Silesian
University
of Technology

CLASSIFIERS LABORATORY REPORT

Classification Quality Estimation

Authors:

Jean Remy UWACU
Maurice NGABONZIZA
Ankit RATHI

Introduction

A quick recap.

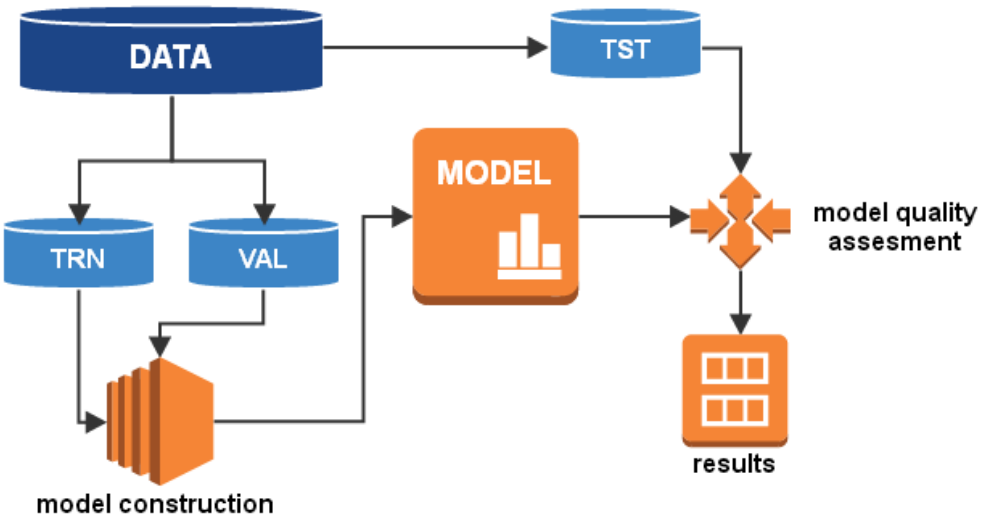
What is classification?

Classification is the process of assigning every object from a collection to exactly one class from a known set of classes.
Examples of classification tasks are:

- assigning a patient (the object) to a group of healthy or ill (the classes) people on the basis of his or her medical record,
- determining the customer's (the object) credibility during credit application using, for example, demographic and financial data; in this case the classes are „credible" and „not credible",
- determining if the customer (the object) is likely to stop using the company's services or products on the basis of behavioral and demographic data; in this case the classes are „disloyal customers" and „loyal customers".
-

How are classification models created?

1. Data preparation (importing, processing, exploration and statistical analysis)
   This stage divides the data into two or three parts:
   - training data – will be used to build the model
   - validation data (in more complex cases) – will be used for evaluation of model quality during its creation
   - testing data – will be used to establish the final quality of the model
2. Model creation (using training and (optionally) validation)
3. Model quality assessment (testing the created model on testing data)
4. Model application and subsequent monitoring (periodical checks if the quality of predictions does not deteriorate over time, for instance due to demographic or market changes)



What indicators can be used to determine the quality of classification models?

There are two kinds of indicators that can be used to estimate the quality of classification models:

- Quantitative quality indicators – statistics which express the quality of classification using numerical values.
- Graphical indicators – the quality of classification is represented on a graph which combines selected quantitative indicators. Graphical methods simplify model quality assessment and visualize classification results. Such indicators include:
  - Confusion matrix
  - ROC curve
  - LIFT chart

Basic notions used in the assessment of the quality of classification models.

Binary and multiclass classification

Binary classification:

- one class is defined as positive (also known as target class, rare class or minority class)
- other class is defined as negative (also known as normal class)
-

Multiclass classification:

- one class is defined as positive
- other classes combined are defined as negative

Positive class should collect objects which should be identified during modeling: for example in churn modeling the positive class would consist of resigning customers; in credit scoring projects the positive class consists of customers who defaulted on their debts. (In both cases the negative class consists of the remaining customers).

TP, TN, FP, FN

- **TP – True Positive** – the number of observations correctly assigned to the positive class
  Example: the model's predictions are correct and resigning customers have been assigned to the class of „disloyal" customers
- **TN – True Negative** – the number of observations correctly assigned to the negative class
  Example: the model's predictions are correct and customers who continue using the service have been assigned to the class of „loyal" customers.
- **FP – False Positive** – the number of observations assigned by the model to the positive class, which in reality belong to the negative class.
  Example: unfortunately the model is not perfect and made a mistake: some customers, who continue using the service have been assigned to the class of "disloyal" customers.
- **FN – False Negative** – the number of observations assigned by the model to the negative class, which in reality belong to the positive class.
  Example: unfortunately the model is not perfect and made a mistake: some churning customers have been assigned to the class of "loyal" customers.

predicted → real ↓	*Class_pos*	*Class_neg*
Class_pos	TP	FN
Class_neg	FP	TN

For a perfect classifier (i.e. every observation has been correctly classified) we would have:
FP = 0
FN = 0
TP = number of all observations from the positive class
TN = number of all observations from the positive class
Pos = TP + FN – number of all observations which in reality belong to the positive class
Neg = FP + TN – number of all observations which in reality belong to the negative class

Tasks

- Prepare artificial data and load *Data_PTC_vs_FTC.mat* dataset.
- Build classifier
- Split the dataset into a training and a test sets with two chosen methods (K-fold, Leave One Out or Bootstrap)
- Specify values of the following quality indicators: sensitivity, specificity, accuracy, error. Plot the ROC curves and compare the area under the curve for both sets.

To split data we can use few methods for example: K-fold, Leave-One-Out, Bootstrap. In our case we will use K-fold and Leave-One-Out to split data. These methods are very similar.

K-fold method is a resampling procedure used to evaluate machine learning models on a limited data sample. The main idea of this procedure is to split our data on k fold (for example for k=10 we split our data on 10 bins). After splitting our data on K groups, we take each group and remain it as a test data set, the rest groups is our training data set. Every time we fit our model on training test and evaluate it on the test set.

Leave one out is a special case of K-fold, but in this case the number of folds is equal the number of instance in the data set. We fit our model once for each instance, where one instance is our training set and the rest of all are instances of training set.

1. To Generate Artificial Data

Let's first generate randomly 400 data from 2 distributions

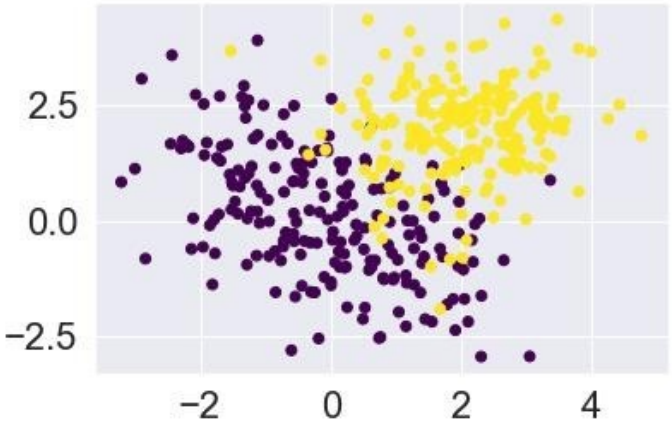


Fig.1 Scatter plot of generated data

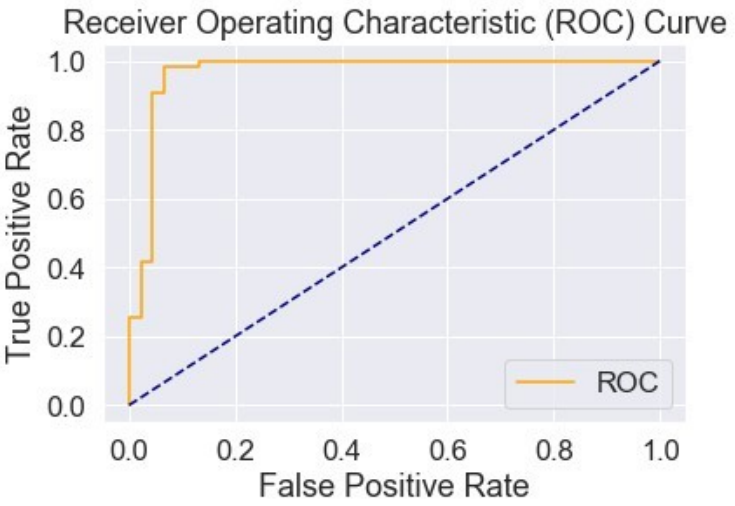


Fig.2 A ROC Curve of model splitted by K-fold

AUC:
0.9758
AUC:
97.58%

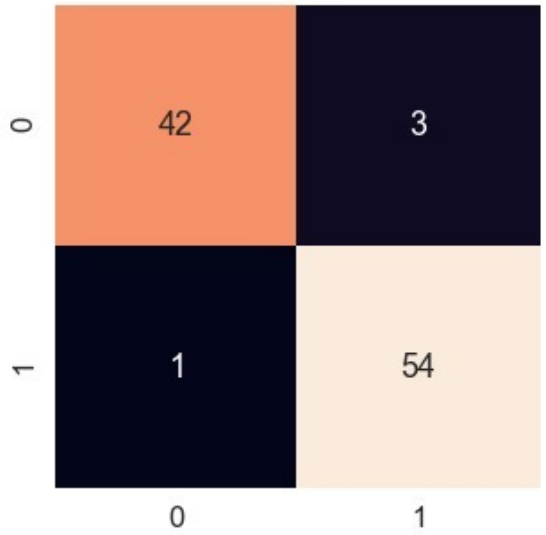


Fig.3 A Confusion matrix of model splinted by K-fold

Quality:

Sensitivity: 0.9767441860465116
Specificity: 0.9473684210526315
Accuracy: 0.96
Error: 0.046511627906976744

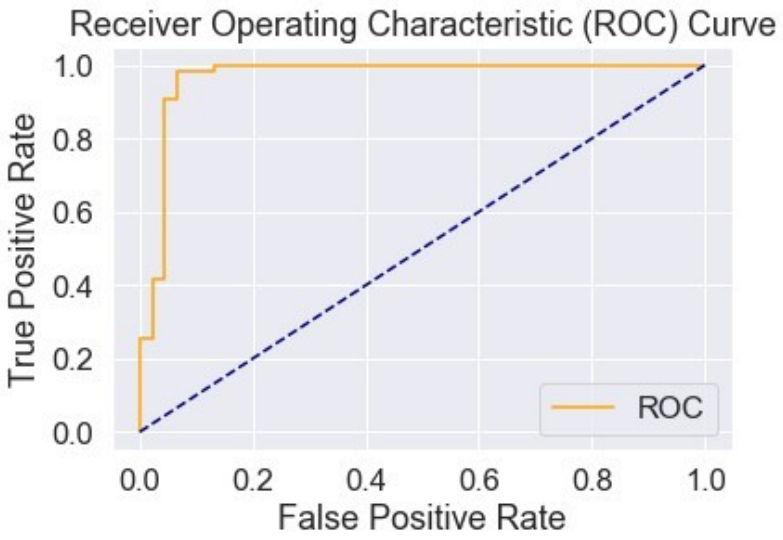


Fig.4 A ROC Curve of model splitted by LOO

AUC:
0.9758
AUC:
97.58%

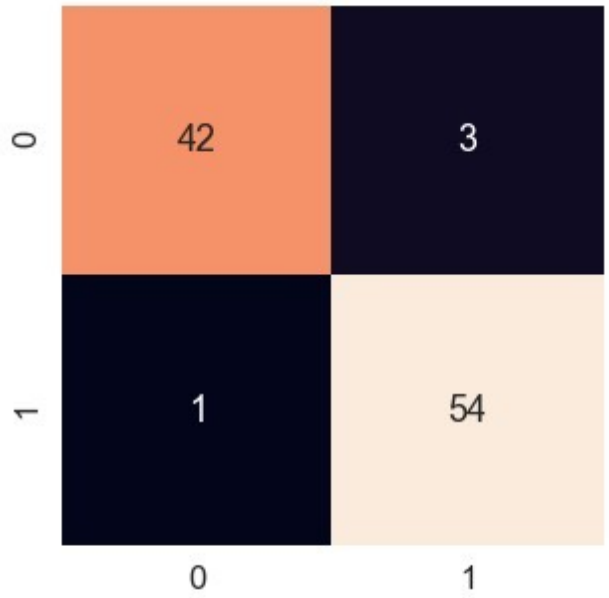


Fig.5 A Confusion matrix of model splitted by LOO

Quality:

Sensitivity: 0.9767441860465116
Specificity: 0.9473684210526315
Accuracy: 0.96
Error: 0.046511627906976744

2. **Real data**

This time we are going to use 2 genes and we plot 86 patients on a scatter plot

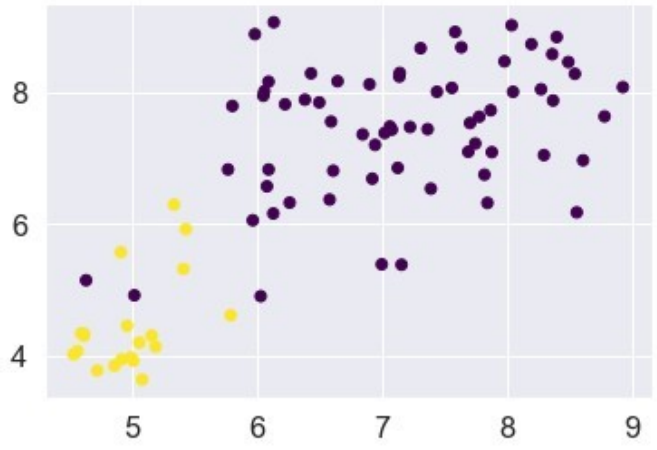


Fig.6 Scatter plots of genes

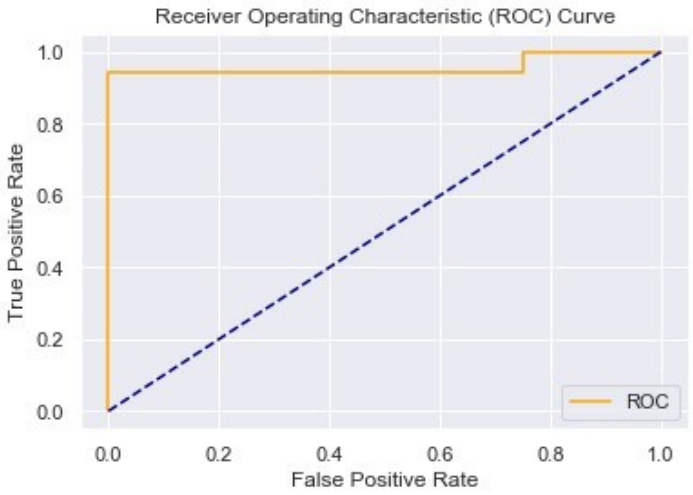


Fig.7 A Confusion matrix of model splitted by K-fold

***AUC**: 0.9583*
***AUC**: 95.8%*

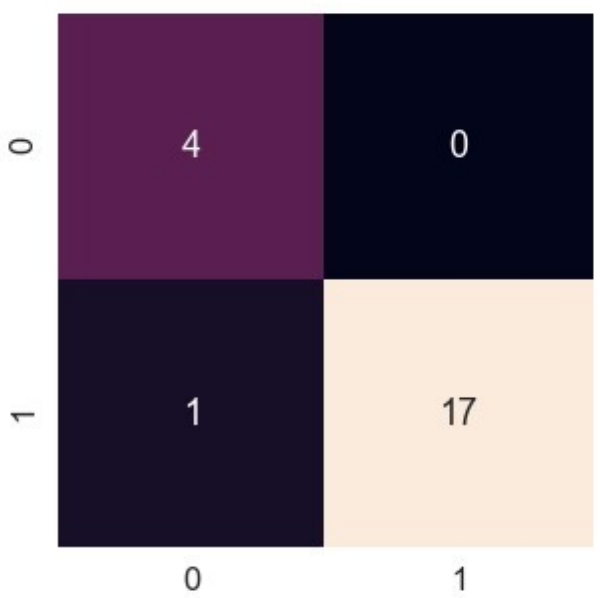


Fig.8 Confusion matrix of model splitted by K-fold

Quality:

Sensitivity: 0.8
Specificity: 1.0
Accuracy: 0.9545454545454546
Error:0.1

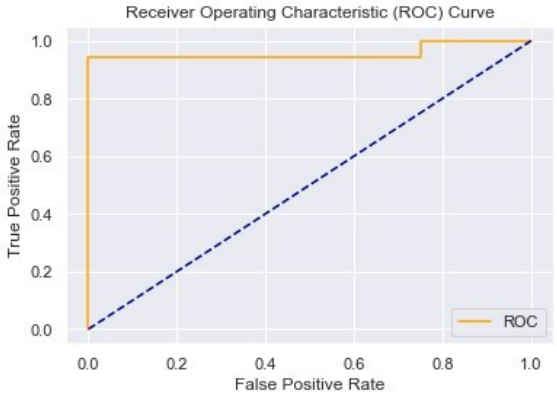


Fig.9 A Confusion matrix of model splitted by LOO

***AUC**:*
0.9583
***AUC**:*
95.8%

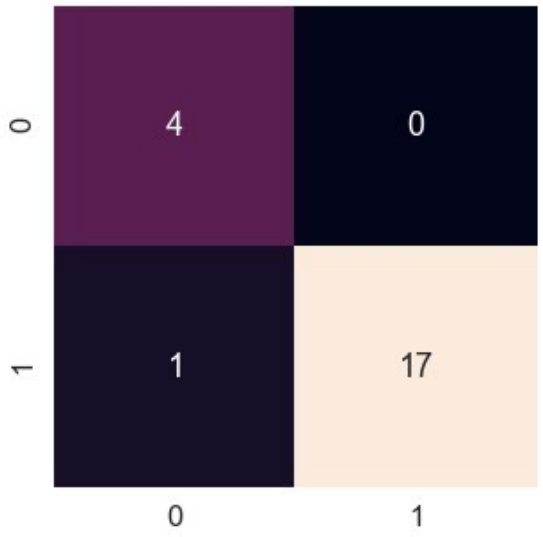


Fig.10 A Confusion matrix of model splitted by LOO

Quality:

Sensitivity: 0.8
Specificity: 1.0
Accuracy: 0.9545454545454546
Error:0.1

3. Conclusion

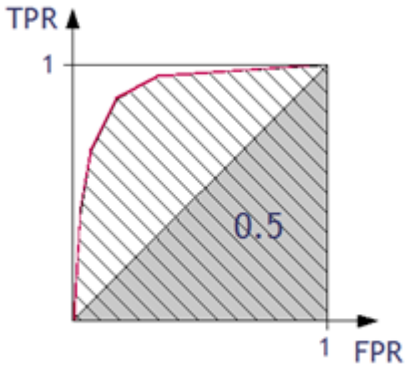
To determine the quality estimation of our classifier, we have used the ROC Curve. The ROC curve is one of the methods for visualizing classification quality, which shows the dependency between TPR (True Positive Rate) and FPR (False Positive Rate).
The more convex the curve, the better the classifier.

predicted → real ↓	*Class_pos*	*Class_neg*
Class_pos	TP	FN
Class_neg	FP	TN

$$\text{TPR (sensitivity)} = \frac{TP}{TP + FN}$$

$$\text{FPR (1-specificity)} = \frac{FP}{TN + FP}$$

Assessing the classifier on the basis of the ROC curve



The quality of classification can be determined using the ROC curve by calculating the:

- area under ROC Curve (AUC) coefficient

The higher the value of AUC coefficient, the better. AUC = 1 means a perfect classifier, AUC = 0.5 is obtained for purely random classifiers. AUC < 0.5 means the classifier performs worse than a random one.

- Gini Coefficient: GC = 2 *AUC – 1 (the classifier's advantage over a purely random one)

The higher, the value of GC, the better. GC = 1 denotes a perfect classifier, GC = 0 denotes a purely random one.

In our case **AUC** results for real data and artificial data by using method of **K-fold** and **LOO** showed similar results. **AUC** for models where data was splitting by K-fold and LOO was the same. **AUC** for artificial data was the same as well for both methods.

At one point we have seen that the results between these methods were different, on a margin of 1% approximately.

Code:

```
import pandas as pd
import numpy as np
import mat4py
from sklearn.svm import SVC
from sklearn.utils import shuffle
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import LeaveOneOut
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split from
sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
import seaborn as sns


###sample dataset
mean_class_1 = [0, 0]
covariance_class_1 = [[2, -1], [-1, 2]]  # diagonal covariance-class 1
class_1_label = [1 for i in range(0, 200)]

mean_class_2 = [2, 2]
covariance_class_2 = [[1, 0], [0, 1]]  # diagonal covariance-class 2
class_2_label = [0 for i in range(0, 200)]

class_1 = np.random.multivariate_normal(mean_class_1, covariance_class_1, 200)
class_2 = np.random.multivariate_normal(mean_class_2, covariance_class_2, 200)

dataset_class_one = pd.DataFrame({'X1': class_1[:, 0], 'X2': class_1[:, 1],
'y': class_1_label})
dataset_class_two = pd.DataFrame({'X1': class_2[:, 0], 'X2': class_2[:, 1],
'y': class_2_label})
```

```
dataset = dataset_class_one.append(dataset_class_two)
#
#plt.scatter(dataset['X1'], dataset['X2'], c=dataset['y'], cmap='viridis_r')
#
#plt.show()

# plotting ROC CURVE
def plot_roc_curve(fpr, tpr):
      plt.plot(fpr, tpr, color='orange', label='ROC') plt.plot([0, 1], [0,
      1], color='darkblue', linestyle='--') plt.xlabel('False Positive
      Rate')
      plt.ylabel('True Positive Rate')
      plt.title('Receiver Operating Characteristic (ROC) Curve')
      plt.legend()
      plt.show()


def cross_validation(estimator, number_of_folds: int, dataset_X: np.array,
dataset_y: np.array):
      accuracies = []
      dataset_X, dataset_y = shuffle(dataset_X, dataset_y)
      kf = KFold(n_splits=number_of_folds)
      kf.get_n_splits(dataset_X)

      for train_index, test_index in kf.split(dataset_X):
            # splitting data
            X_train, X_test = dataset_X[train_index], dataset_X[test_index]
            y_train, y_test = dataset_y[train_index], dataset_y[test_index]

            # generating classifier and calculation of accuracy
            estimator.fit(X_train, y_train.ravel())
            prediction = estimator.predict(X_test)
            accuracies.append(accuracy_score(y_test, prediction))

      return accuracies


def leave_one_out(estimator, dataset_X: np.array, dataset_y: np.array):
      accuracies = []
      dataset_X, dataset_y = shuffle(dataset_X, dataset_y)
      loo = LeaveOneOut()
      loo.get_n_splits(dataset_X)

      for train_index, test_index in loo.split(dataset_X):
#        print("TRAIN:", train_index, "TEST:", test_index)
            X_train, X_test = dataset_X[train_index], dataset_X[test_index]
            y_train, y_test = dataset_y[train_index], dataset_y[test_index]

            # generating classifier and calculation of accuracy
```

```
            estimator.fit(X_train, y_train.ravel())
            prediction = estimator.predict(X_test)
            accuracies.append(accuracy_score(y_test, prediction))

    return accuracies


def calulate_qualities(cm):
    print()
    print("CONFUSION MATRIX: ")
    Sensitivity = cm[0][0] / (cm[0][0] + cm[1][0]) # Sensitivity = TP / (TP +
FN)
    Specificity = cm[1][1] / (cm[1][1] + cm[0][1]) # Specificity = TN / (TN +
FN)
    Accuracy = (cm[0][0] + cm[1][1]) / (cm[0][0] + cm[1][0] + cm[0][1] +
cm[1][1]) # Accuracy= TP + TN / (TP + TN + FP + FN)
    Error = (cm[1][0] + cm[0][1]) / (cm[0][0] + cm[1][0] + cm[0][0] +
cm[1][0]) # Error = FP + FN / (TP + TN + FN + FP)
    print("Sensitivity: ", Sensitivity) print("Specificity:
    ", Specificity) print("Accuracy: ", Accuracy)
    print("Error: ", Error)



########################## ARTFICIAL DATA
#########################################
# getting data
X = np.array(dataset[['X1', 'X2']])
y = np.array(dataset[['y']])


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=0)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

 # creating classifier K-fold
classifier_K_fold = SVC(kernel='rbf', random_state=0, gamma='auto',
probability=True)
classifier_K_fold.fit(X_train, y_train)

 # creating classifier Leave one out
classifier_Leave_one_out = SVC(kernel='rbf', random_state=0, gamma='auto',
probability=True)
classifier_Leave_one_out.fit(X_train, y_train)

#calculating accuracy_score
```

```
accuracy_score_K_fold = cross_validation(estimator = classifier_K_fold,
number_of_folds=10, dataset_X=X_train, dataset_y=y_train)
accuracy_score_Leave_One_Out =
leave_one_out(estimator=classifier_Leave_one_out, dataset_X=X_train,
dataset_y=y_train)

#prediction of estimators
prediction_K_fold = classifier_K_fold.predict(X_test)
prediction_Leave_one_out = classifier_Leave_one_out.predict(X_test)

prediction1 = classifier_K_fold.predict_proba(X_test)
prediction1 = prediction1[:, 1]

prediction2 = classifier_K_fold.predict_proba(X_test)
prediction2 = prediction2[:, 1]

#confusion matrix
cm_K_fold = confusion_matrix(y_test, prediction_K_fold)
cm_Leave_one_out = confusion_matrix(y_test, prediction_Leave_one_out)

#ploting ROC curves
fpr, tpr, threshold = roc_curve(y_test, prediction1)
plot_roc_curve(fpr, tpr)

fpr, tpr, threshold = roc_curve(y_test, prediction2)
plot_roc_curve(fpr, tpr)

calulate_qualities(cm_K_fold)
calulate_qualities(cm_Leave_one_out)


########################################## REAL DATA
##########################################

data = mat4py.loadmat('Data_PTC_vs_FTC.mat')

D = pd.DataFrame(data['Data']['D'])
X = pd.DataFrame.transpose(pd.DataFrame(data['Data']['X']))
G = pd.DataFrame.transpose(pd.DataFrame(data['Data']['gene_names']))

X_data_var = pd.DataFrame(data['Data']['X'])

X_data = np.array(pd.DataFrame.transpose(pd.DataFrame(X_data_var.iloc[1:3,
:].values)))
Y_data = np.array(pd.DataFrame(D.iloc[:, 0].values))

plt.scatter(X_data[:, 0:1], X_data[:, 1:], c=Y_data, cmap='viridis_r')

plt.show()
```

```
X_train2, X_test2, y_train2, y_test2 = train_test_split(X_data, Y_data,
test_size=0.25, random_state=0)
sc = StandardScaler()
X_train2 = sc.fit_transform(X_train2)
X_test2 = sc.transform(X_test2)

##creating classifier K-fold
classifier_Leave_one_out_R = SVC(kernel='rbf', random_state=0, gamma='auto',
probability=True)
classifier_Leave_one_out_R.fit(X_train2, y_train2)

# creating classifier Leave one out
classifier_K_fold_R = SVC(kernel='rbf', random_state=0, gamma='auto',
probability=True)
classifier_K_fold_R.fit(X_train2, y_train2)

accuracy_score_Leave_One_OutR =
leave_one_out(estimator=classifier_Leave_one_out_R, dataset_X=X_train2,
dataset_y=y_train2)
accuracy_score_K_foldR = cross_validation(estimator = classifier_K_fold_R,
number_of_folds=10, dataset_X=X_train2, dataset_y=y_train2)

prediction_K_fold_R = classifier_K_fold_R.predict(X_test2)
prediction_Leave_one_out_R = classifier_Leave_one_out_R.predict(X_test2)

prediction3 = classifier_Leave_one_out_R.predict_proba(X_test2)
prediction3 = prediction3[:, 1]

prediction4 = classifier_K_fold_R.predict_proba(X_test2)
prediction4 = prediction4[:, 1]

cm_K_fold_R = confusion_matrix(y_test2, prediction_K_fold_R)
cm_Leave_one_out_R = confusion_matrix(y_test2, prediction_Leave_one_out_R)

fpr, tpr, threshold = roc_curve(y_test2, prediction3)
plot_roc_curve(fpr, tpr)

fpr, tpr, threshold = roc_curve(y_test2, prediction4)
plot_roc_curve(fpr, tpr)

# calculation of qualities
calulate_qualities(cm_K_fold_R)
calulate_qualities(cm_Leave_one_out_R)
##########################################################################
##########################
#####
ax= plt.subplots(figsize=(5,5))
sns.set(font_scale=1.5)
```

```
sns_plot = sns.heatmap(cm_K_fold, cbar=False, square=True, annot=True,fmt='g')

ax= plt.subplots(figsize=(5,5))
sns.set(font_scale=1.5)
sns_plot = sns.heatmap(cm_Leave_one_out, cbar=False, square=True,
annot=True,fmt='g')

ax= plt.subplots(figsize=(5,5))
sns.set(font_scale=1.5)
sns_plot = sns.heatmap(cm_K_fold_R, cbar=False, square=True,
annot=True,fmt='g')

ax= plt.subplots(figsize=(5,5))
sns.set(font_scale=1.5)
sns_plot = sns.heatmap(cm_Leave_one_out_R, cbar=False, square=True,
annot=True,fmt='g')
```