Machine Learning Programming
Assignment 3: K-Nearest Neighbors

Professors: Baptiste Busch, Aude Billard
Assistants: Laila El Hamamsy, Matthieu Dujany
and Victor Faraut
Contacts:
baptiste.busch@epfl.ch, aude.billard@epfl.ch
laila.elhamamsy@epfl.ch, matthieu.dujany@epfl.ch, victor.faraut@epfl.ch

Winter Semester 2018

Introduction

In this practical, you will code the K-Nearest Neighbors algorithm in Matlab. You will then use the code for classification on 2D datasets before applying it on higher dimensional datasets.

Submission Instructions

Deadline: November 13, 2018 @ 6pm. Assignments must be turned in by the deadline. 1pt will be removed for each day late. A day late starts one hour after the deadline.
Procedure: From the course Moodle webpage, the student should download and extract the .zip file named `TP3-KNN-Assignment.zip` which contains the following files:

Part 1 - Algorithm	Part 2 - Applications
`my_knn.m`	`confusion_matrix`
`my_accuracy.m`	`knn_ROC.m`
`knn_eval.m`	`cross_validation.m`
`test_knn_2d.m`	`preprocess_data.m`
--	`gower_similarity.m`
--	`test_knn_9D.m`
--	`test_adult_dataset.m`

Assignment Instructions

The assignment consists of implementing the blue colored MATLAB functions from scratch. These functions can be tested with the `test_*.m` scripts. These testing scripts depend on `ML_toolbox`, which is already installed on the Virtual Machines. In case you want to work on your own computer you will have to download it from: `https://github.com/epfl-lasa/ML_toolbox` and change the path in each scripts by modifying the following line.

```
>> addpath(genpath('path_to_ML_toolbox'))
```

Once you have tested your functions, you can submit them as a .zip file with the name: `TP3-KNN-Assignment-SCIPER.zip` on the submission link in the Moodle webpage. Your submission archive should contain ONLY the following:

```
🟥 TP3-KNN-Assignment-SCIPER
 └─🟦 Part1
 └─🟦 Part2
```
DO NOT upload `ML_toolbox`, the `check_utils`, directory, the `utils` directory, or the `Datasets` directory.

1 Part 1: K-Nearest Neighbors (K-NN)

K-NN (K-Nearest Neighbors) is a nonparametric "lazy" learning algorithm that can be used for classification or regression. In this assignment we will cover its use for classification purposes only. It is one of the simplest classification algorithms in the machine learning literature, generally used as a baseline for more complex algorithms.

K-NN is considered as *nonparametric* as it does not make any theoretical assumptions of the distribution of the underlying data (e.g. linearly separable, normally distributed, etc.). This makes it applicable to a lot of dataset. It is considered a *lazy* algorithm because it does not learn any parameters or model to generalize the observed data, it rather uses the full training dataset to find the best outcome for a query point. This is done by keeping the complete training data during testing and using a distance-based majority vote mechanism, to predict a label for a new sample (i.e. query point).

K-NN relies on a similar mechanism as DBSCAN, presented in the Applied Machine Learning course, but for supervised learning tasks (i.e. classification). Like DBSCAN, it groups points according to how distant they are from each other using a minimum (K) of points. DBSCAN adds a condition on the minimal distance to detect outliers and merge the clusters. K-NN does not have these two additional mechanisms and is hence very sensitive to outliers.

K-Nearest Neighbor Algorithm

In classification problems we are given a training dataset $D = \{(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^M, y^M)\}$ where $\mathbf{x}^i \in \mathbb{R}^N$ is the i-th N-dimensional data point (or feature vector) for $i = 1, \ldots, M$ and $y^i \in \mathbb{N}$ is the categorical outcome (or class label) corresponding to each data point. Generally, the labels are a sequence of integer ranging from 1 to C, where C is the number of classes (you might also find some datasets labelled from 0 to $C-1$). The goal is then, given a new sample (or query point) $\mathbf{x}' \in \mathbb{R}^N$ we would like to predict its label $\hat{y}' \in \mathbb{N}$[1]. The K-NN algorithm addresses this problem by applying a very simple rule (For further reading, refer to Chapter 4 of the Pattern Classification book by Duda et al. [1]:

"$\mathbf{x}'$ is assigned the label $\hat{y}'$ most frequently represented among its k nearest samples."

Hence, the classification decision for $\mathbf{x}'$ involves the following steps:

1. Calculate the pairwise distances between $\mathbf{x}'$ and all points in the training dataset $D_x = \{\mathbf{x}^1, \ldots, \mathbf{x}^M\}$.
2. Extract the k nearest neighbors of $\mathbf{x}'$ from the pairwise distances computed in step 1 $\mathbf{x}'_k = \{x_1, \ldots, x_k\}$ and their corresponding class labels $\mathbf{y}^k = [y^1, \ldots, y^k]$.
3. Majority vote on class labels based on the k nearest neighbor list $\mathbf{y}^k$.

In other words, the K-NN algorithm begins with a query point $\mathbf{x}'$ (as can be seen in Figure 1) and grows a spherical region until it encloses k training points, regardless of their labels. The query point is then labeled by a majority vote from the enclosed training points.

[1]We use $\hat{}$ for $\hat{y}$ to indicate that it is an estimated value and not the true label y.

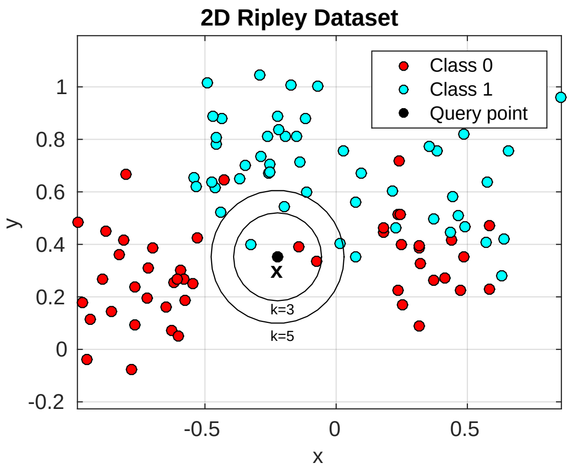


Figure 1: KNN example on the Ripley Dataset (generated by a mixture of two Gaussian distributions). In the case of $k = 3$, $\mathbf{x}$ would be labeled as class 0. However, in the case of $k = 5$, $\mathbf{x}$ would be labeled as class 1.

Special considerations of KNN:

- k must be an odd value for an even-class problem. This avoids ties during the majority voting step. The inverse holds, use an even number for k when you have an odd-class problem.
- Choosing k is the most important step in this algorithm. If $k \to 1$ is too small, noise or outliers in the data will have a higher effect on the result. On the other hand, if $k \to M$ is too large, computation time is hindered since KNN has a computational complexity of $\mathcal{O}(Mk)$. Moreover, setting k to a large value, contradicts the idea behind KNN; i.e. points closer together belong to the same class. Intuitively, one can assume equal distribution of points across classes and we can find a feasible range of k around $k = \kappa * M/C$ with $0 < \kappa <= 1$ and C is the number of classes, which is the ratio of number of datapoints and number of classes. This is not a strict rule, but can be useful when finding an optimal k, one can also make an informed decision of k by analyzing the density of the points, balance in classes, etc.

Data Handling for Classification

We will test our implementation of KNN on the 2D concentric circle dataset shown in Figure 2. For any type of classification algorithm, one must first split the dataset into *training* and *testing* sets. This train/test split is used in order to reduce overfitting with your classifier. By using the full dataset for training, the model or classifier will tend to overfit to the noise of the observed data, rather than the real, underlying model. The data points in your *training* set are used to tune parameters (i.e. choose the best k for KNN) or learn models of your classifier. The data points on your *testing* set are considered as a separate dataset used solely to evaluate the performance of your learned classifier.
To partition a dataset for classification one generally selects a validation ratio: `valid_ratio`, where `valid_ratio`=0.3 corresponds to 30% of your points used for *testing* (or validation) and the rest of your points used for *training*. One can vary the `valid_ratio` to have a better estimate of the classifier's performance. In general, `valid_ratios` are tested in the range of $\{30, 40, 50, 60, 70\}$, depending on how much data you have. Special considerations of train/test sets:

1. A large validation or test set generally gives a more reliable estimate of the classifier's accuracy (i.e. a lower variance estimate).
2. A large training set will generally be more representative of how much data we actually have for learning process.

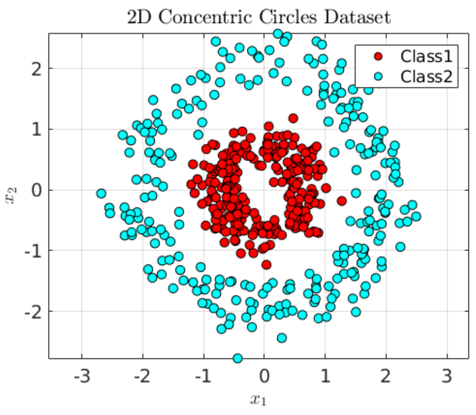


Figure 2: 2D Concentric circles Dataset. $M = 500$, $N = 2$, $y \in 1, 2$

3. Using only a training set (no validation set) does not give a reliable accuracy estimate. It cannot tell how sensitive the classifier is wrt. specific samples.

The splitting function has already been implemented during the exercise sessions and, therefore, is given in `utils` folder.

Distances for KNN Classification

The KNN classifier relies on a metric or "distance" function between the data points in order to accomplish the first step of the algorithm:

1. Calculate the pairwise distances between $\mathbf{x}'$ *and all points in the training dataset* $D_x = \{\mathbf{x}^1, \ldots, \mathbf{x}^M\}$.

Such a distance function has already been implemented in the `my_distance.m` function for the `TP2-KMeans-Assignment`. Although we have three types of distance implemented in this function, for simplicity, during this assignment we will mostly use the Euclidean distance:

$$d_{L_2}(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_2 = \sqrt{\sum_{i=1}^{N} |x_i - x_i'|^2}. \tag{1}$$

which should be called as `my_distance(x_1,x_2,'L2')`. In Part 2 of the assignment you will also implement another metric for dataset that combines continuous and categorical data. For ease of use, we provide the correct implementation of this function in the `utils` folder.

1.1 KNN algorithm

1.1.1 Compute Pairwise Distances

Now that we have a distance function, we can start implementing the KNN algorithm. The first step is to compute the pairwise distances between $\mathbf{x}'$ and D_x,

$$\mathbf{d} = \{d^i(\mathbf{x}', \mathbf{x}^i) \quad | \quad \forall i = 1, \ldots, M\} \tag{2}$$

where $\mathbf{d} \in \mathbb{R}^M$, is 1

1.1.2 Extract k-Nearest Neighbors

To extract the k-nearest neighbors, one then chooses the k elements of $\mathbf{d}$ which have the smallest distance. This can be done by first sorting the $\mathbf{d}_{(s)}$ in ascending order

$$\mathbf{d}_{(s)} = \{d^i_{(s)} \quad | \quad d^i_{(s)} < d^{i+1}_{(s)} < \cdots < d^M_{(s)}\}$$

and then selecting the subset of k points and labels that are closest to $\mathbf{x}'$:

$$y_{knn} = \{y^{I(d^1_{(s)})}, y^{I(d^2_{(s)})}, \ldots, y^{I(d^k_{(s)})}\}, \tag{3}$$

where $I(d^i_{(s)})$ outputs the index in the original dataset D of the selected point.

1.1.3 Majority Vote

Once $y_{knn} = \{y^1_{knn}, \ldots, y^k_{knn}\}$ has been retrieved from the previous step, we can estimate the label of $\mathbf{x}'$ by a majority vote from y_{knn}:

$$\hat{y}' = argmax(\{\sum y_{knn} = c_1, \sum y_{knn} = c_2, \cdots \sum y_{knn} = c_N\}), \tag{4}$$

where $\{c_1, c_2, \ldots c_N\}$ are the labels associated to each classes C. Now we will implement these three steps in the `my_knn.m` function.

TASK 1: Implement my_knn.m function (6pts)

```
function [ y_est ] =  my_knn(X_train,  y_train, X_test, params)
%MY_KNN Implementation of the k-nearest neighbor algorithm
%   for classification.
%
%   input -----------------------------------------------------------------
%
%       o X_train   : (N x M_train), a data set with M_test samples each ...
    being of dimension N.
%                               each column corresponds to a datapoint
%       o y_train   : (1 x M_train), a vector with labels y \in {0,1} ...
    corresponding to X_train.
%       o X_test    : (N x M_test), a data set with M_test samples each being ...
    of dimension N.
%                               each column corresponds to a datapoint
%       o params : struct array containing the parameters of the KNN (k,
%                  d_type and eventually the parameters for the Gower
%                  similarity measure)
%
%   output ----------------------------------------------------------------
%
%       o y_est    : (1 x M_test), a vector with estimated labels y \in {1,2}
%                   corresponding to X_test.
```

`Implementation Hint: Useful functions: my_distance(), sort() and unique().`

Test Implementation

By running the **third** code block, you will visualize the results of your KNN implementation and the encrypted `test_myknn.p` function will compare the predicted labels from your `my_knn.m` function with the ones from KNN implemented in `ML_toolbox`. If you get the following messages for the entire range of K on your MATLAB command window, you can move on to the next task.

```
--- Testing my_knn.m with k=1:2:44 ---
[Test k=1] kNN result is: CORRECT
[Test k=3] kNN result is: CORRECT
[Test k=5] kNN result is: CORRECT
[Test k=7] kNN result is: CORRECT
[Test k=9] kNN result is: CORRECT
...
```

1.2 Classification Accuracy

To evaluate the performance of our classifier, we can estimate the classification accuracy. Given the true test labels $\mathbf{y}'$ and the estimated test labels $\hat{\mathbf{y}}'$ for a test set D_{test} of M_{test} points the accuracy can be computed as follows,

$$acc = \frac{\sum_{i=1}^{M_{test}} \delta_{(y_i', \hat{y}_i')}}{M_{test}} \tag{5}$$

where $\delta_{(y_i', \hat{y}_i')} = \begin{cases} 1 & \text{if} \quad y_i' = \hat{y}_i' \\ 0 & \text{otherwise} \end{cases}$. In other words, the accuracy is the percentage of correctly classified data points from all points in D_{test}.

TASK 2: Implement my_accuracy.m function (1pt)

```
function [acc] =  my_accuracy(y_test , y_est)
%My_accuracy Computes the accuracy of a given classification estimate.
%   input -----------------------------------------------------------------
%
%       o y_test  : (1 x M_test),  true labels from testing set
%       o y_est   : (1 x M_test),  estimated labes from testing set
%
%   output ----------------------------------------------------------------
%
%       o acc     : classifier accuracy
```

Test Implementation

By running the **fourth** code block, the encrypted `test_myaccuracy.p` function will compare the estimated accuracy from your `my_accuracy.m` function with the `ML_toolbox` function.

```
--- Testing my_accuracy.m ---
[Test] Accuracy implementation: Correct.
```

Moreover, by running the **fifth** code block, you will be able to visualize the results of your `my_knn.m` and `my_accuracy.m` functions, as in Figure 3 for $k = 1$. You can modify k to have the following values $k = \{1, 5, 15, 35, 61, 75\}$, to see how k-NN behaves on the same dataset. If you obtain the plots in Figure4, 5, 6, 7 and 8, you can move on to the next task. You must take into consideration that since `valid_ratio=0.7`, due to the random data split you can have different values of *accuracy* for the same K. By running the decision-boundary code block, one can visualize the decision boundaries for each output as in Figures 9,10 and 11.

NOTE: Given the random nature of the split_data function, the following plots don't necessarily have to be the same. In fact, for a low `valid_ratio`; i.e. `valid_ratio` > 0.7 and a high K; i.e. $K > 30$ this will undoubtedly be the case.

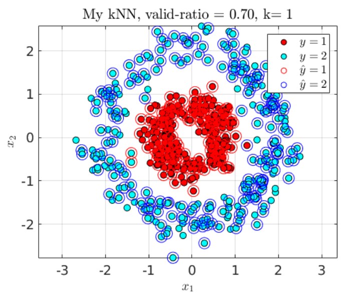


Figure 3: 2D Concentric Dataset k(1)-NN Results.

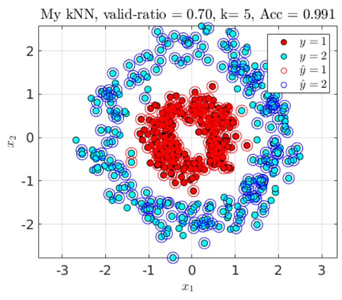


Figure 4: 2D Concentric Dataset k(5)-NN Results.

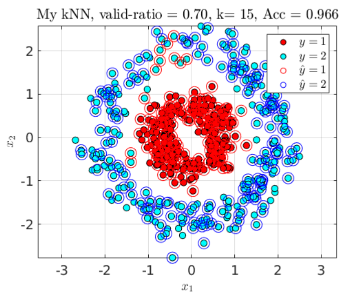


Figure 5: 2D Concentric Dataset k(15)-NN Results.

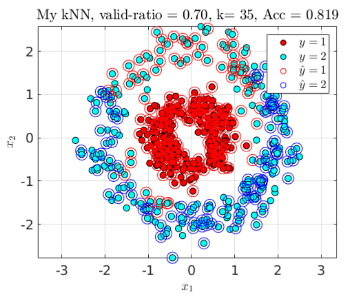


Figure 6: 2D Concentric Dataset k(35)-NN Results.

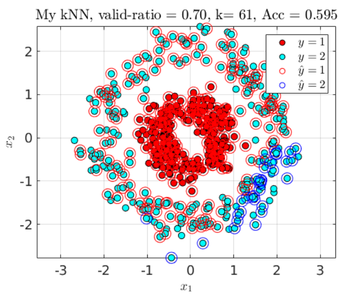


Figure 7: 2D Concentric Dataset k(61)-NN Results.

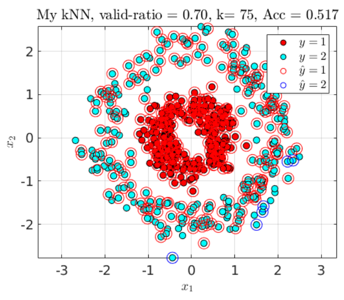


Figure 8: 2D Concentric Dataset k(75)-NN Results.

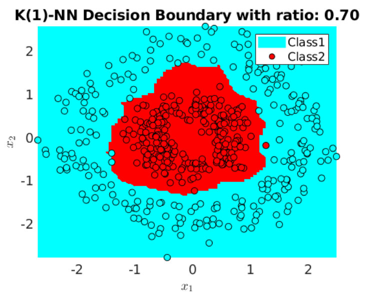


Figure 9: 2D Concentric Dataset k(1)-NN Decision Boundaries.

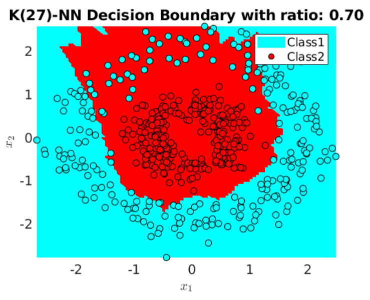


Figure 10: 2D Concentric Dataset k(27)-NN Decision Boundaries.

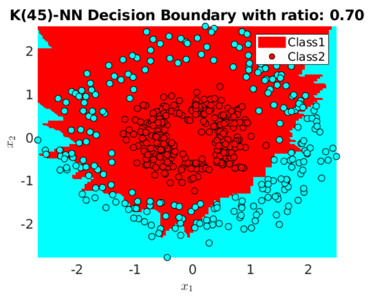


Figure 11: 2D Concentric Dataset k(45)-NN Decision Boundaries.

1.3 Choosing the optimal k for kNN Classification

Until now, we have not addressed the issue of selecting the appropriate k value for our dataset and how it influences the classification result. This can be done by analyzing the *acc* for a range of k values, the same way we analyzed k-means.

TASK 3: Implement knn_eval.m function (1pt)

```
function [acc_curve] = knn_eval( X_train, y_train, X_test, y_test, k_range )
%KNN_EVAL Implementation of kNN evaluation.
%   input ----------------------------------------------------------------
%       o X_train   : (N x M_train), a data set with M_test samples each ...
    being of dimension N.
%                           each column corresponds to a datapoint
%       o y_train   : (1 x M_train), a vector with labels y corresponding to ...
    X_train.
%       o X_test    : (N x M_test), a data set with M_test samples each ...
    being of dimension N.
%                           each column corresponds to a datapoint
%       o y_test    : (1 x M_test), a vector with labels y corresponding to ...
    X_test.
%       o k_range   : (1 X K), Range of k-values to evaluate
%
%   output ----------------------------------------------------------------
%       o acc_curve : (1 X K), Accuracy for each value of K
```

Test Implementation

By running the **sixth** code block, you can directly test **knn_eval.m** function and visualize your plots . The output of your function must be the accuracy curve, if implemented correctly you should be able to plot Figure 12 for the 2D-Concentric Circle dataset.

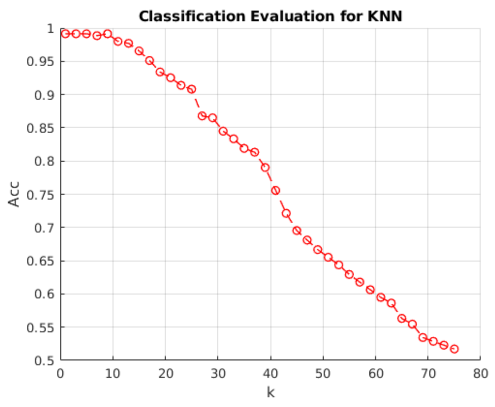


Figure 12: Classification Evaluation for KNN ($k = 1, \ldots, 75$) on the 2d-Concentric-Circle Dataset. Optimal $k \approx 1 \rightarrow 10$. `valid_ratio=0.7`

Further, the encrypted `test_knneval.p` function will compare the estimated accuracy curve from your `knn_eval.m` function with the `ML_toolbox` function. If you achieve the following, you can move on to part 2:

```
--- Testing my_accuracy.m ---
[Test] Accuracy implementation: Correct.
```

2 Part 2: Classification with K-Nearest Neighbors (KNN)

In this second part of the assignment we will apply KNN algorithm on two different datasets. One with high-dimensional continuous data, the Breast-Cancer-Wisconsin (Diagnostic) Dataset[2] and a second which mixes continuous and categorical data, the Adult Dataset[3].

K-NN for classification of high dimensional data

The Breast-Cancer-Wisconsin (Diagnostic) Dataset is composed of $M = 698$ data points of $N = 9$ dimensions, each corresponding to cell nucleus features in the range of $[1, 10]$, with data points belonging to two classes $y \in \{\text{benign}, \text{malignant}\}$. In this part, we will apply KNN classification on this dataset and develop evaluation tools to assess the performances of the classifier. Load the Breast-Cancer-Wisconsin (Diagnostic) Dataset by running the **first** block of code in `test_knn_9D.m`. By running the **second** and **third** blocks you will split the datasets with your `split_data.m` and evaluate the classification accuracy with the function `knn_eval.m`.values that should generate a plot similar to Figure **??**.

2.1 Confusion matrix

For real high-dimensional datasets, such as the Breast-Cancer-Wisconsin (Diagnostic) Dataset, estimating the classification accuracy is not a sufficient statistic to evaluate the classifiers performance. In these cases, one commonly computes a **confusion matrix** or error matrix, which is a specific table to visualize the performance in terms of classification of an algorithm. In this table, the rows represents the real classes of the data and the columns the estimated classes. The diagonal represents the well classified examples while the rest indicates confusions. In the case of a binary classifier (i.e. two classes such as the Breast-Cancer-Wisconsin Dataset), 4 values need to be computed to fill the confusion matrix. One of the labels is considered as positive and the other one as negative.

- **True positives (TP)**: number of test examples with a positive estimated label for which the actual label is also positive (good classification)
- **True negatives (TN)**: number of test examples with a negative estimated label for which the actual label is also negative (good classification)
- **False positives (FP)**: number of test examples with a positive estimated label for which the actual label is negative (classification errors)
- **False negatives (FN)**: number of test examples with a negative estimated label for which the actual label is positive (classification errors)

[2] The Breast-Cancer-Wisconsin (Diagnostic) Dataset can be found in the UCI Machine Learning Repository: `https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29`

[3] The Adult Dataset can be found in the UCI Machine Learning Repository: `https://archive.ics.uci.edu/ml/datasets/adult`

Table 1: Confusion matrix

		Estimated labels	
		Positive	**Negative**
Real labels	**Positive**	True positives (TP)	False negatives (FN)
	Negative	False positives (FP)	True negatives (TN)

TASK 4: Implement confusion_matrix.m function (2pts)
Write a function that computes the confusion matrix of a binary classifier, given the real labels of the test dataset `y_test` and the labels estimated by the classifier `y_est`. NOTE: For this 2 class dataset, a positive label is when $\mathbf{y} = \mathbf{2}$ and a negative label is when $\mathbf{y} = \mathbf{1}$.

```
function [C] =  confusion_matrix(y_test, y_est)
%CONFUSION_MATRIX Implementation of confusion matrix
%   for classification results.
%   input -----------------------------------------------------------------
%
%       o y_test    : (1 x M), a vector with true labels y
%                     corresponding to X_test.
%       o y_est     : (1 x M), a vector with estimated labels y
%                     corresponding to X_test.
%
%   output ----------------------------------------------------------------
%       o C           : (2 x 2), 2x2 matrix of |TP & FN|
%                                              |FP & TN|.
```

Test Implementation

To test that the implementation if your confusion matrix is correct, run the **fourth** code block, the encrypted `test_confusionmatrix.p` function will compare the estimated confusion matrix from your `confusion_matrix.m` function with the `ML_toolbox` function. If you achieve the following, you can move on to the next task:

```
    --- Testing confusion_matrix.m ---
    [Test] Confusion Matrix implementation: Correct.
```

2.2 ROC curve

Based on the values of the confusion matrix, one can estimate a graphical representation of the classifier performance, the Receiver Operating Characteristic, so called ROC curve. The ROC curve plots the fraction of true positives and false positives over the total number of samples of class $\mathbf{y}$ (positive,negative) in the dataset. Each point on the curve corresponds to a different value of the classifier's parameter (e.g. k). Figure 13 illustrates this concept.
To compute such a curve, one needs the **TPR** (True Positive Rates), otherwise known as *sensitivity* or *recall* and **FPR** (False Positive Rates), these are computed as follows [2]:

- **True Positive Rate (TPR)**:

$$TPR = \frac{TP}{TP + FN} = \frac{TP}{P} \tag{6}$$

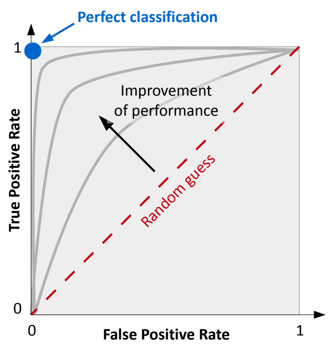


Figure 13: Illustration of the ROC curve.

- **False Positive Rate (FPR)**:

$$FPR = \frac{FP}{FP + TN} = \frac{FP}{N} \tag{7}$$

Where P are all positive values ($\mathbf{y = 2}$) and N are all negative values ($\mathbf{y = 1}$) [2].

TASK 5: Implement knn_ROC.m function (2pts)
This function shall compute the values for the ROC curve of K-NN for a range of k-values. This function must return two $(1 \times K)$ vectors, the first vector is the TPR of the classifier for each value of k and the second vector is the FPR. The columns correspond to the k-values.

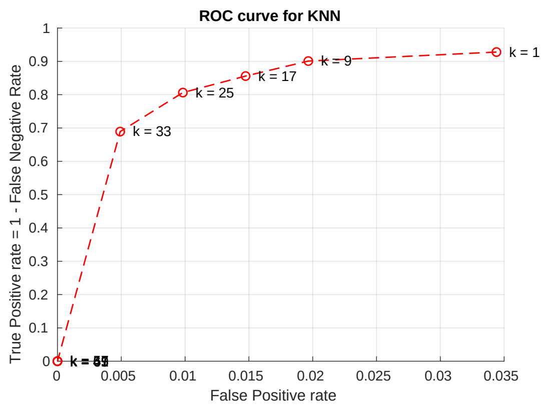
```
function [ TP_rate, FP_rate ] = knn_ROC( X_train, y_train, X_test, y_test, ...
    k_range )
%KNN_ROC Implementation of ROC curve for kNN algorithm.
%   input -----------------------------------------------------------------
%       o X_train  : (N x M_train), a data set with M_test samples each being
%                    of dimension N each column corresponds to a datapoint
%       o y_train  : (1 x M_train), a vector with labels y corresponding to ...
    X_train.
%       o X_test   : (N x M_test), a data set with M_test samples each being
%                    of dimension N each column corresponds to a datapoint
%       o y_test   : (1 x M_test), a vector with labels y corresponding to ...
    X_test.
%       o k_range  : (1 x K), Range of k-values to evaluate
%
%   output ----------------------------------------------------------------

%       o TP_rate  : (1 x K), True Positive Rate computed for each value of k.
%       o FP_rate  : (1 x K), False Positive Rate computed for each value of k.
```

Test Implementation

The ROC curve for a binary classification problems is then defined as the two dimensional plot by representing **FPR** on its x-axis against **TPR** (sensitivity) on the y-axis. By running the **fifth** code block in `test_knn_9D.m` testing script you will be able to plot the resulting ROC curve. Rerun this code block a few times, using the same `valid_ratio = 0.9` to display the ROC curve for different randomizations of train and test datasets. You should obtain figures similar to Figure 14. To check that the implementation if your ROC curve is correct, the encrypted `test_knnROC.p` function will compare the estimated ROC curve for K-NN from your functions with the `ML_toolbox` function. If you achieve the following, you can move on to the next task:

```
--- Testing knn_ROC.m ---
[Test 1] TPR for KNN implementation: Correct.
[Test 2] FPR for KNN implementation: Correct.
```



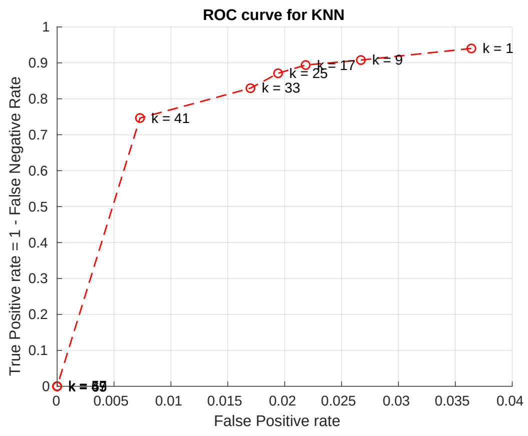


Figure 14: Two examples of ROC curves for K-NN generated with different randomization of train and test datasets for the same range of k-values. `valid_ratio=0.9` in the 2 cases.

2.3 Cross-validation

One can observe that the obtained ROC curve is very sensitive to the train dataset that is extracted by the function `split_dataset.m`. This observation is particularly true for kNN, because each example of the train set is directly used for classification. To assess the performances of the algorithm in a more robust way, a solution is to use cross-validation.
"Cross-validation refers to the practice of confirming an experimental finding by repeating the experiment using an independent assay technique" (Wikipedia). In Machine Learning, this consists of splitting the dataset several times at random into training and validation sets. The number of repetitions is referred to as the number of folds F. When one proceeds to 10 such random draws of training and testing sets, one says that one performs 10-fold cross-validation.

Here, we will use the function `split_data.m` to split the data F times at random between train and test while keeping the same `valid_ratio`. By averaging the resulting ROC curve across F, we will obtain a more robust assessment of the performance of our algorithm.

We will also compute the standard deviation σ across cross-validation runs using matlab function `std.m`. The standard deviation shows how the computed averages can vary (note that in normal distributions, 68.2% of the distribution takes place between $\pm\sigma$).

TASK 6: Implement cross_validation.m function (4pts)

```
function [TP_rate_F_fold, FP_rate_F_fold, std_TP_rate_F_fold, ...
    std_FP_rate_F_fold] =  cross_validation(X, y, F_fold, valid_ratio, k_range)
%CROSS_VALIDATION Implementation of F-fold cross-validation for kNN algorithm.
%
%   input -----------------------------------------------------------------
%
%       o X          : (N x M), a data set with M samples each being of ...
    dimension N.
%                          each column corresponds to a datapoint
%       o y          : (1 x M), a vector with labels y corresponding to X.
%       o F_fold     : (int), the number of folds of cross-validation to compute.
%       o valid_ratio  : (double), Training/Testing Ratio.
%       o k_range    : (1 x K), Range of k-values to evaluate
%
%   output ----------------------------------------------------------------
%
%       o TP_rate_F_fold  : (1 x K), True Positive Rate computed for each ...
    value of k averaged over the number of folds.
%       o FP_rate_F_fold  : (1 x K), False Positive Rate computed for each ...
    value of k averaged over the number of folds.
%       o std_TP_rate_F_fold  : (1 x K), Standard Deviation of True Positive ...
    Rate computed for each value of k.
%       o std_FP_rate_F_fold  : (1 x K), Standard Deviation of False ...
    Positive Rate computed for each value of k.
%
```

Test Implementation

Run the **last** code block in `test_knn_9D.m` to plot the resulting ROC curve created from F-fold cross-validation. You should obtain a curve similar to the Fig.15, which is a plot of the ROC curve displaying standard deviations for each k across cross-validation runs.

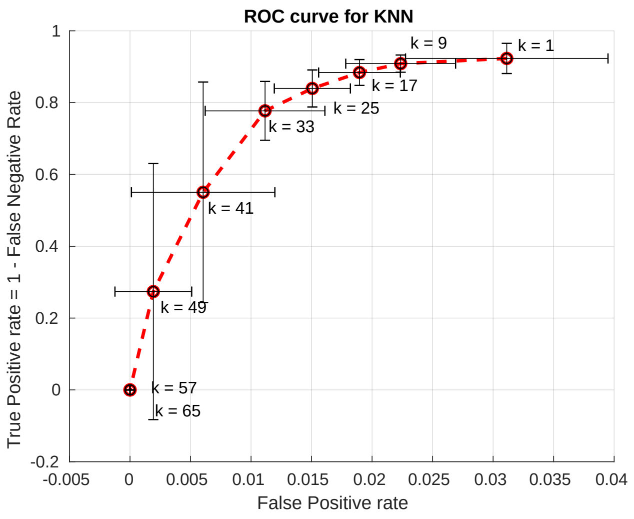


Figure 15: ROC curve with standard deviations for kNN with 10-fold cross-validation. `valid_ratio=0.9`

To check that the implementation if your ROC curve through cross-validation is correct, the encrypted `test_cross_validation.p` function will compare your functions with the `ML_toolbox` functions. If you achieve the following, you can move on to the next task:

```
--- Testing cross_validation.m ---
[Test] Cross Validation implementation: Correct.
```

KNN for classification of mixed continuous and categorical data

Previously, we classified data points which presented continuous feature, i.e. the distance between two points could be calculated using a N-dimensional euclidean distance. However, KNN makes no assumption on the underlying structure of the data which means that it is also a good choice for classifying points that present categorical features, such as gender or nationality. The Adult Dataset, for example, contains a combination of $N = 14$ features, 6 continuous (age, fnlwgt, education, capital_gain, capital_loss, and hours_per_week) and 8 categorical (workclass, marital_status, occupation, relationship, ethnic_origin, sex, native_country, and salary_class). For this exercise, we will try to classify whether an American person as a yearly salary below or above $50K\$$ (corresponding to the salary_class feature) based on all the other features. All the dataset is loaded from a `csv` file stored in a Matlab table `training_data`.

2.4 Pre-processing

In any machine learning problem, this is one of the most important step. You need to ensure that your choice of algorithm can be applied to the data at hand using `transformations`. One common transformation is to replace categories in categorical data with an integer value, e.g. male and female of the sex feature can be replaced by 1 and 2. Another common transformation is to normalize continuous data between 0 and 1. To help you decide whether a feature is continuous or categorical, a boolean cell array `data_type` has been created. It contains `true` when the feature is continuous. Because this dataset contains a very large number of samples, we will only work with a random subset of it, calculated for the `ratio` value, e.g. if $ratio = 0.2$ you will randomly select 20% of the total number of samples.

Therefore, for this exercise, do the following:

- Extract only a certain number of samples based on `ratio` input.
- Replace categorical data by an integer value.
- Calculate the range (difference between minimum and maximum values) R_k of continuous data. This value will be used in the distance calculation instead of normalizing the data directly.
- Separate features data from classification label (the salary_class feature) and store them in X and Y arrays respectively.
- Return the X and Y arrays in the correct shape for using them directly in your KNN implementation.

NOTE: Because you work on a subset of the samples, be careful to calculate the range of continuous values on this subset and not on the whole table.

TASK 7: Implement preprocess_data.m function (1pts)

```
function [X, Y, rk] = preprocess_data(table_data, ratio, data_type)
%PREPROCESS_DATA Preprocess the data in the adult dataset
%
%   input -----------------------------------------------------------------
%
%       o table_data    : (M x N), a cell array containing mixed
%                         categorical and continuous values
%       o ratio : The pourcentage of M samples to extract
%
%   output ----------------------------------------------------------------
%       o X : (N-1, M*ratio) Data extracted from the table where
%             categorical values are converted to integer values
%       o Y : (1, M*ratio) The label of the data to classify. Values are 1
%             or 2
%       o rk : (N-1 x 1) The range of values for continuous data (will be 0
%               if the data are categorical)
```

Implementation Hint: Useful functions: grp2idx(), randperm(), floor().

Test Implementation

Run the **first** code block in `test_adult_dataset.m`. If your implementation of the pre-processing is correct you should see the following:

```
--- Testing pre_processing.m ---
[Test] Pre-processing dimensions: Correct.
[Test] Pre-processing implementation: Correct.
```

2.5 Distances in mixed categorical and continuous data

Now that you have pre-processed your data, you could apply directly your implementation of KNN using euclidean distance as a metric. By doing so you would introduce some bias when comparing values coming from categorical features. Consider as an example the native_country feature. The first possible values are United-States, Cambodia, England, Puerto-Rico, and Canada. The label for those values could be ranging from 1 to 5. An euclidean distance on this feature would mean that a Cambodian is as closed to an American than a Puerto-Rican is to a Canadian. Therefore, you would create an order that is non-meaningful. But the question is, how to define a distance between them in this case, especially when you are mixing many different categorical features.

J.C Gower has introduced a measure to estimate the similarity between two samples x_1 and x_2 with mixed categorical and continuous values[3]. For a given feature F_k with $k \in N$, the measure of similarity S_k is defined as follow:

$$S_k = \begin{cases} 1 - \frac{|x_{1k} - x_{2k}|}{R_k} & \text{if} \quad F_k \quad \text{is continuous} \\ (1 \quad \text{if} \quad x_{1k} = x_{2k}, \quad 0 \quad \text{otherwise}) & \text{if} \quad F_k \quad \text{is categorical} \end{cases} \tag{8}$$

The final similarity measure S is obtained as $S = \frac{\sum_k^N S_k}{N}$. This measure ensures that no bias are introduced when comparing categorical values, and continuous values of different ranges are normalized. Note that this is a similarity measure, i.e. the higher the value the closer two samples are. As KNN uses distances in `my_distance` function, the dissimilarity measure is calculated as $1 - S$. You can call this distance by passing the argument `Gower` in `my_distance` function.

TASK 8: Implement gower_similarity.m function (1pts)

```
function [S] = gower_similarity(X1,X2,data_type, rk)
%GOWER_SIMILARITY Compute the Gower similarity between X1 and X2
%
%   input -----------------------------------------------------------------
%       o X1 : (N x 1) First sample point
%       o X2 : (N x 1) Second sample point
%       o data_type : {N x 1}, a boolean cell array with true when
%                     feature is continuous
%       o rk : (N x 1) The range of values for continuous data
%
%   output ----------------------------------------------------------------
%      o S : The similarity measure between two samples
```

Test Implementation

Run the **second** code block in `test_adult_dataset.m`. If your implementation of the `gower_similarity` is correct you should see the following:

```
   --- Testing gower_similarity.m ---
   [Test] Gower similarity: Correct.
```

After the test, KNN is run with multiple values of k and its accuracy is plotted in Fig.16. You should obtain a similar curve in the **last** code block in `test_adult_dataset.m`.

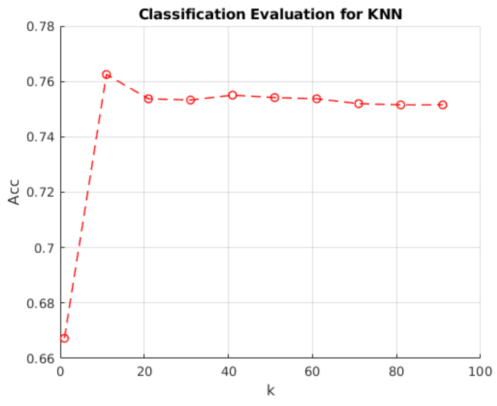


Figure 16: Classification Evaluation for KNN ($k = 1, \ldots, 91$) on the Adult Dataset. `valid_ratio=0.7`

References

[1] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000.

[2] Matjaz Majnik and Zoran Bosnic. Roc analysis of classifiers in machine learning: A survey. *Intell. Data Anal.*, 17(3):531–558, 2013.

[3] John C Gower. A general coefficient of similarity and some of its properties. *Biometrics*, pages 857–871, 1971.