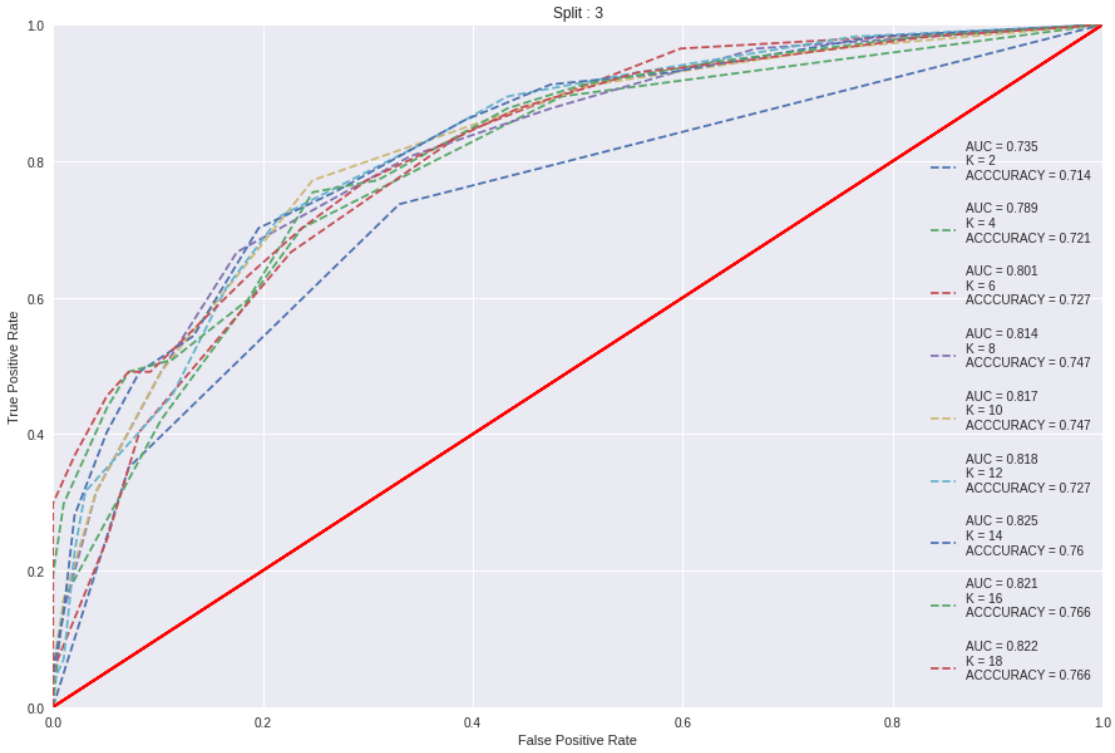
```
<Figure size 576x396 with 0 Axes>
```



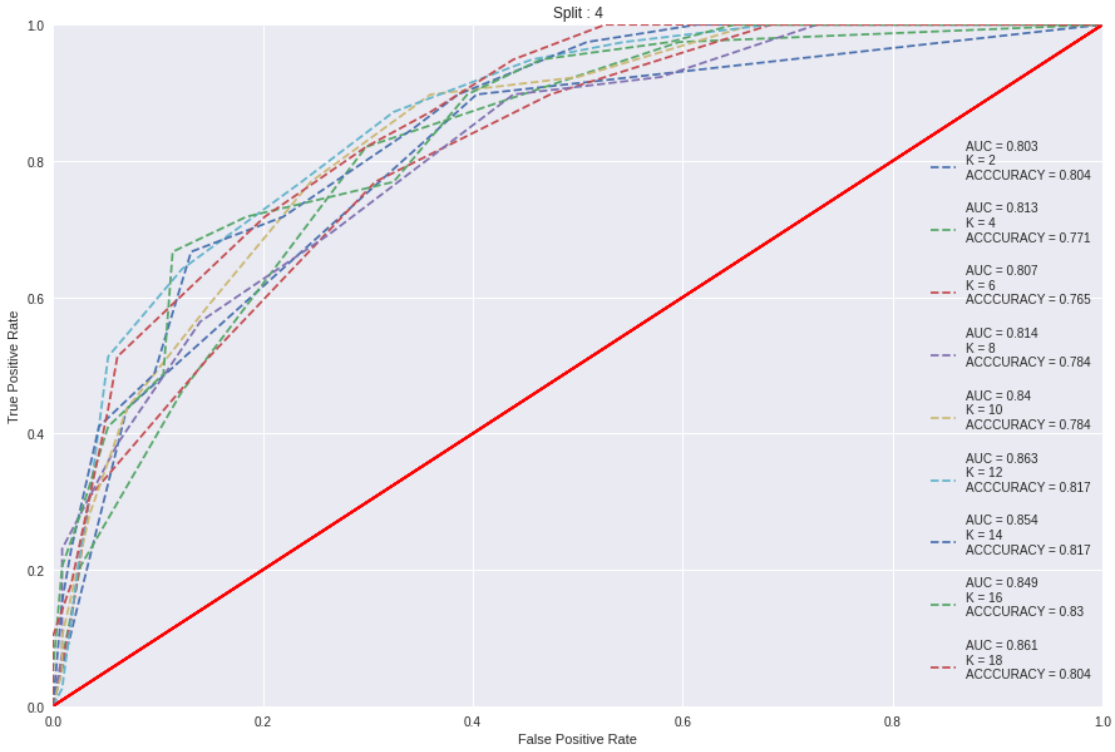

```
<Figure size 576x396 with 0 Axes>
```



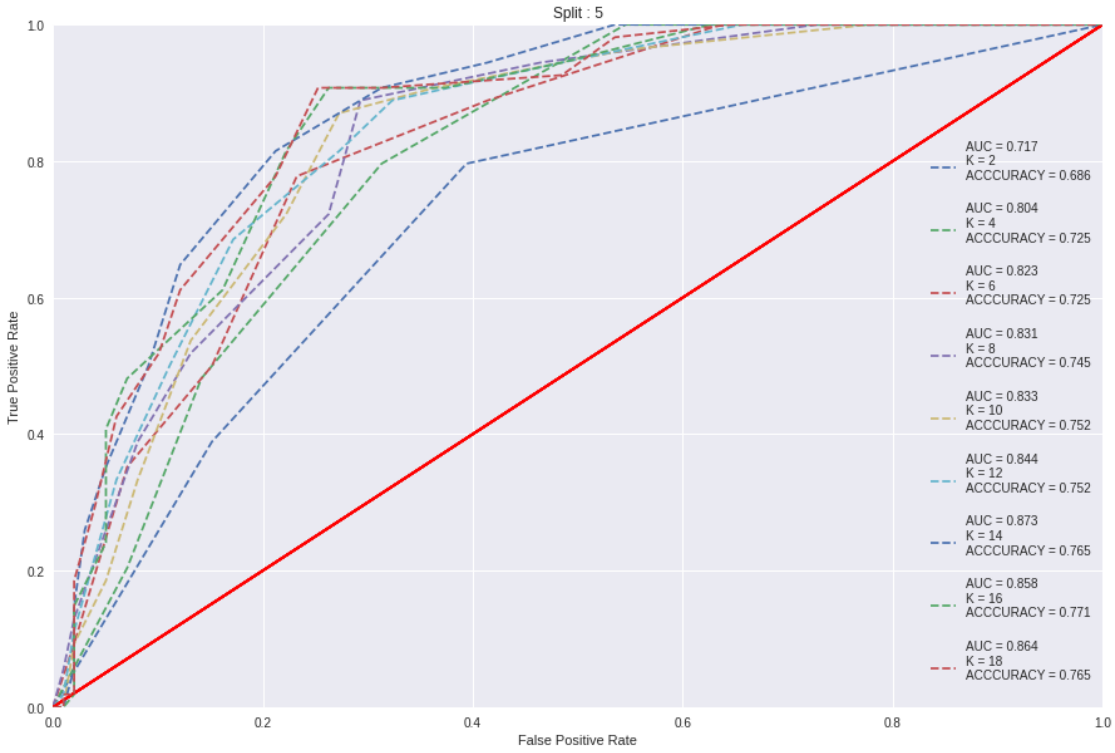
<Figure size 576x396 with 0 Axes>



<Figure size 576x396 with 0 Axes>



<Figure size 576x396 with 0 Axes>

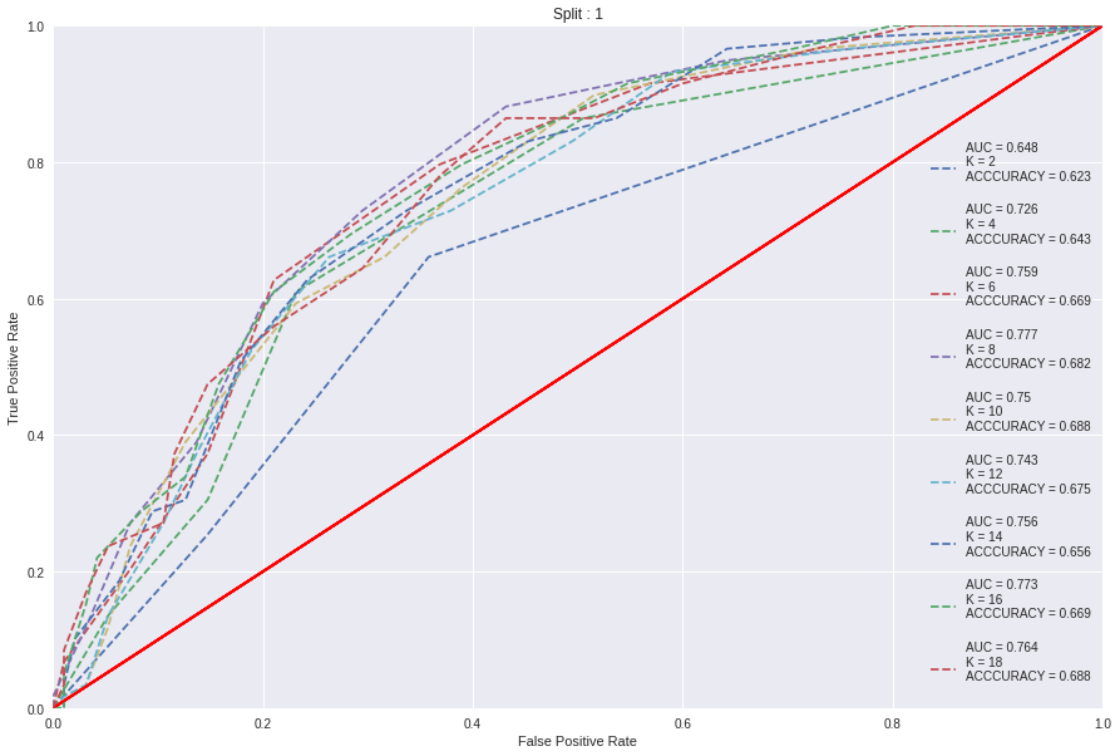


ROC Curve for Repeated K Fold

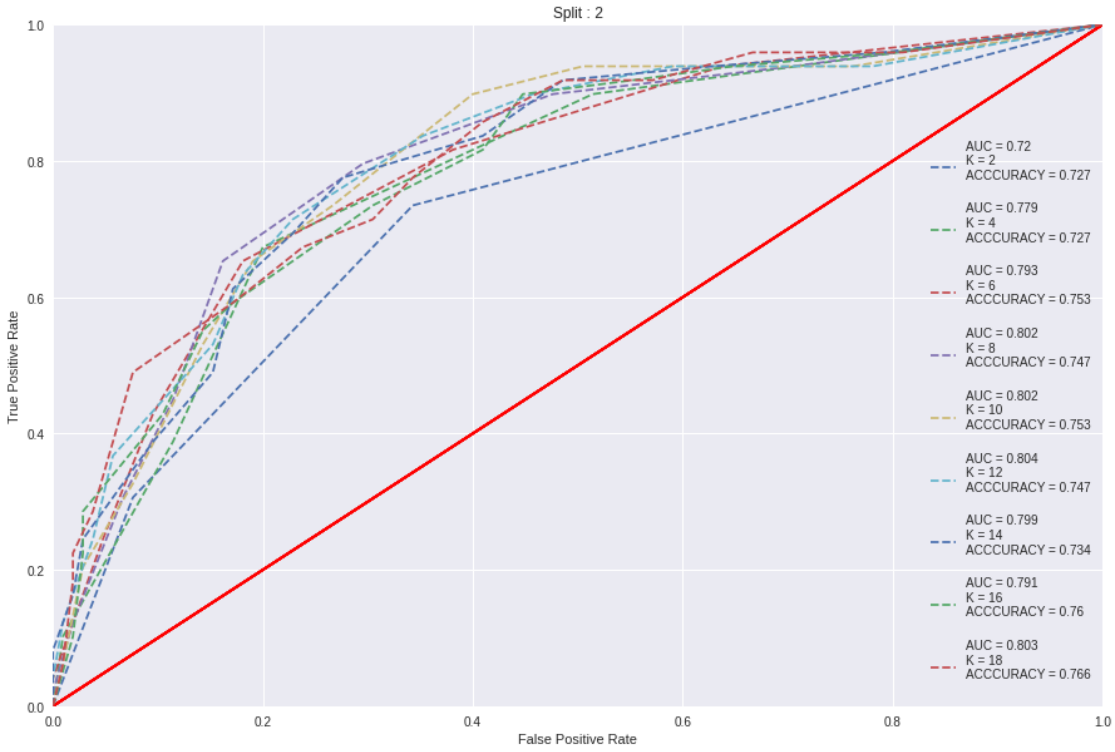
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import RepeatedKFold
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score

kf = RepeatedKFold(n_splits=5, n_repeats=10, random_state=None)

j = 1

for train_index, test_index in kf.split(X):

    #print("Train:", train_index, "\nValidation:",test_index)

    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]

    # Standardization
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Training the Model
    model = KNeighborsClassifier( n_neighbors = 5 )
    model.fit( X_train, Y_train.ravel() )

    plt.figure()
    plt.figure(figsize=(15,10))

    for i in range(2,20,2):

      # Training the Model
      model = KNeighborsClassifier( n_neighbors = i )
      model.fit(X_train, Y_train.ravel())

      # The function predict_proba() returns a numpy array of two columns.
      # The first column is the probability that target=0 and the second
column is the probability that target = 1

      Y_pred = model.predict_proba(X_test)

      # Select the probability estimates of the positive class
      fpr, tpr, threshold = roc_curve(Y_test, Y_pred[:, 1])

      roc_auc = roc_auc_score(Y_test, Y_pred[:, 1])

      # Predicting Test Data Set
      Y_pred = model.predict(X_test)
```

```
        plt.title('Split : {}'.format(j))
        plt.plot([0, 1], [0, 1],'r-')
        plt.xlim([0, 1])
        plt.ylim([0, 1])

        plt.ylabel('True Positive Rate')
        plt.xlabel('False Positive Rate')

        plt.plot(fpr, tpr, linestyle='--', label = 'AUC = {}\nK = {}\nACCCURACY
= {}\n'.format(roc_auc.round(3),i,accuracy_score(Y_test, Y_pred).round(3)))
        plt.legend()

    plt.show()
    j += 1
```

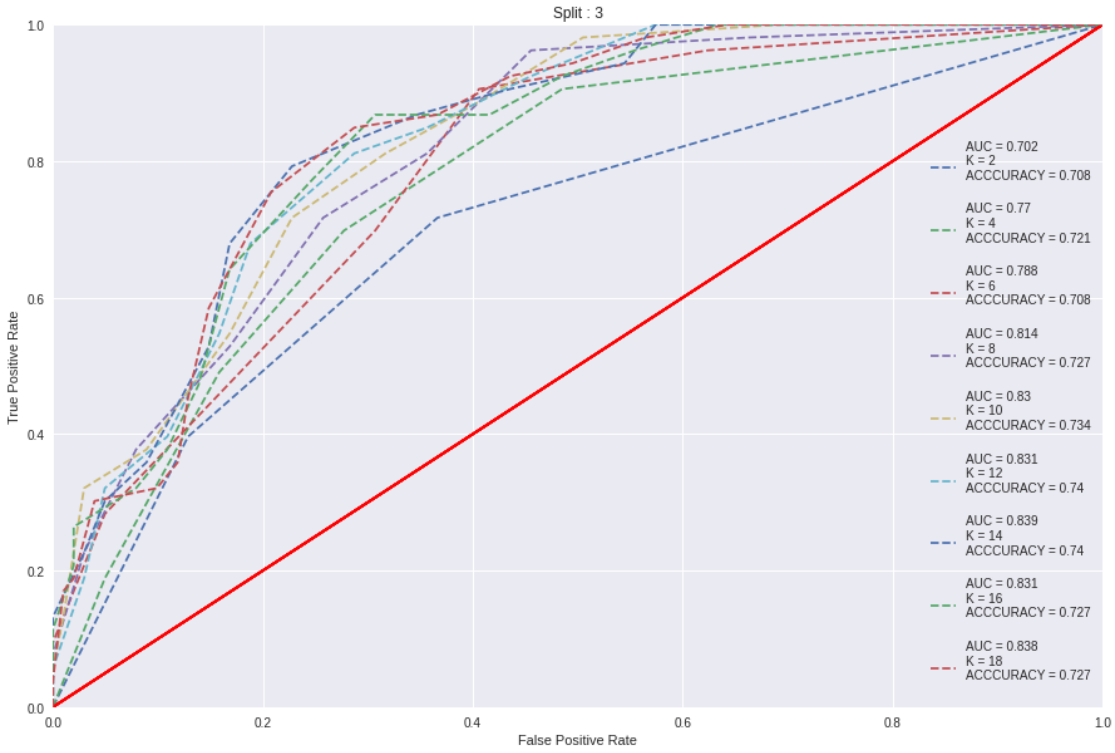
```
<Figure size 576x396 with 0 Axes>
```



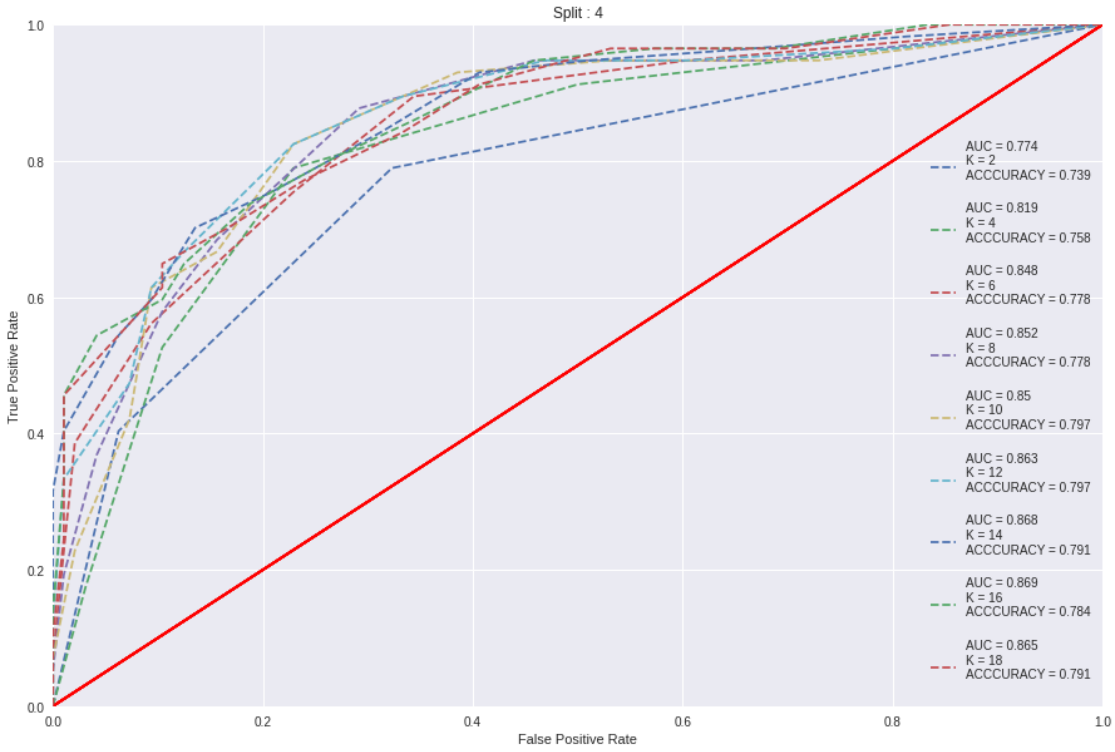

```
<Figure size 576x396 with 0 Axes>
```



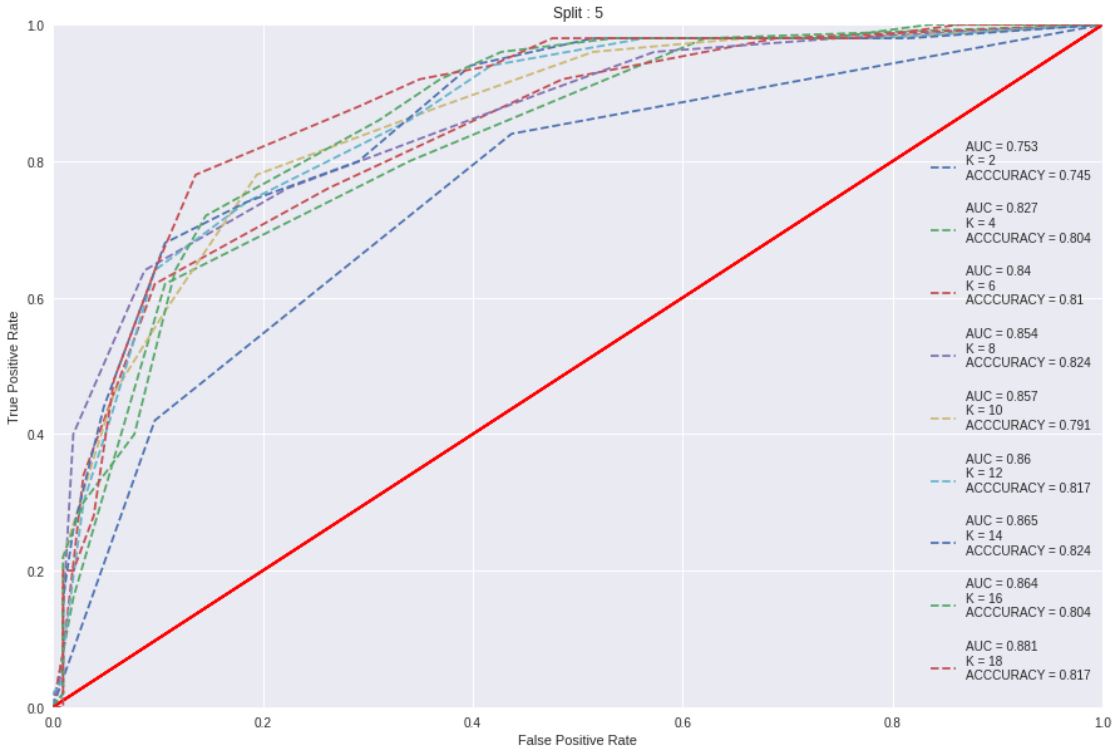
<Figure size 576x396 with 0 Axes>



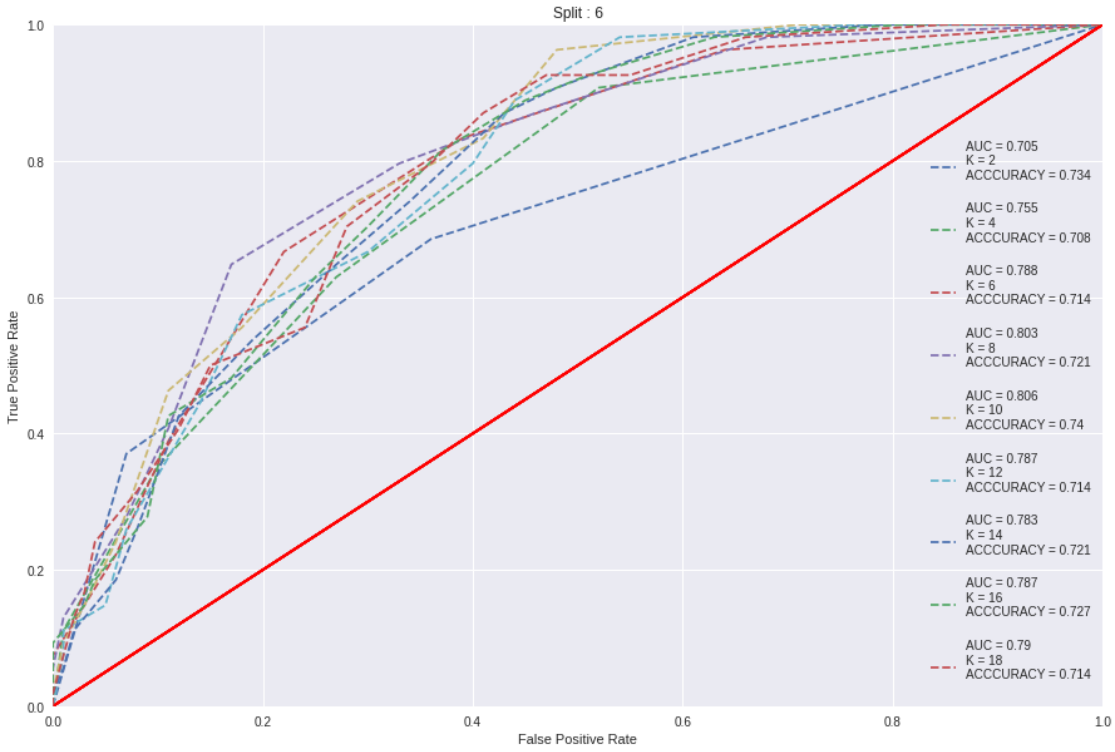
<Figure size 576x396 with 0 Axes>



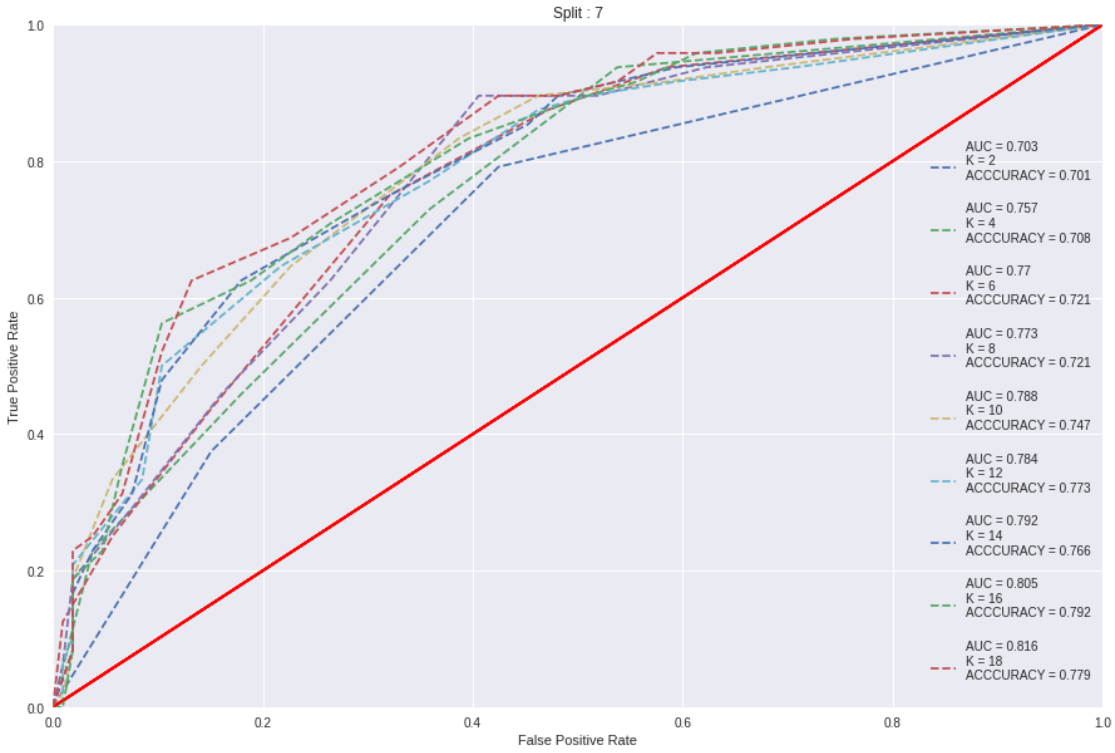
<Figure size 576x396 with 0 Axes>



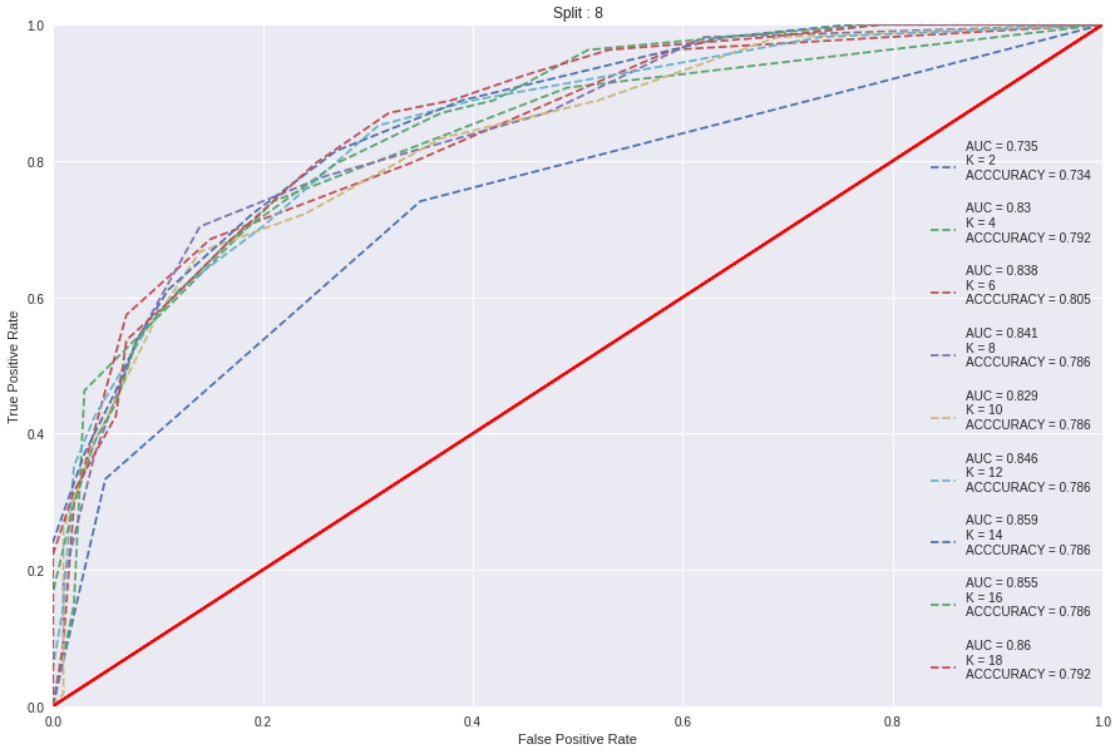
<Figure size 576x396 with 0 Axes>



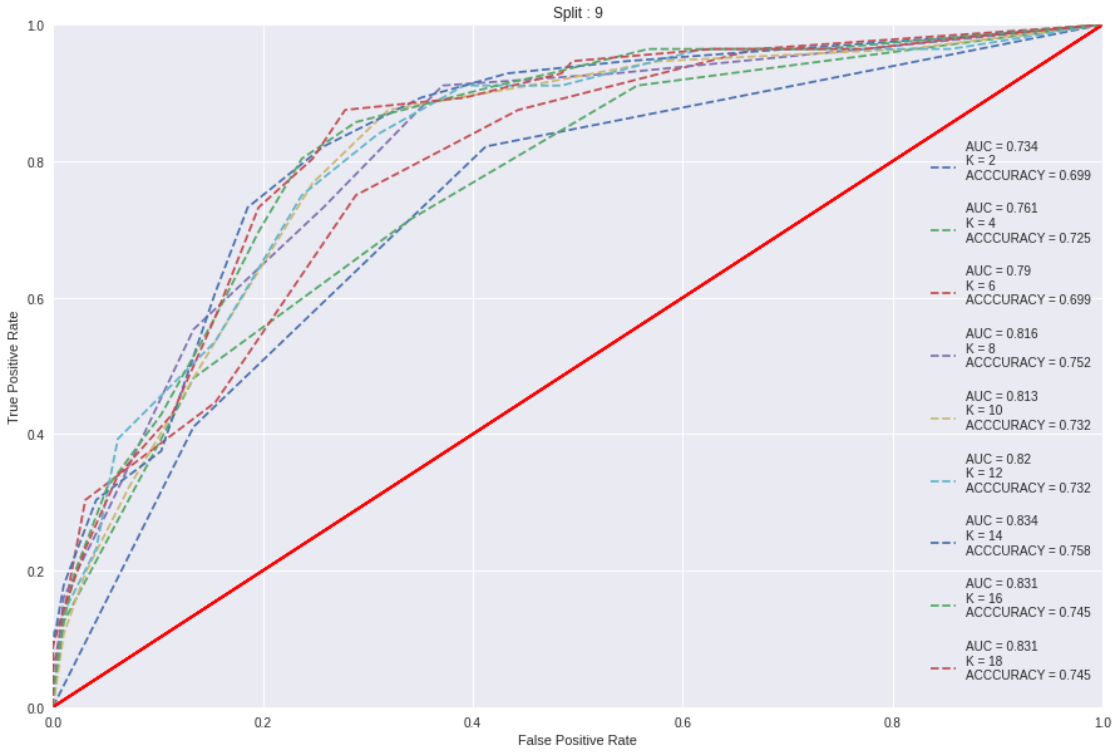
<Figure size 576x396 with 0 Axes>



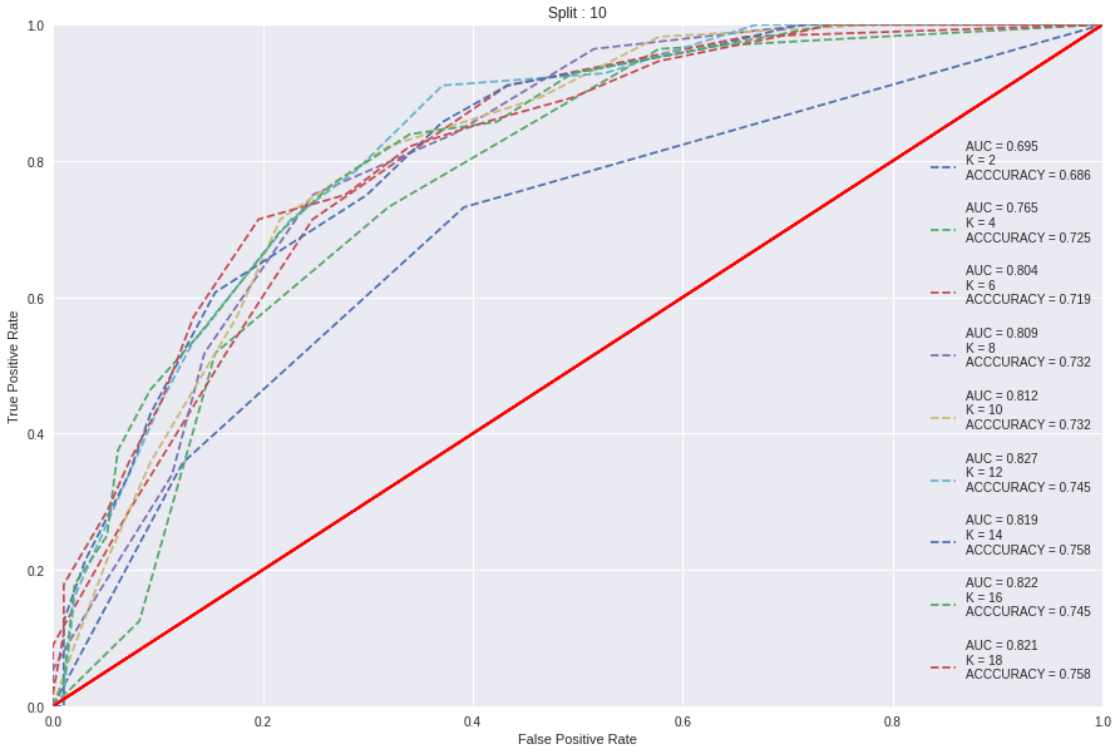
<Figure size 576x396 with 0 Axes>



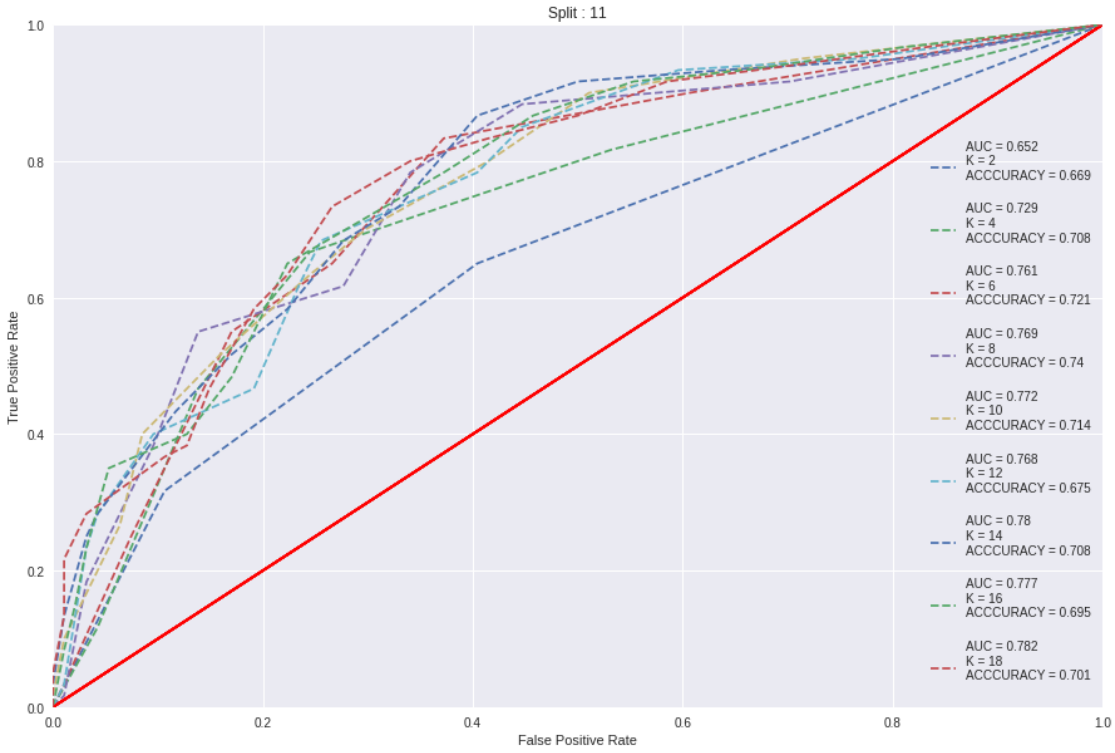
<Figure size 576x396 with 0 Axes>



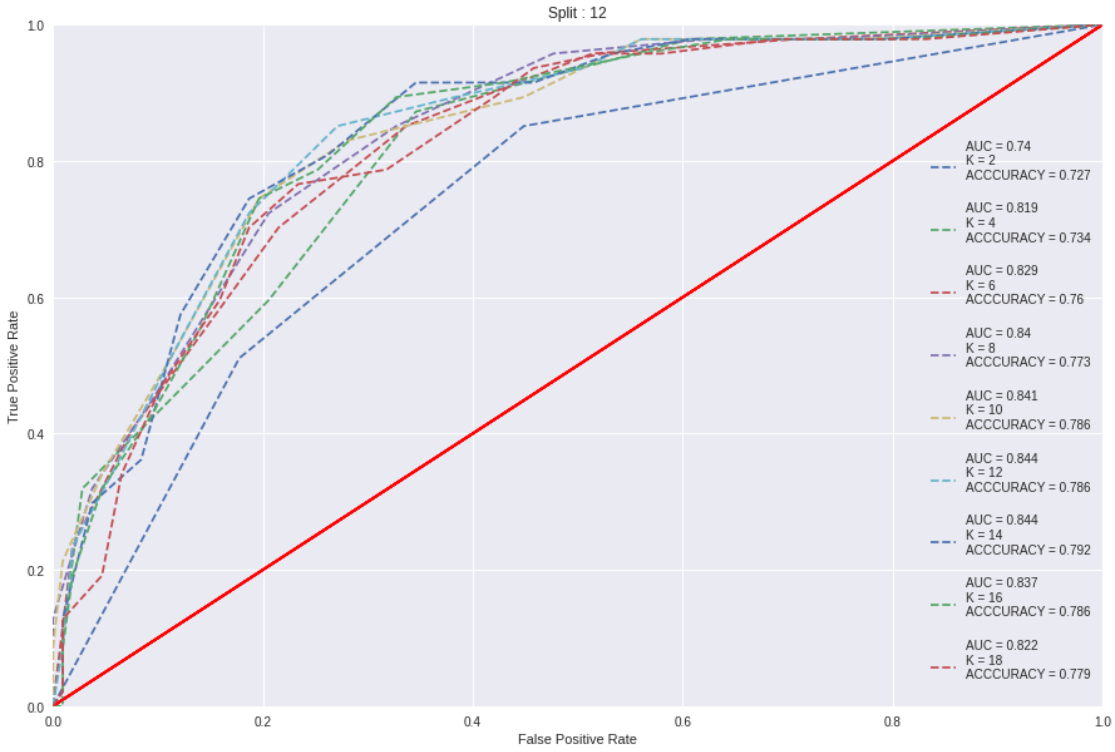
<Figure size 576x396 with 0 Axes>



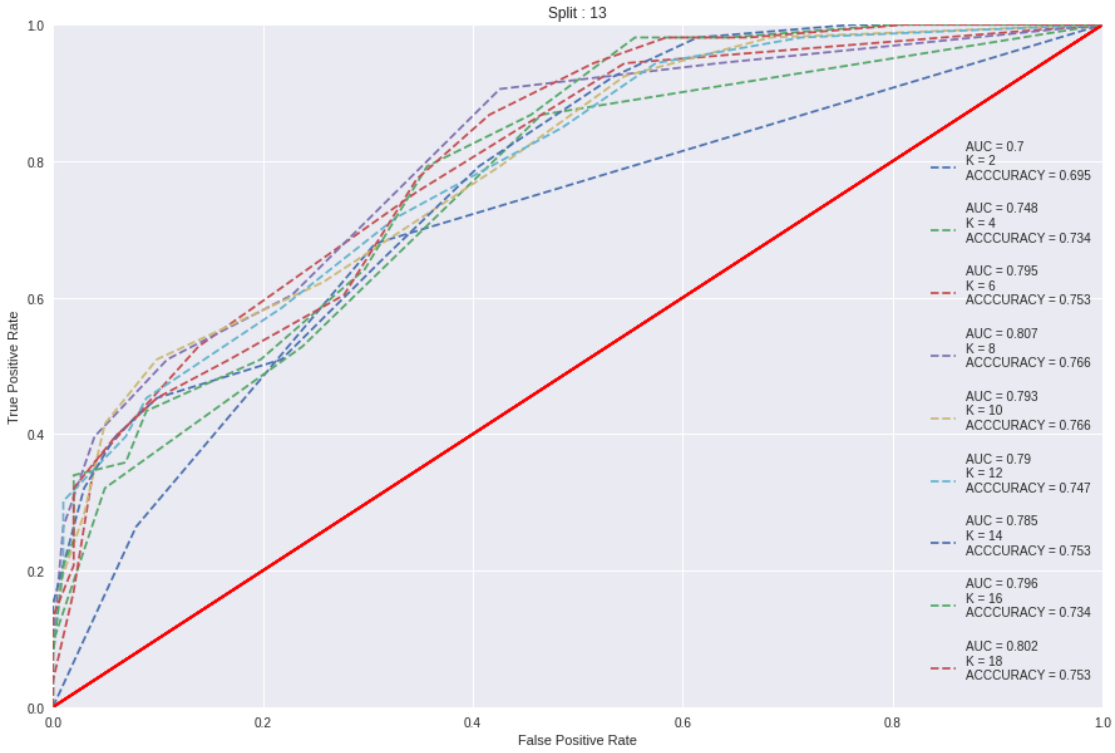
<Figure size 576x396 with 0 Axes>



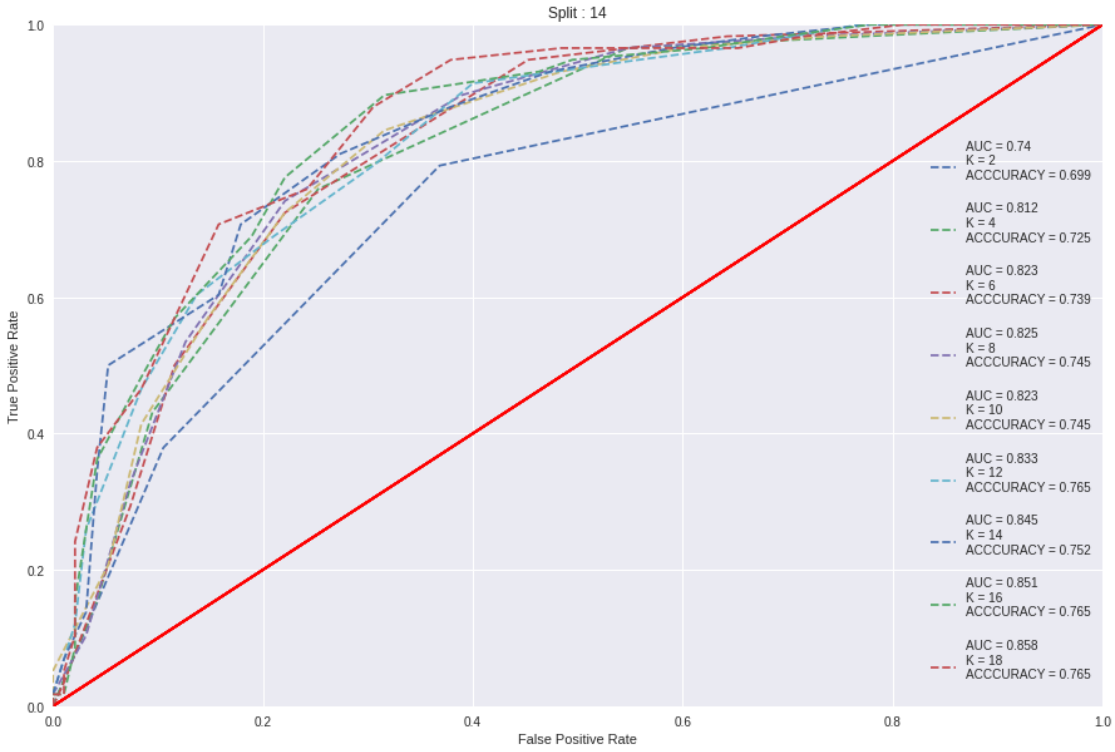
<Figure size 576x396 with 0 Axes>



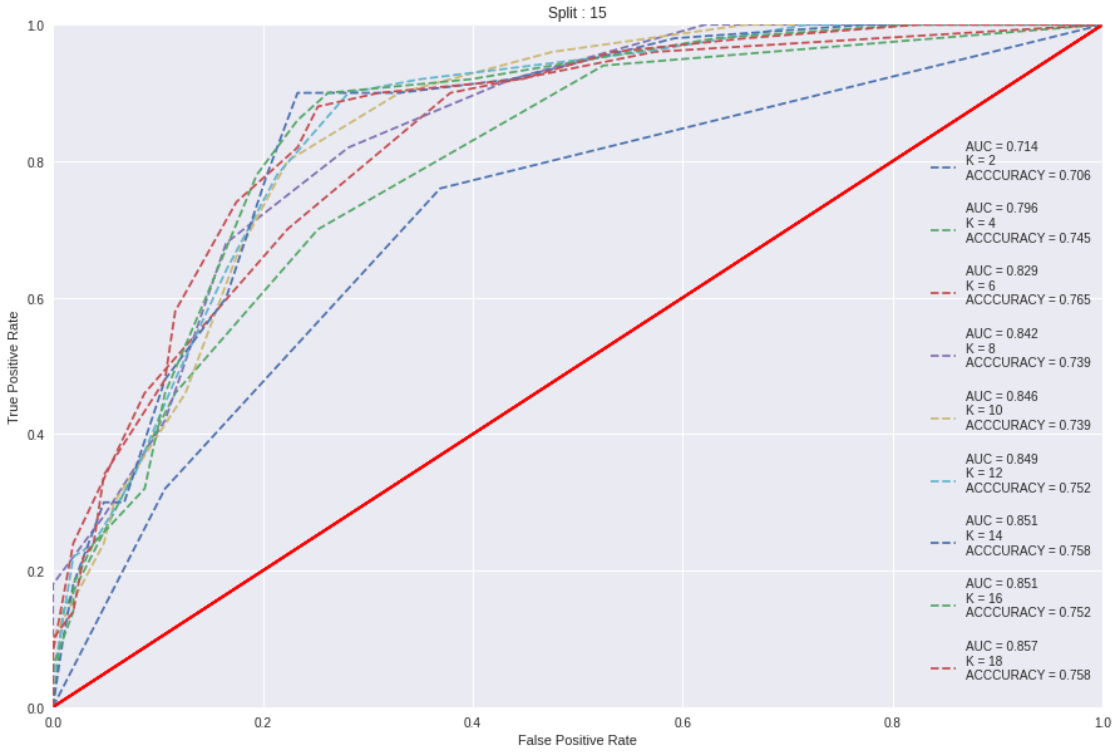
<Figure size 576x396 with 0 Axes>



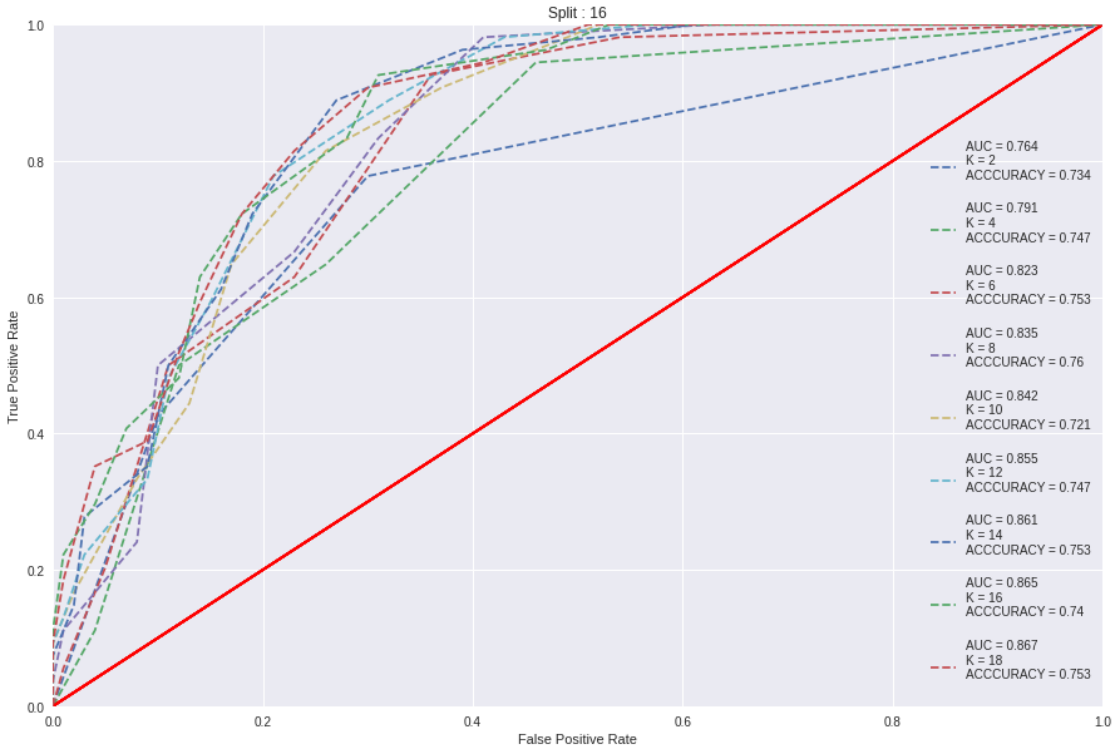
<Figure size 576x396 with 0 Axes>



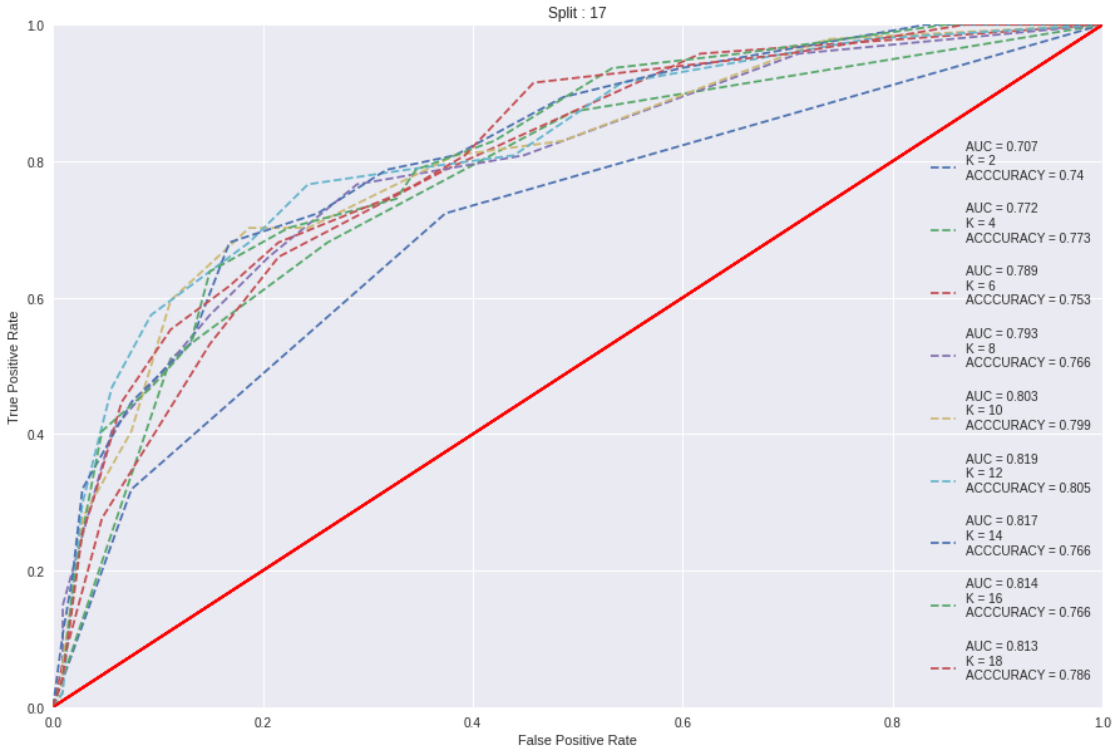
<Figure size 576x396 with 0 Axes>



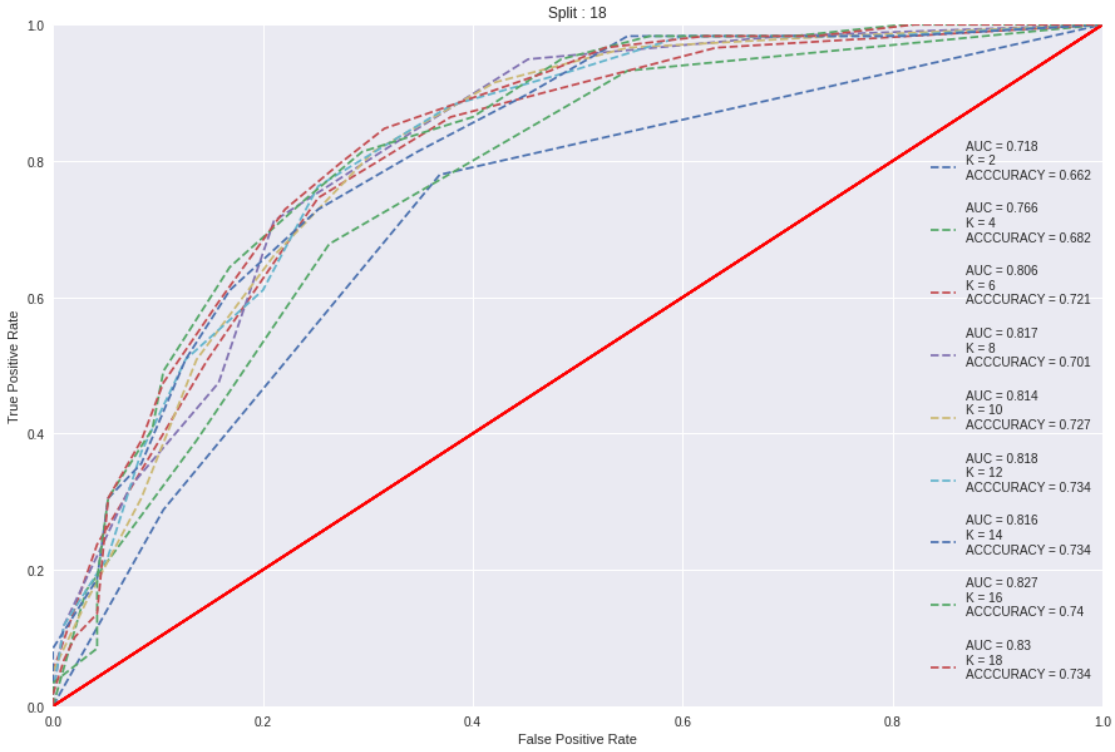
<Figure size 576x396 with 0 Axes>



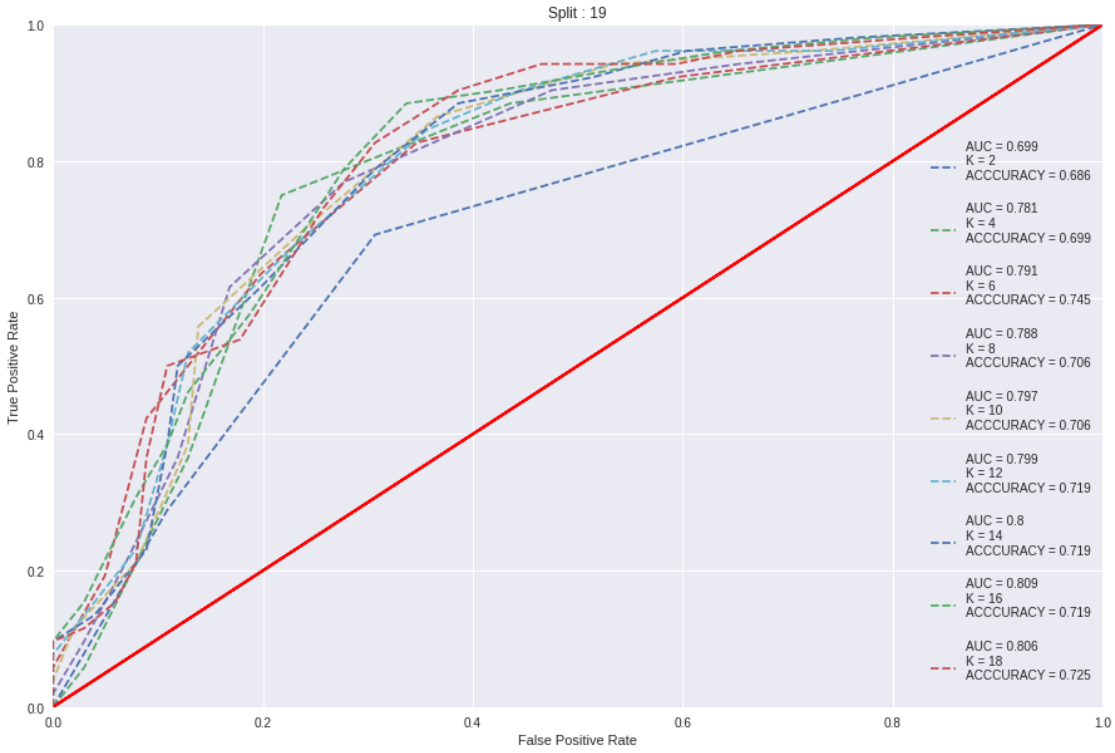
<Figure size 576x396 with 0 Axes>



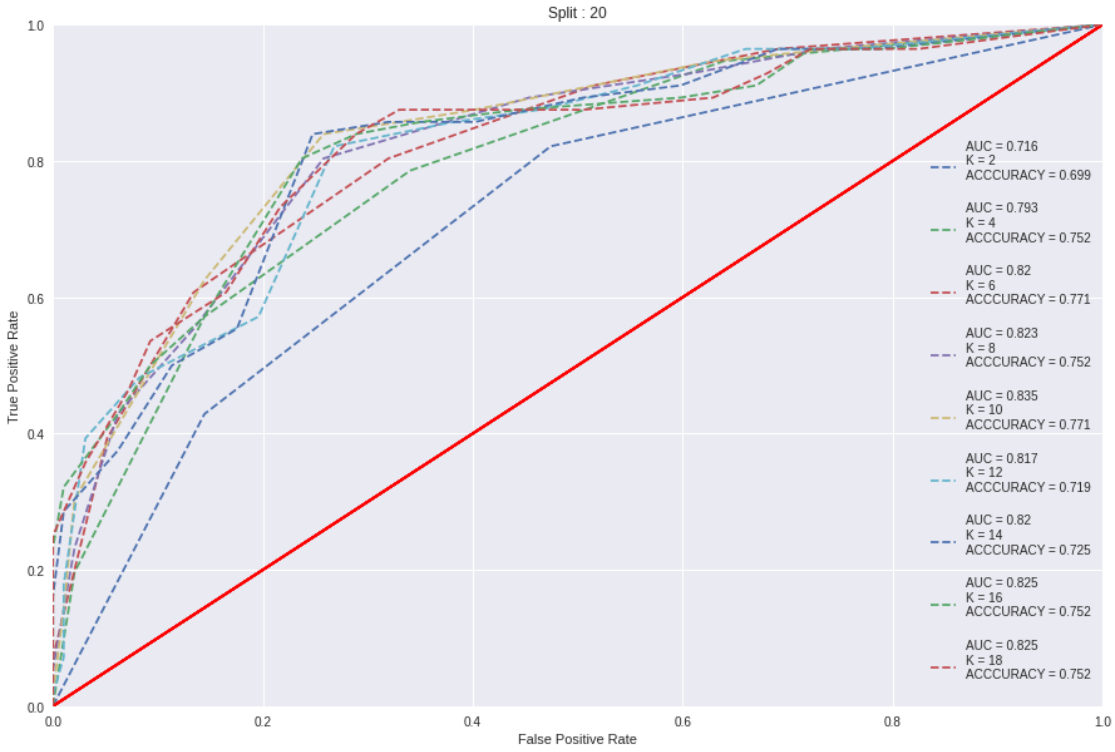
<Figure size 576x396 with 0 Axes>



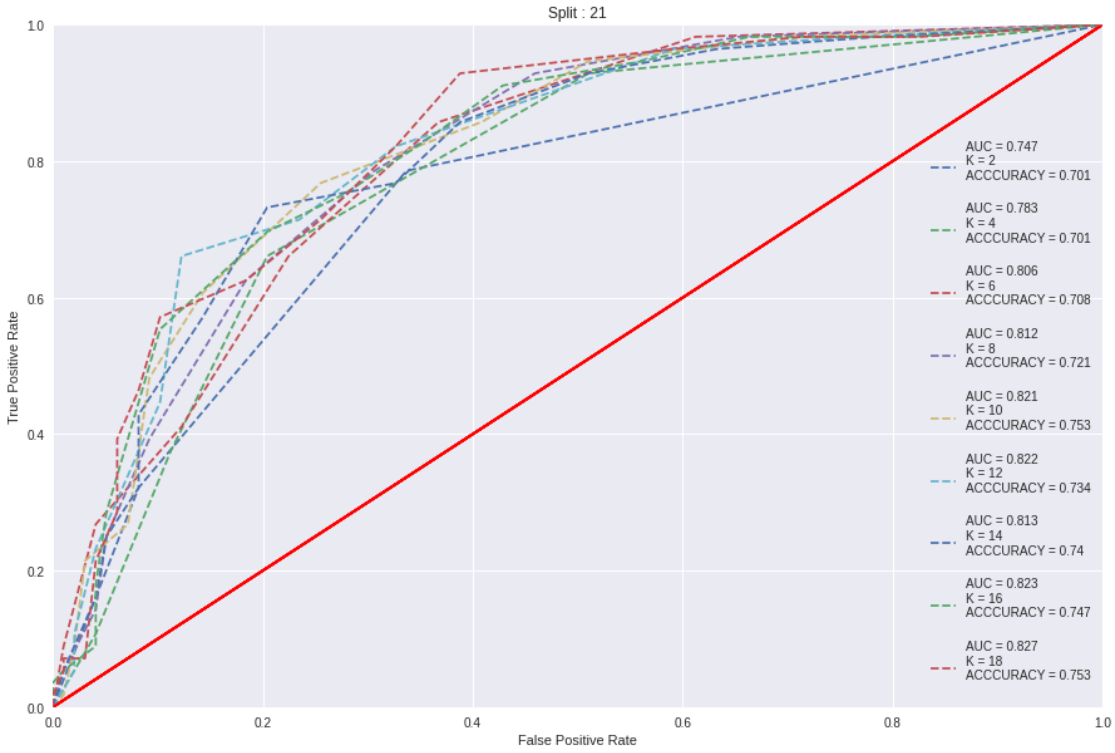
<Figure size 576x396 with 0 Axes>



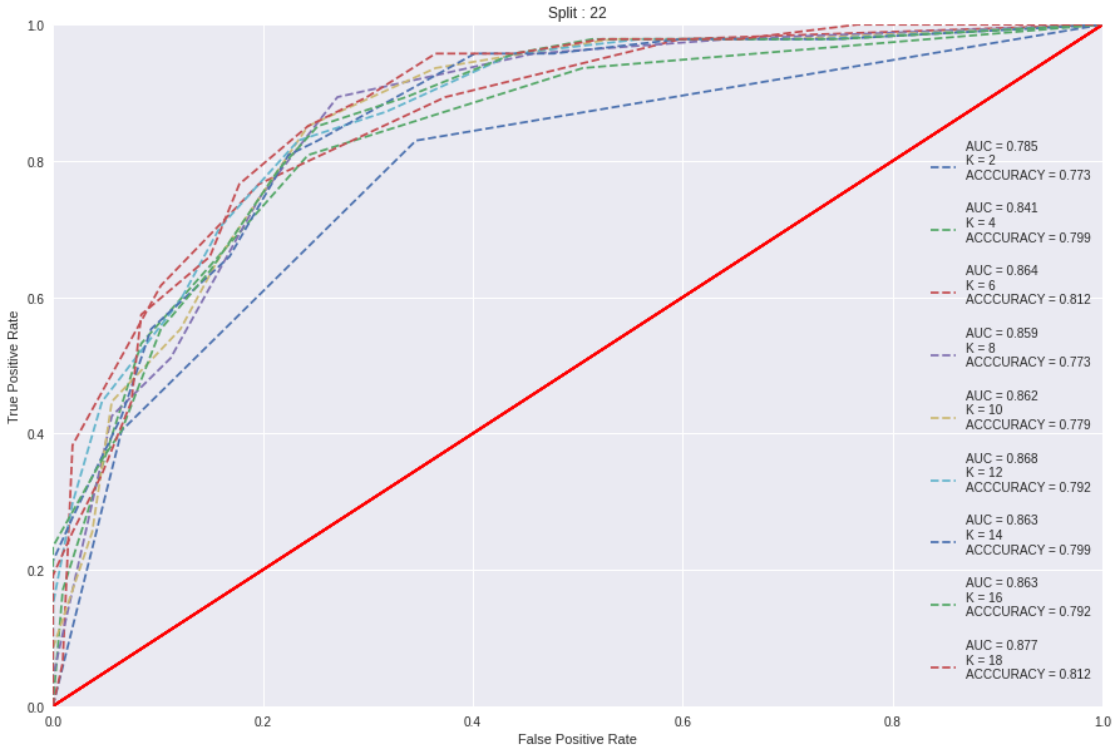
<Figure size 576x396 with 0 Axes>



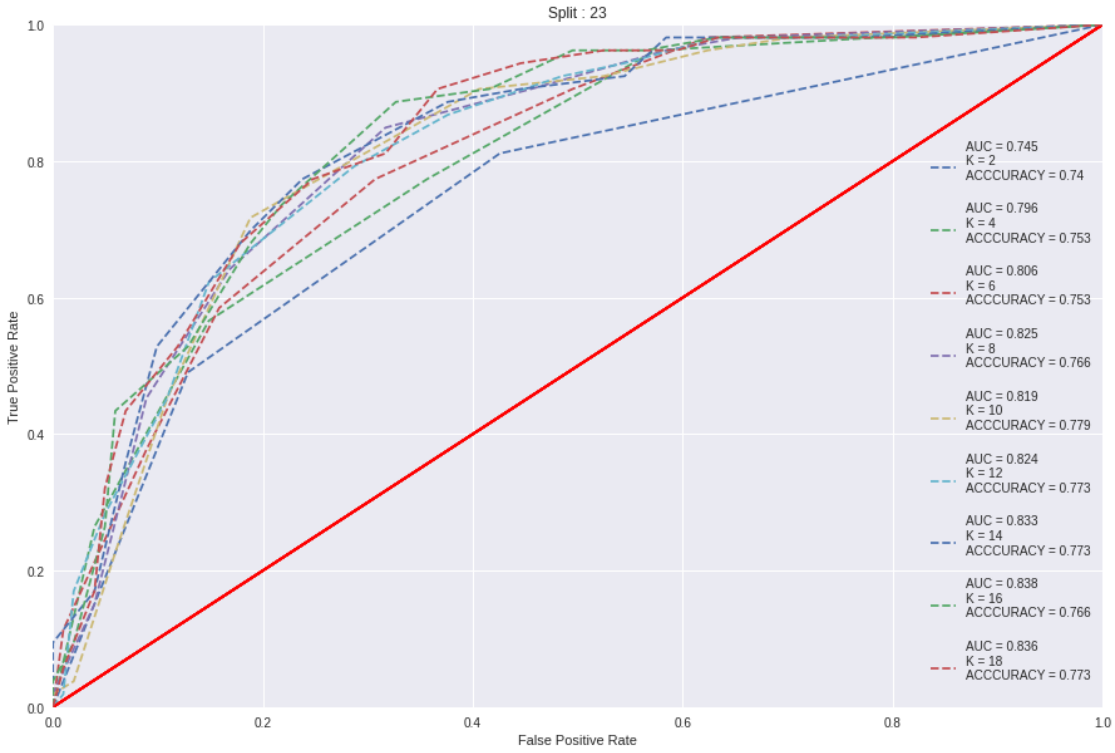
<Figure size 576x396 with 0 Axes>



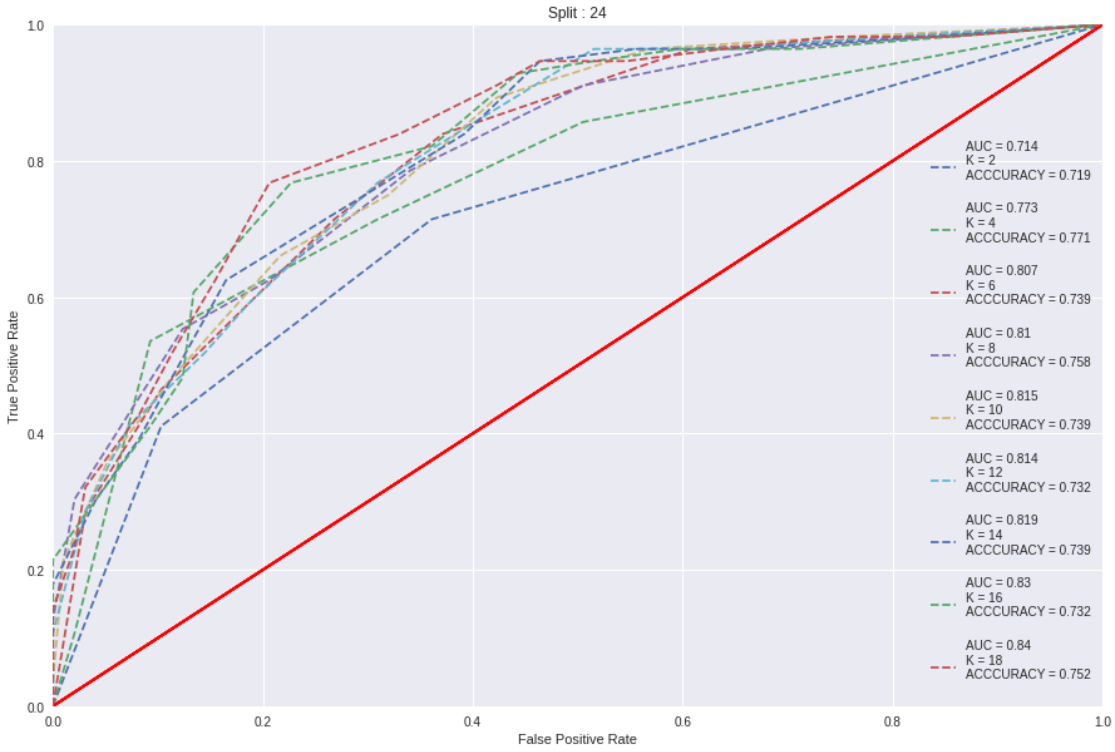
<Figure size 576x396 with 0 Axes>



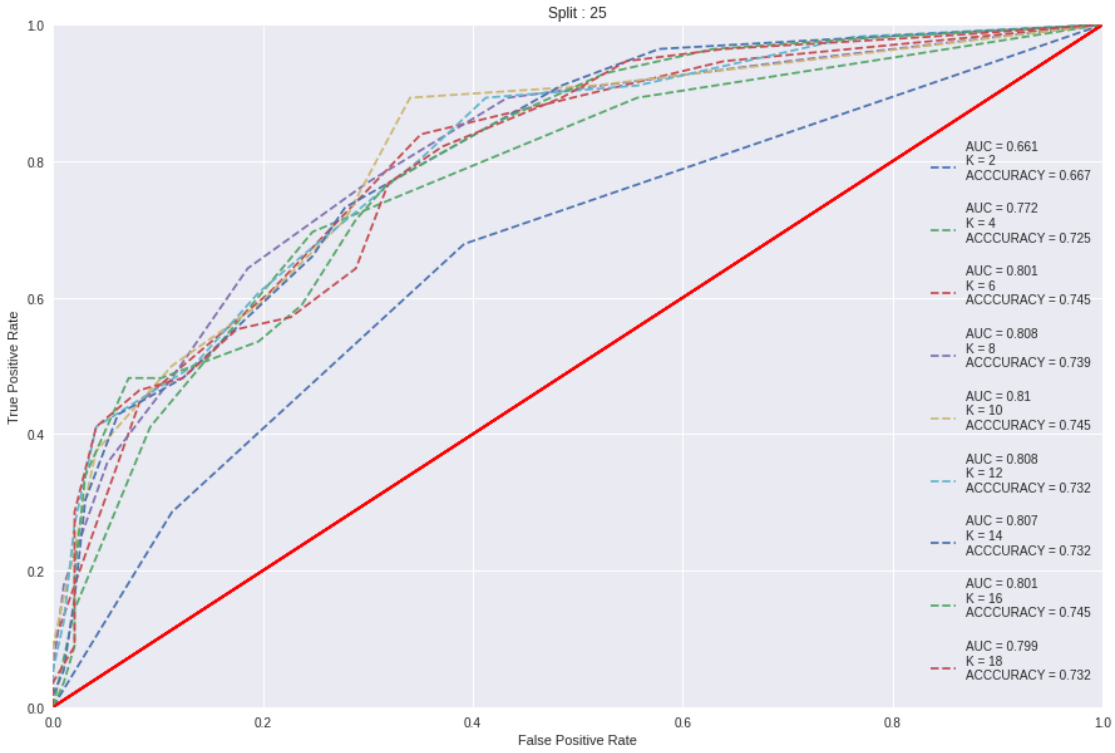
<Figure size 576x396 with 0 Axes>



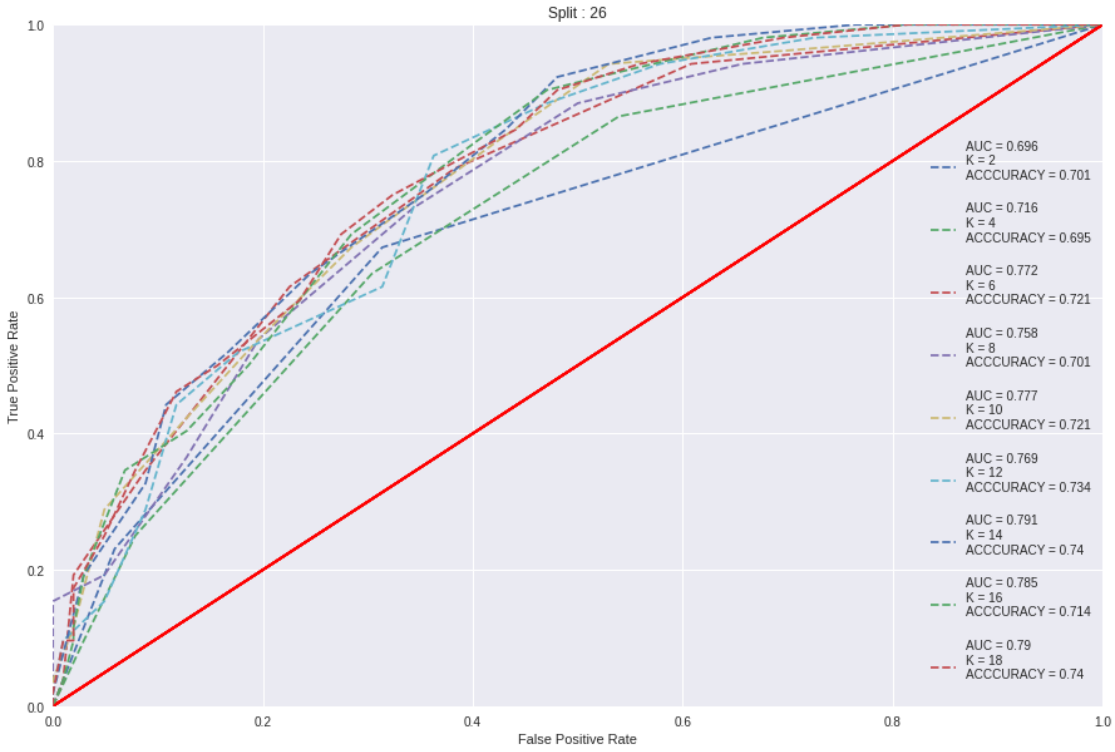
<Figure size 576x396 with 0 Axes>



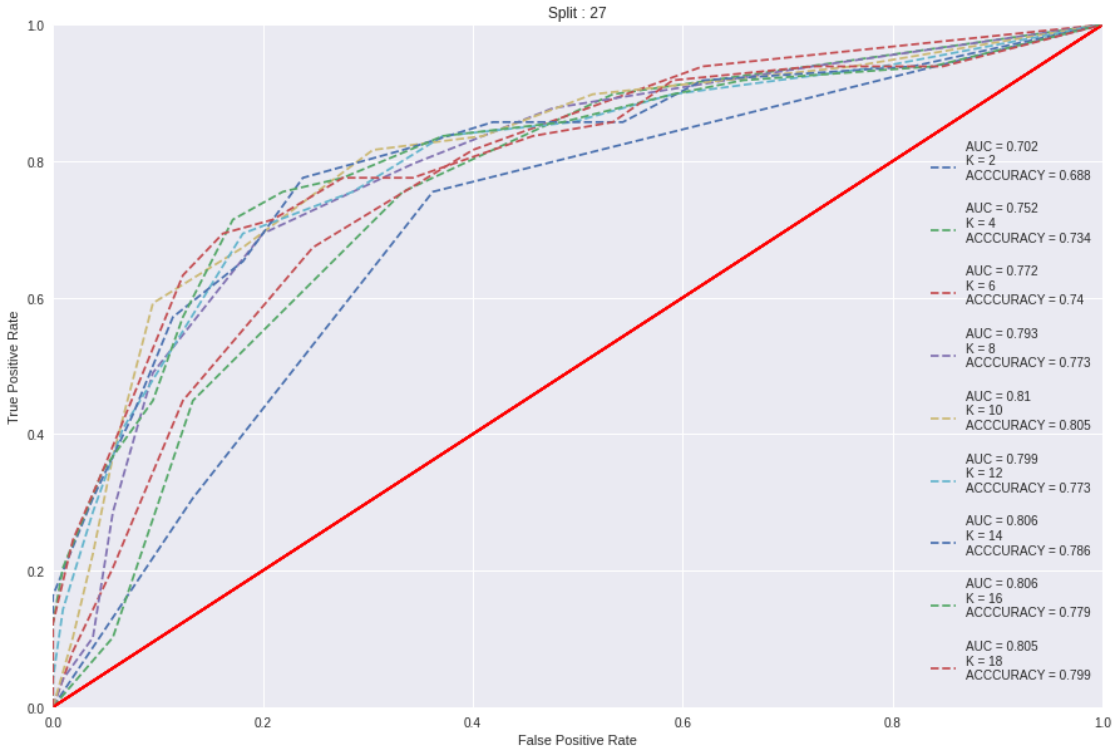
<Figure size 576x396 with 0 Axes>



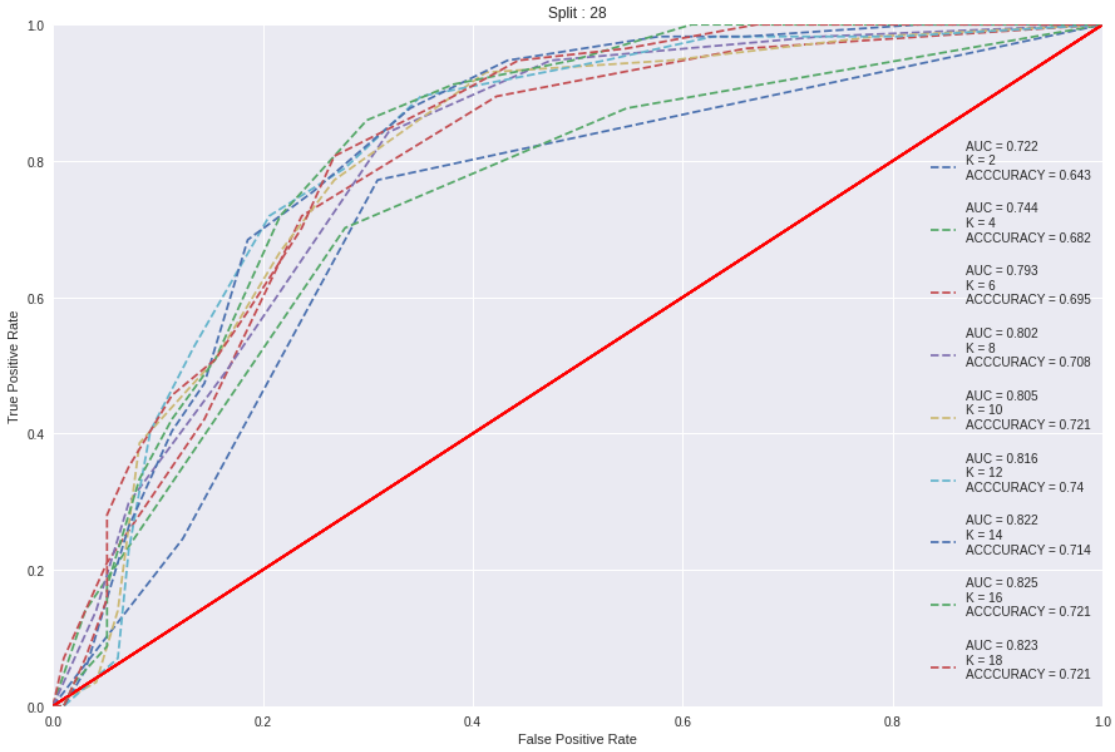
<Figure size 576x396 with 0 Axes>



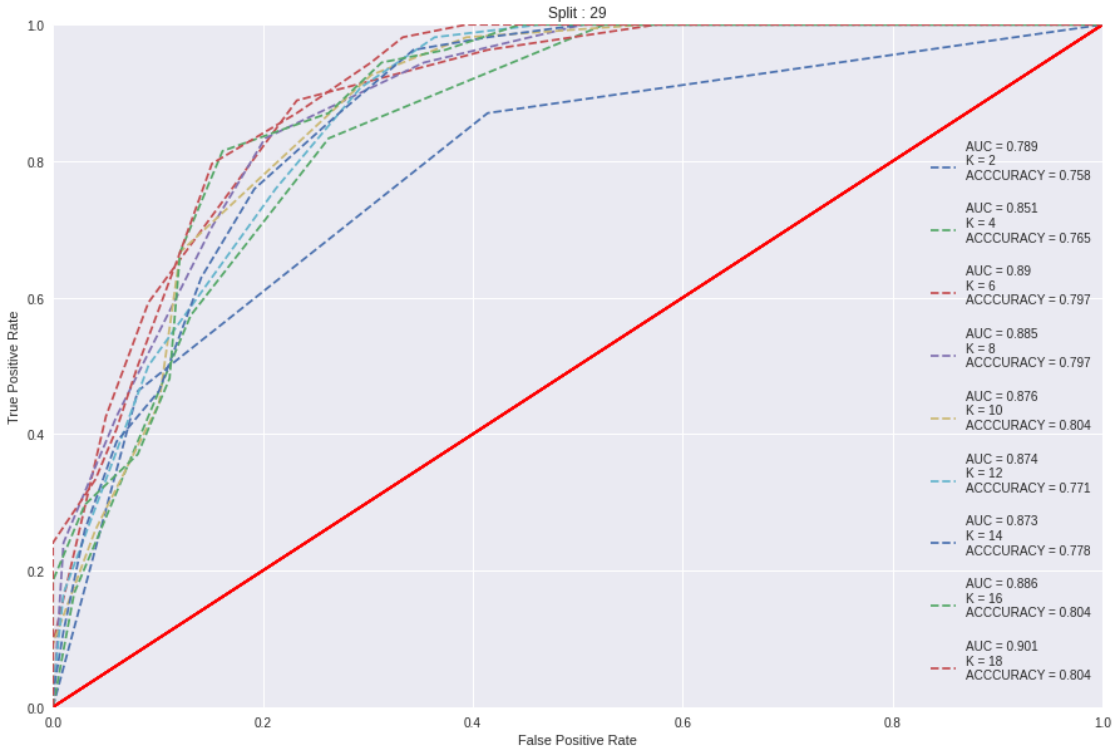
<Figure size 576x396 with 0 Axes>



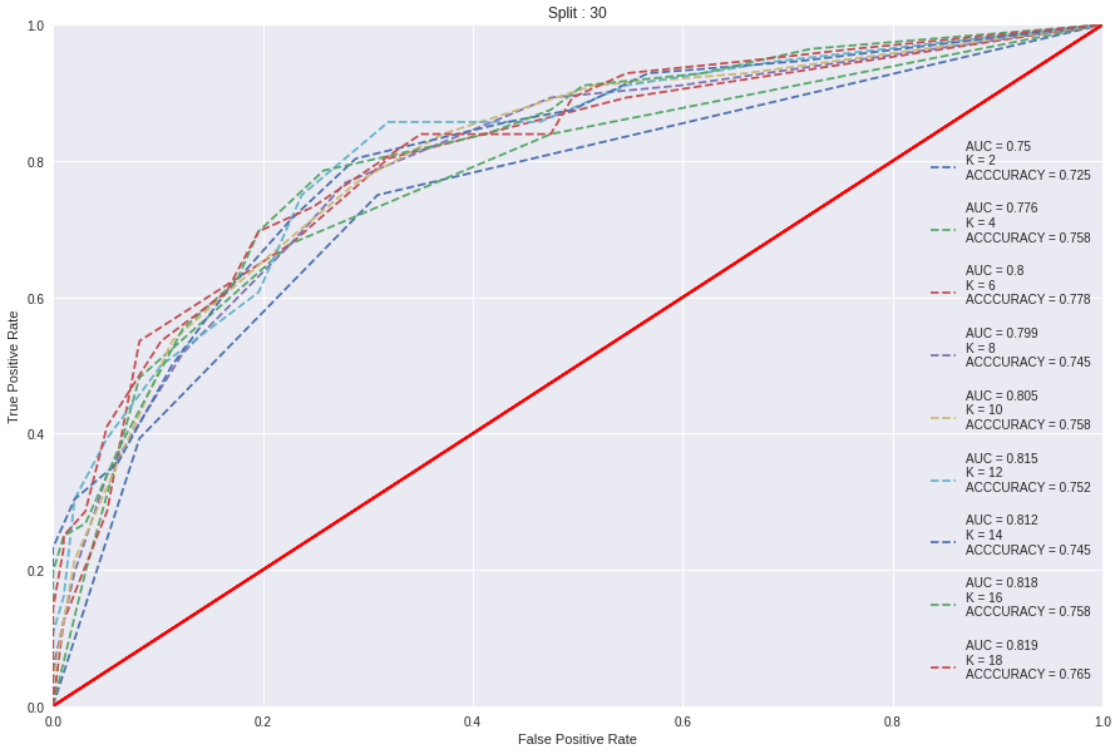
<Figure size 576x396 with 0 Axes>



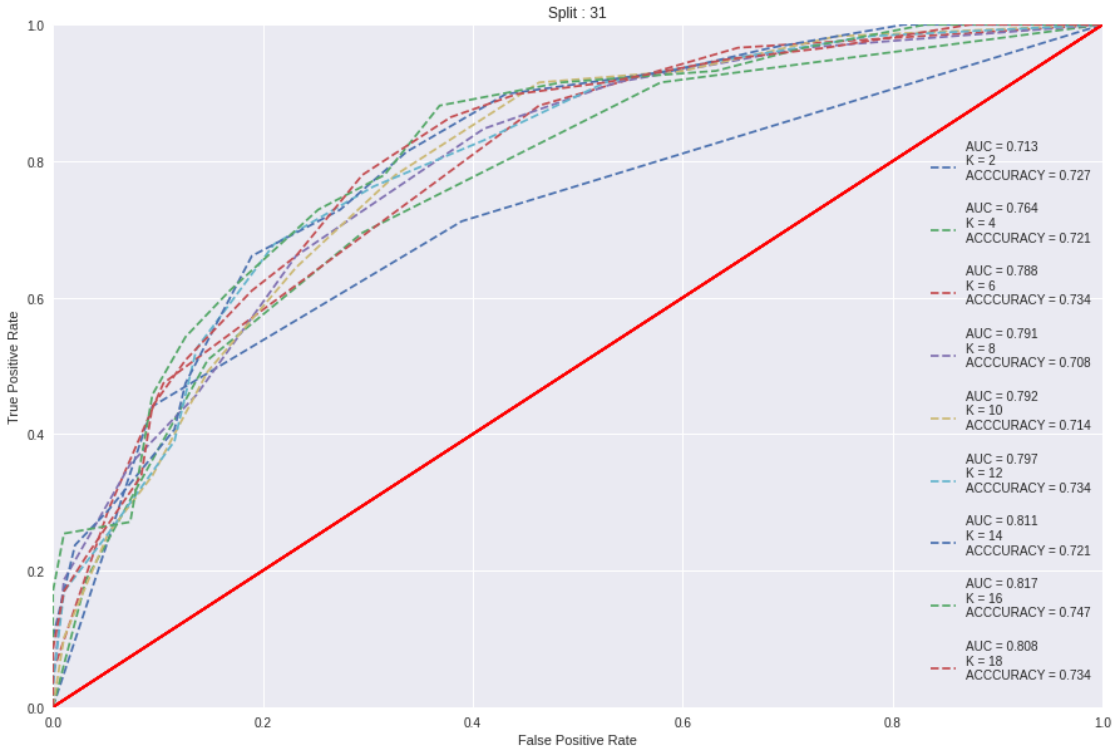
<Figure size 576x396 with 0 Axes>



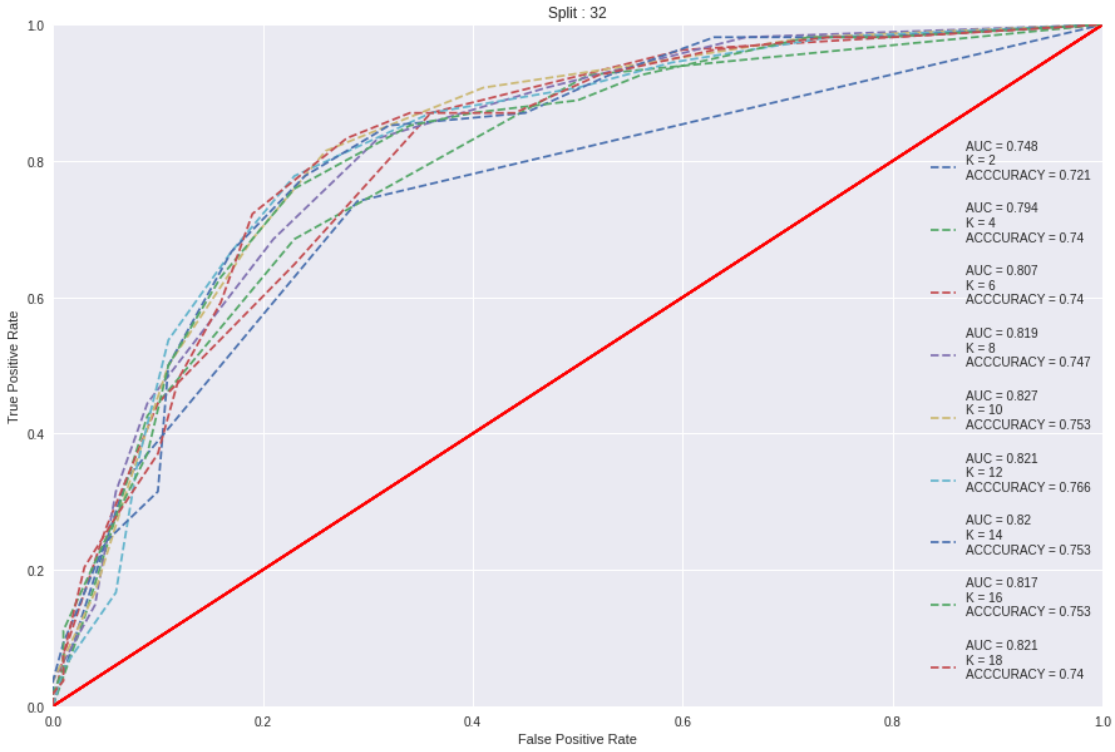
<Figure size 576x396 with 0 Axes>



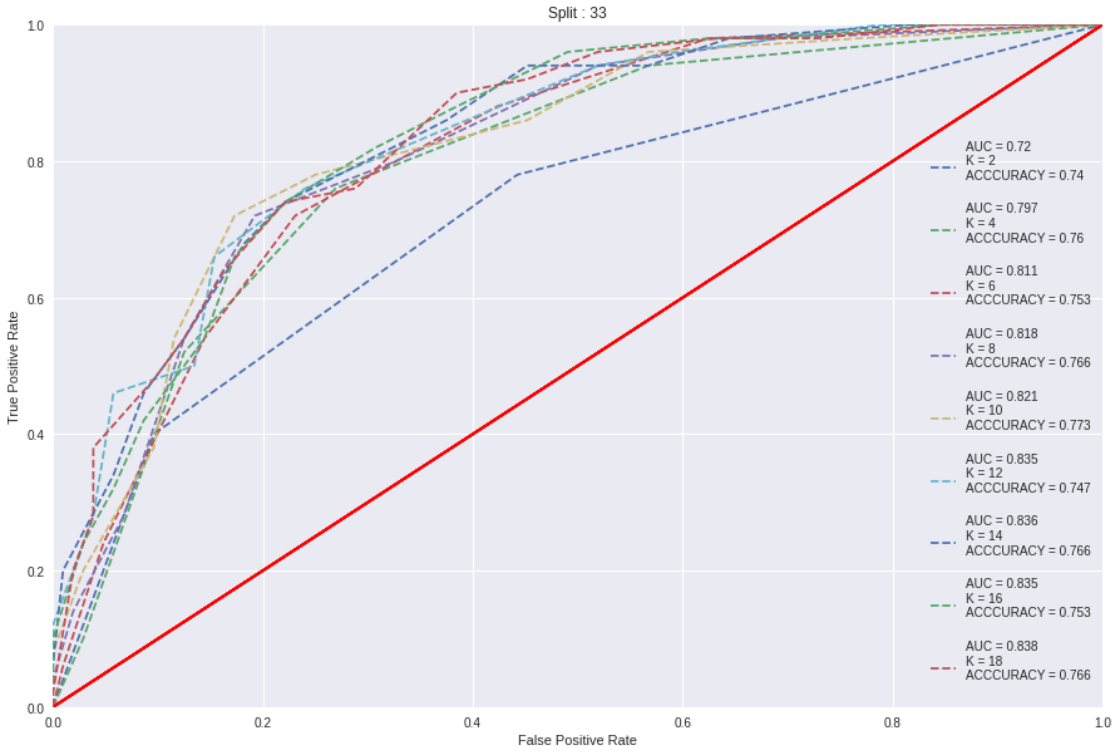
<Figure size 576x396 with 0 Axes>



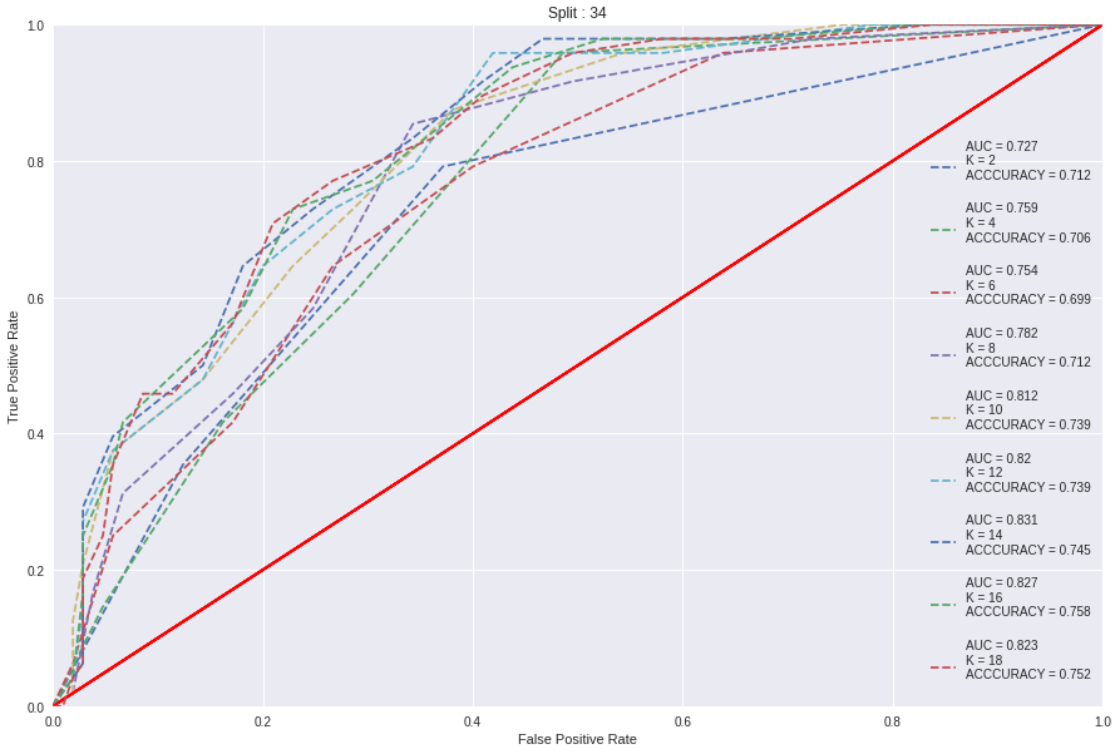
<Figure size 576x396 with 0 Axes>



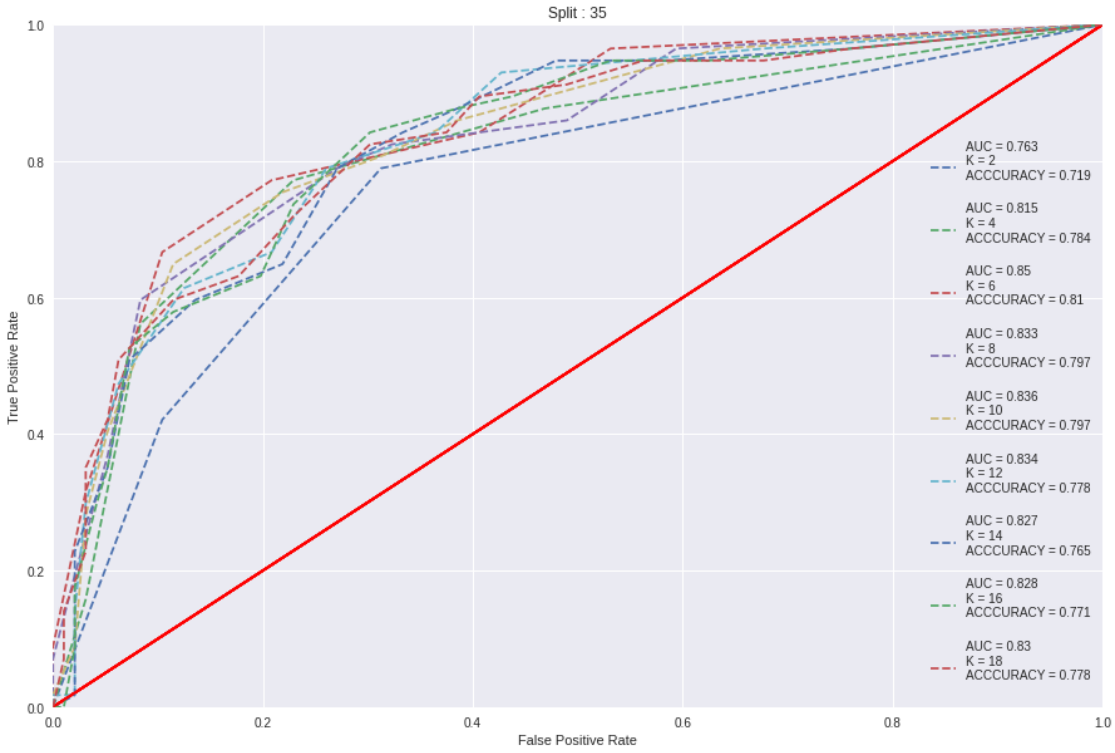
<Figure size 576x396 with 0 Axes>



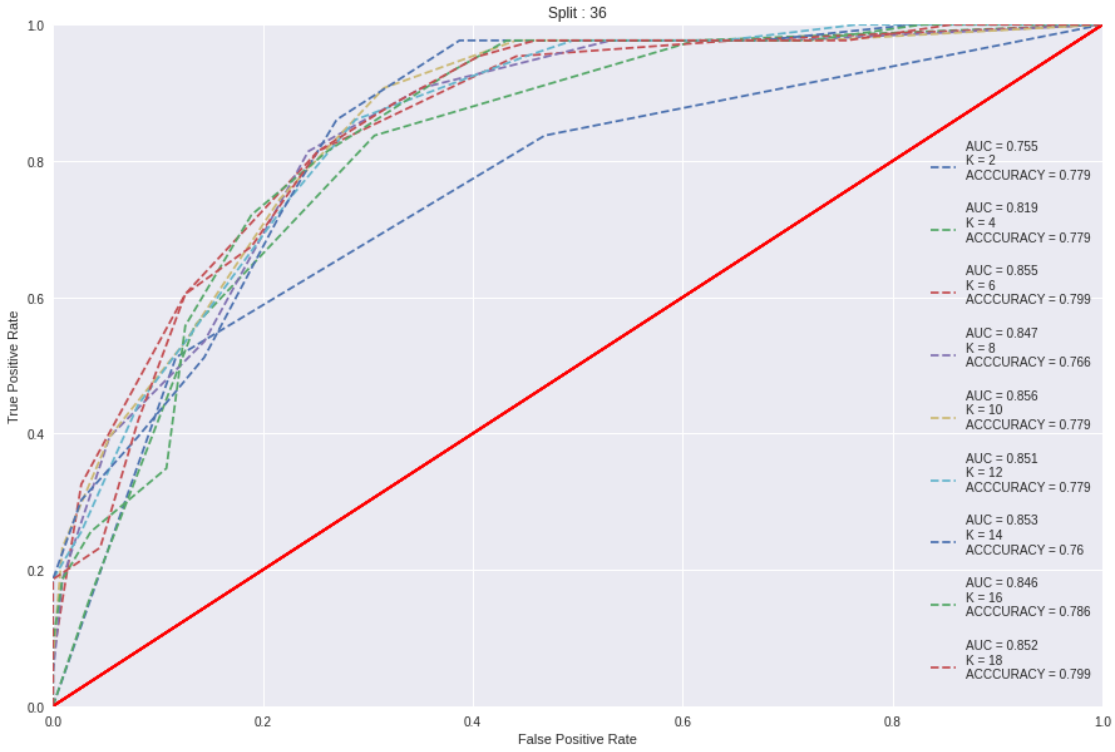
<Figure size 576x396 with 0 Axes>



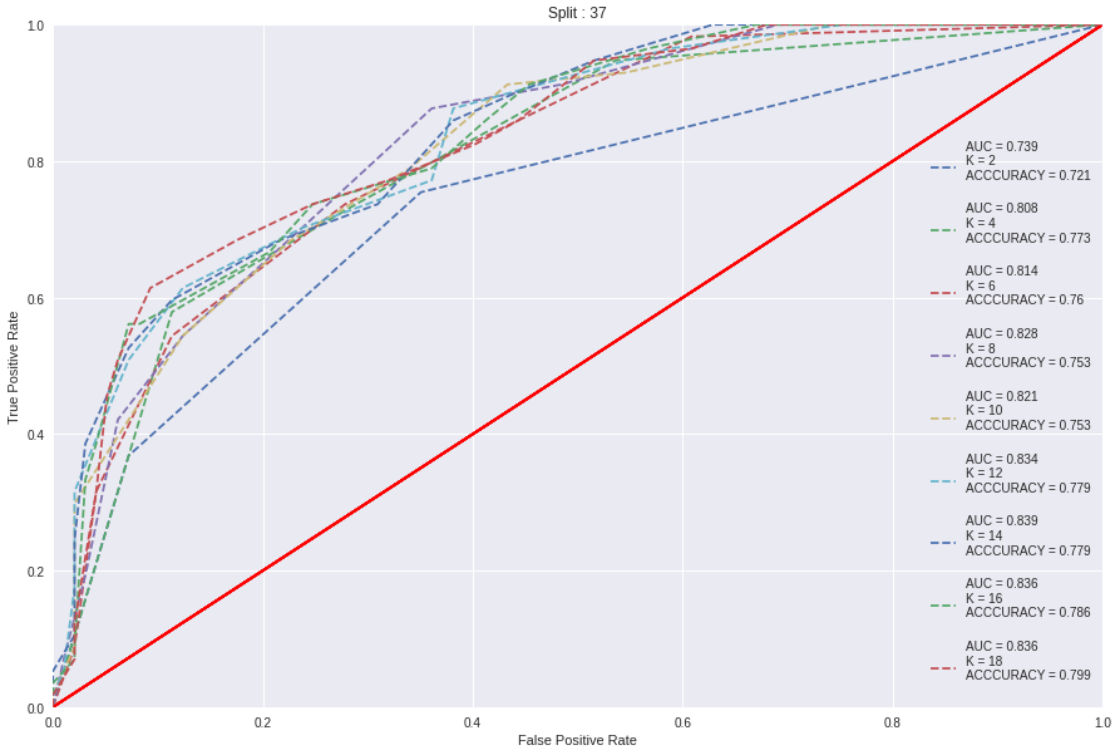
<Figure size 576x396 with 0 Axes>



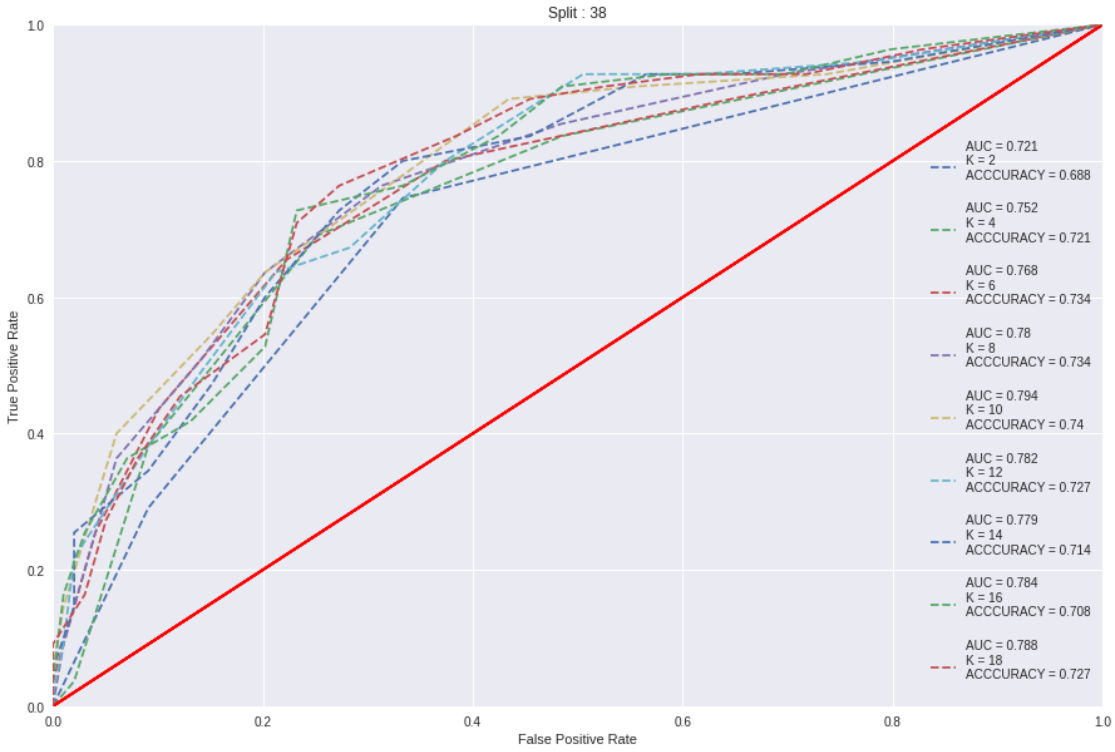
<Figure size 576x396 with 0 Axes>



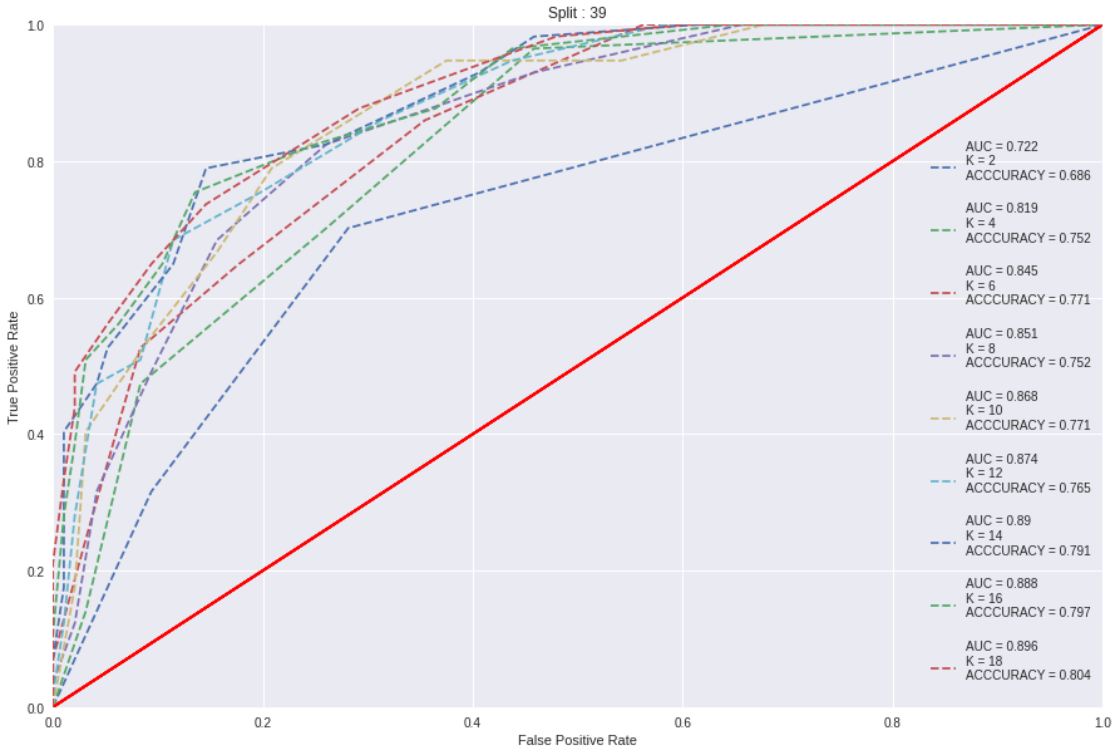
<Figure size 576x396 with 0 Axes>



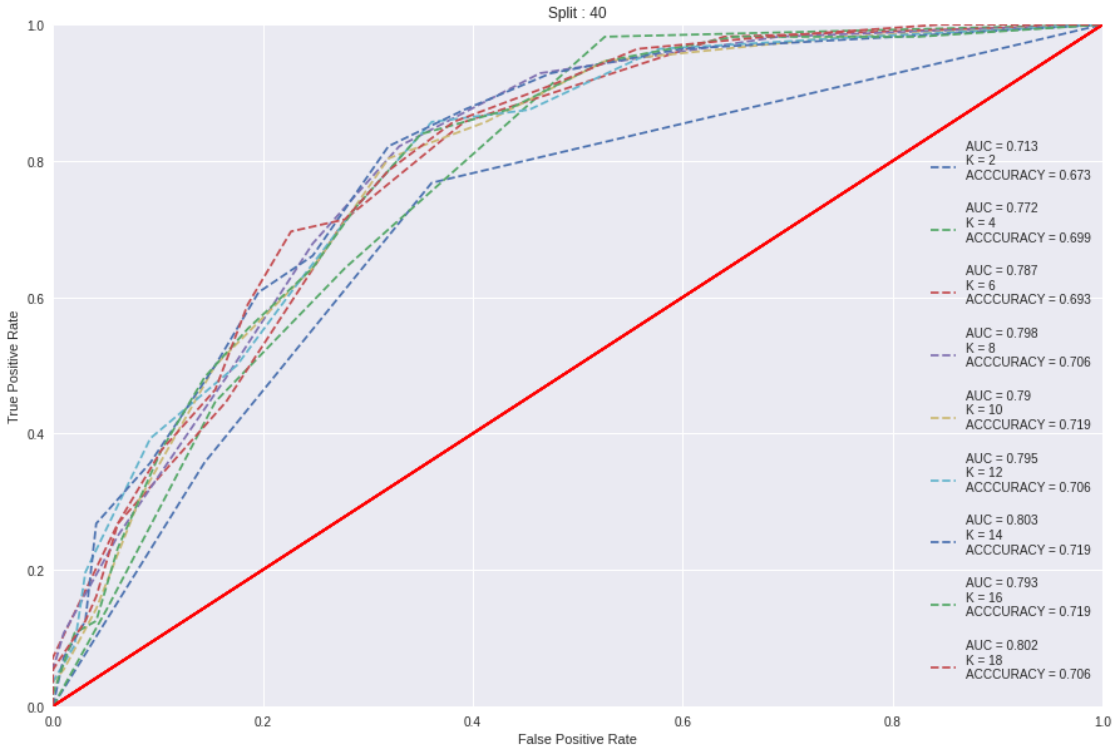
<Figure size 576x396 with 0 Axes>



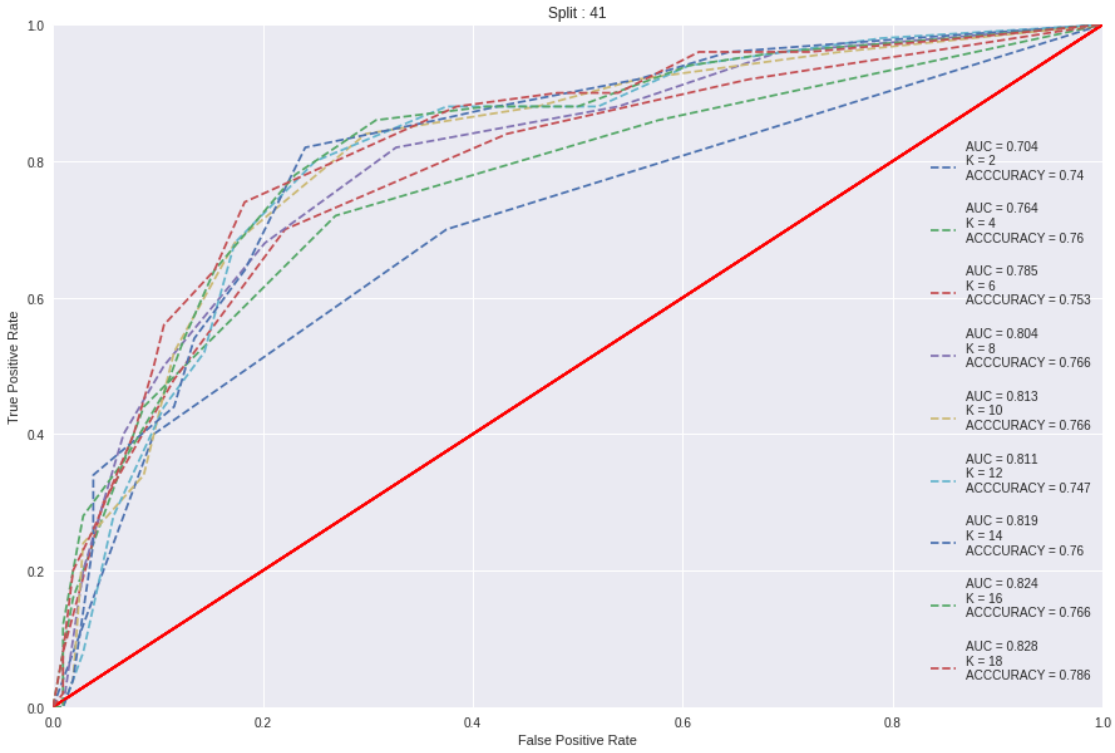
<Figure size 576x396 with 0 Axes>



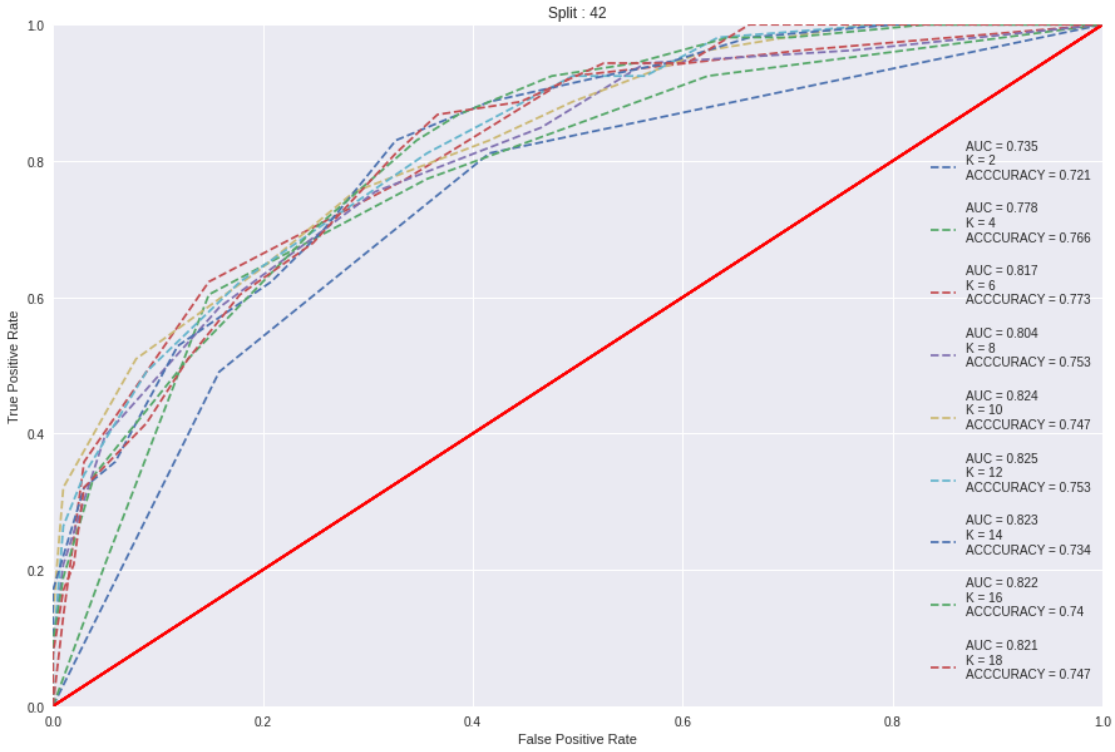
<Figure size 576x396 with 0 Axes>



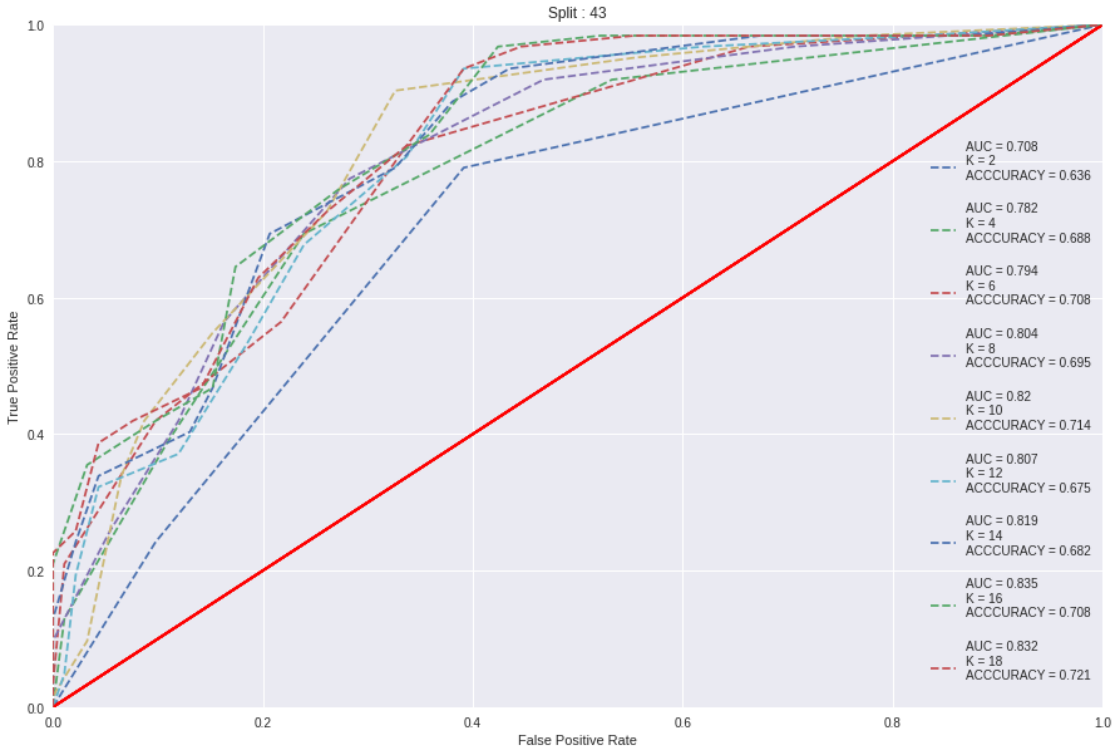
<Figure size 576x396 with 0 Axes>



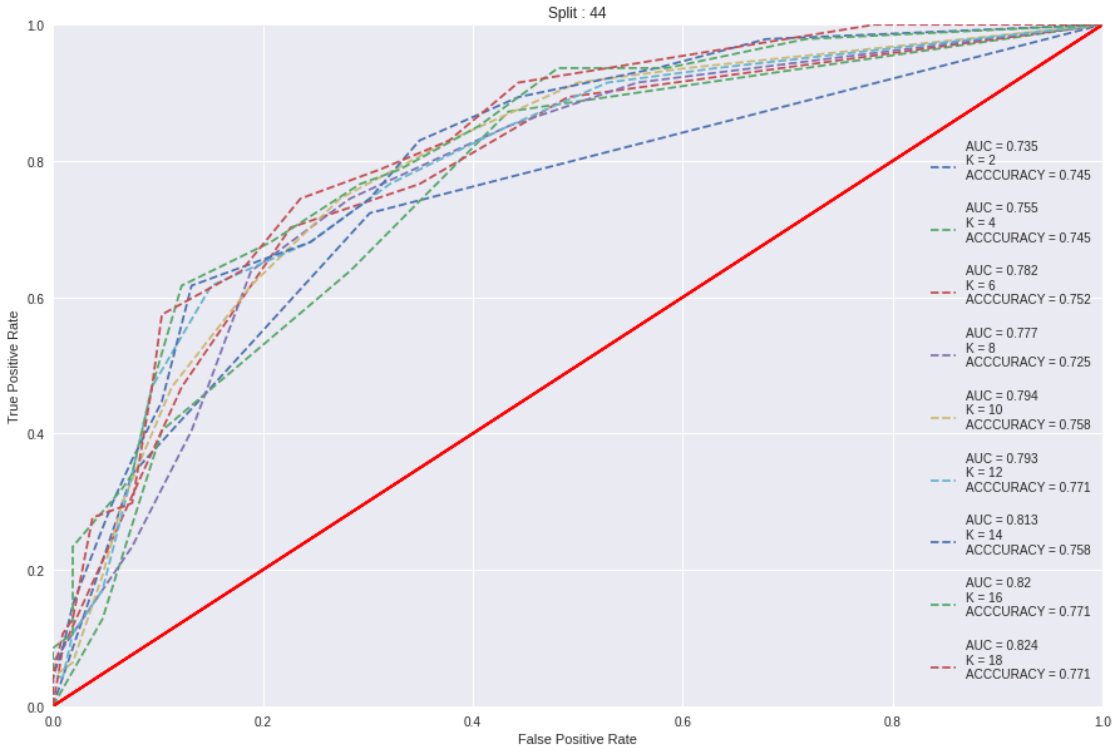
<Figure size 576x396 with 0 Axes>



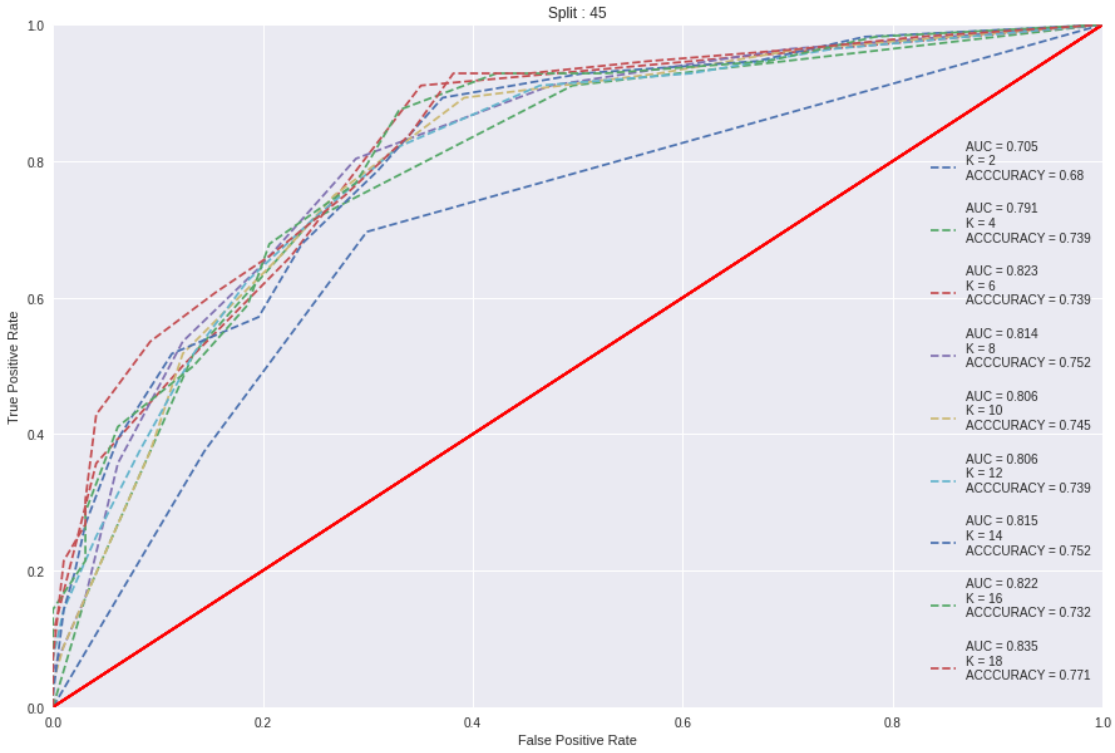
<Figure size 576x396 with 0 Axes>



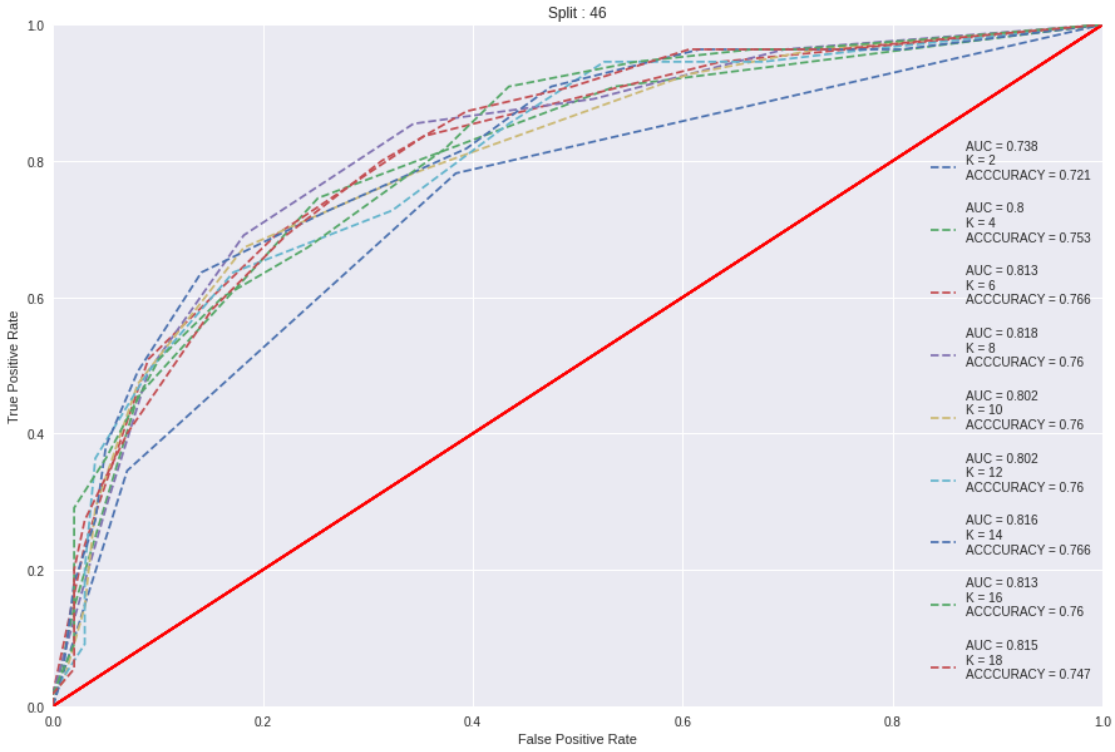
<Figure size 576x396 with 0 Axes>



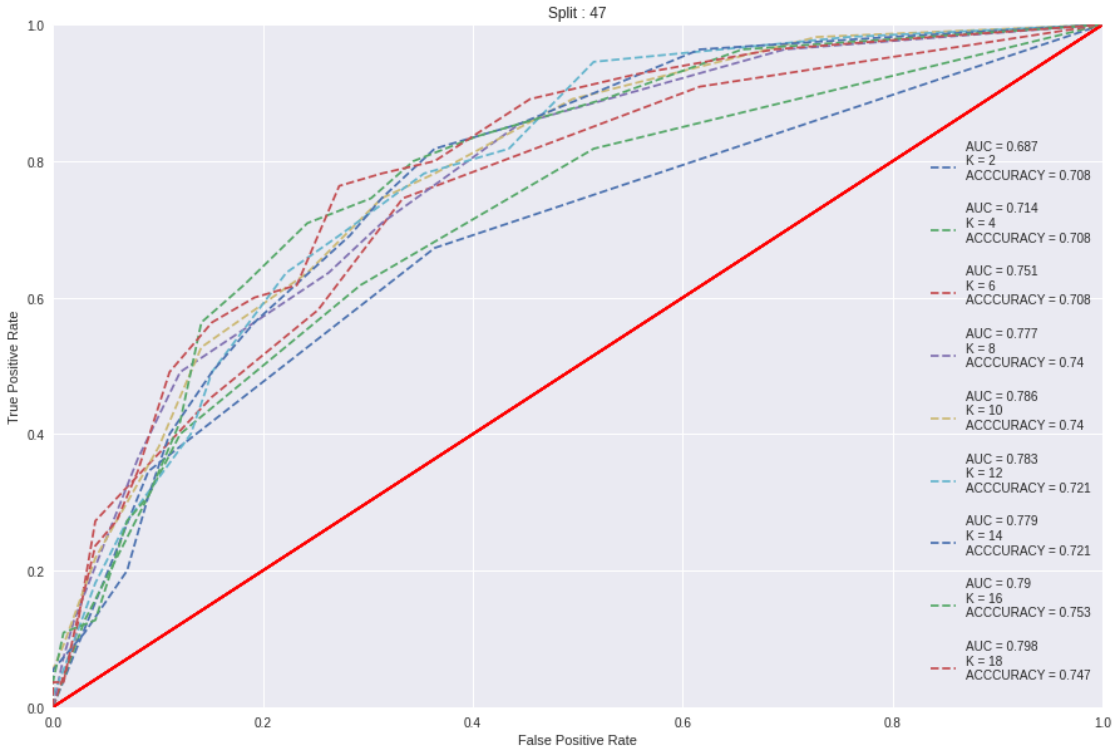
<Figure size 576x396 with 0 Axes>



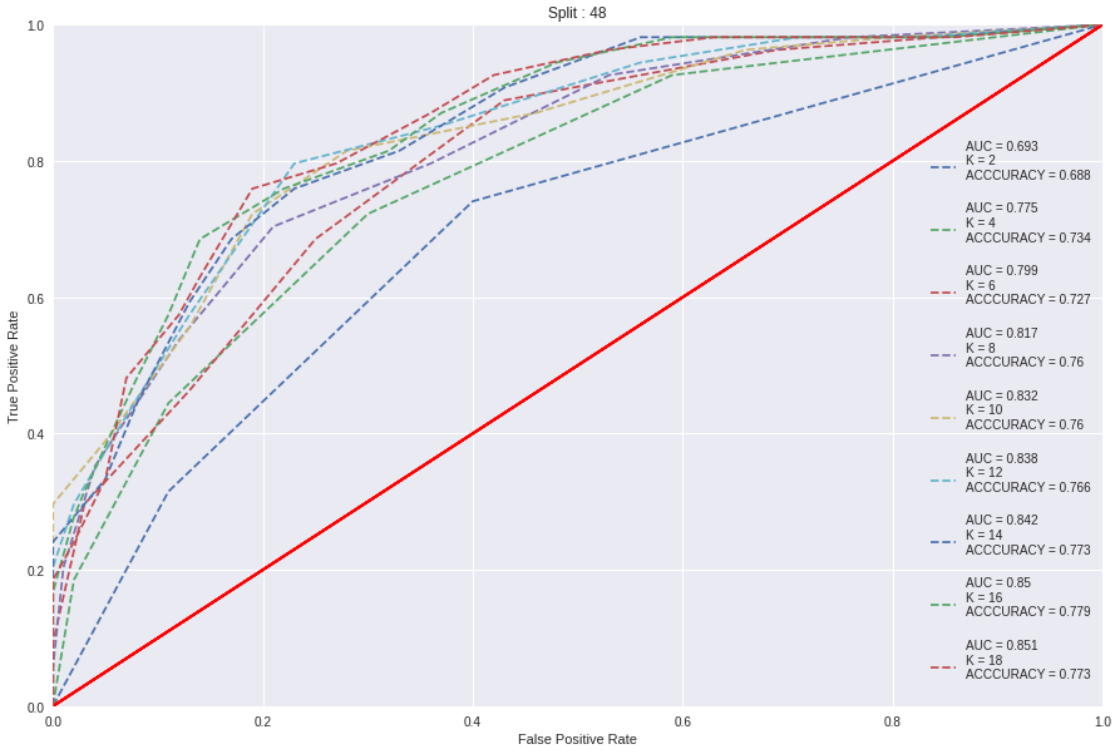
<Figure size 576x396 with 0 Axes>



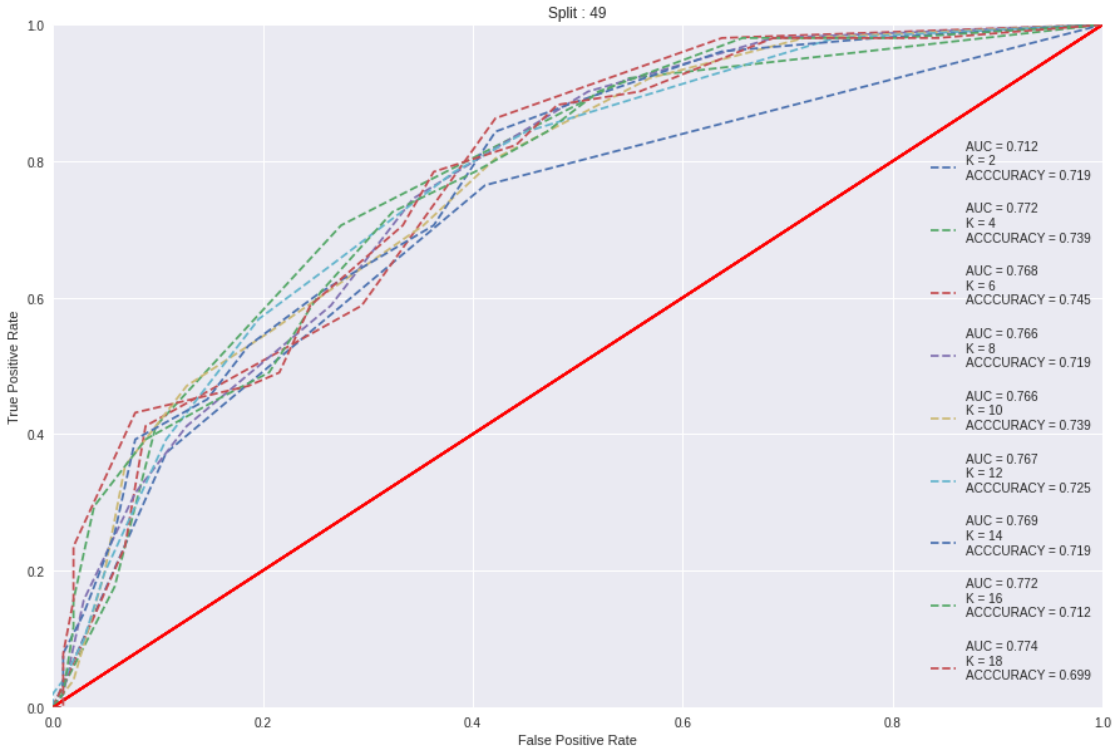
<Figure size 576x396 with 0 Axes>



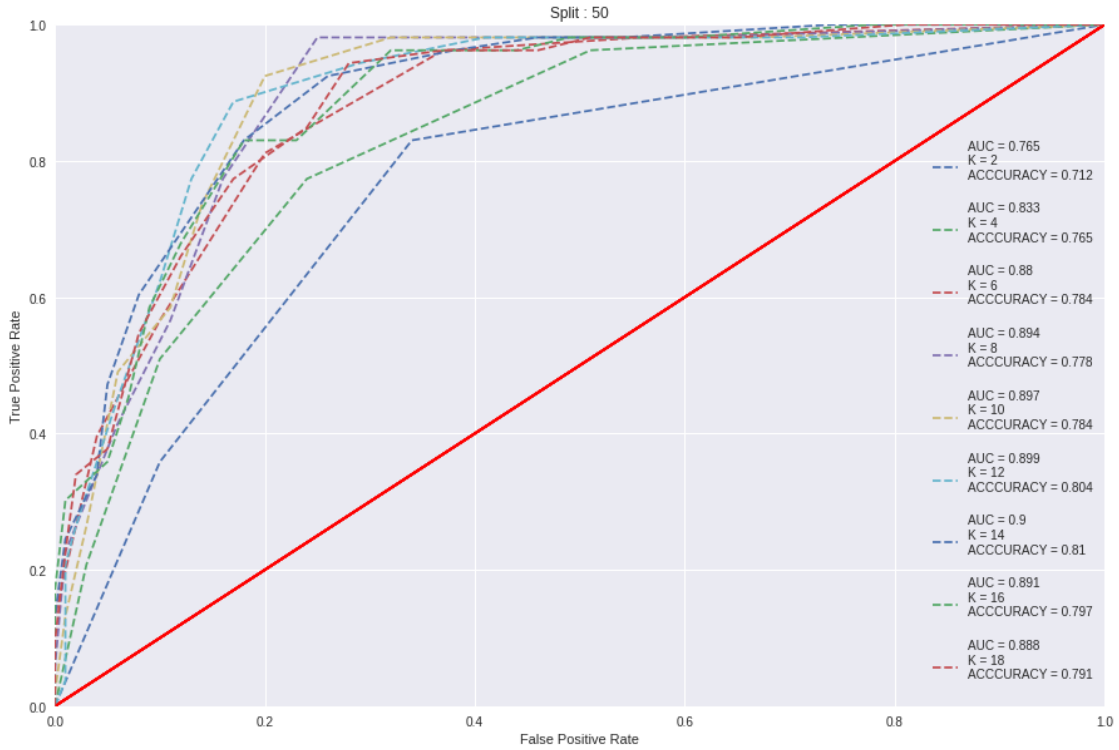
<Figure size 576x396 with 0 Axes>



<Figure size 576x396 with 0 Axes>



<Figure size 576x396 with 0 Axes>


Split : 50
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.765
K = 2
ACCCURACY = 0.712
AUC = 0.833
K = 4
ACCCURACY = 0.765
AUC = 0.88
K = 6
ACCCURACY = 0.784
AUC = 0.894
K = 8
ACCCURACY = 0.778
AUC = 0.897
K = 10
ACCCURACY = 0.784
AUC = 0.899
K = 12
ACCCURACY = 0.804
AUC = 0.9
K = 14
ACCCURACY = 0.81
AUC = 0.891
K = 16
ACCCURACY = 0.797
AUC = 0.888
K = 18
ACCCURACY = 0.791

Thankyou!!