IMT Atlantique
Départament Informatique
Technopôle de Brest-Iroise - CS 83818
29238 Brest Cedex 3
URL: **www.imt-atlantique.fr**

TAF MCE - UE A

Machine Learning Project - Report

Martina María BALBI ANTUNES
Mateo BENTURA LARREGUI
Ezequiel Tomás CENTOFANTI
Kevin MICHALEWICZ

Date d'édition : 2021-12-15

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Contents

List of Figures

List of Tables

1. Introduction

This report describes our final work for the UE Machine Learning of IMT Atlantique. Throughout it, we present several techniques to solve the classification problem for the Banknote Authentication and Chronic Kidney Disease datasets.

Some of the objectives of this project are to use standard development tools, work collaboratively, develop good programming practices and become familiar with Machine Learning datasets.

First of all, the datasets will be cleaned and normalized. We will proceed by using the Principal Component Analysis (PCA) technique for dimensionality reduction. After that, the following Machine Learning methods will be applied: K-nearest Neighbors (KNN), Support Vector Machines (SVM), Gaussian Mixture Models (GMM) and Neural Networks (NN). Finally, the results will be analyzed, a section on good programming practices will be included and conclusions will be drawn. The complete code and the Git log can be found in the appendix.

2. Datasets

2.1. Banknote Authentication

The Banknote Authentication dataset [1] contains 1372 records of several banknotes. The data present was extracted from images of each banknote being some genuine and some fake. These images are of $400x400$ pixels, gray-scaled and with a resolution of about $660dpi$. Then, the images were transformed using the Wavelet Transform tool and the following information, the features present in the dataset, were extracted from them: Variance, Skewness, Curtosis and Entropy.

2.1.1. Preprocessing

Fortunately, this dataset is 100% complete, meaning that all examples contain all the features and they are all already floats. Therefore, the dataset was only imported, the column names were added for simplicity, the labels were extracted and finally the data was normalized.

2.2. Chronic Kidney Disease

The Chronic Kidney Disease dataset [2] was obtained after medical studies in India which lasted two months. 25 features, being some of them categorical and other numerical, may help to predict if a given patient possesses the disease or not. There are 400 rows.

2.2.1. Preprocessing

The data preprocessing stage for this dataset was more complicated than in the previous case. The first thing to do was to convert the data in the *packed_cell_volume*, *white_blood_cell_count* and *red_blood_cell_count* columns to numeric type, leaving *NaNs* in case of conversion errors. Afterwards, categorical and numerical columns were clearly separated. In this sense, *NaN* values were replaced by the mean of valid amounts from the same feature in the case of numerical columns, and by the the most frequent binary label for categorical columns. In addition, tabs and blanks had to be removed.

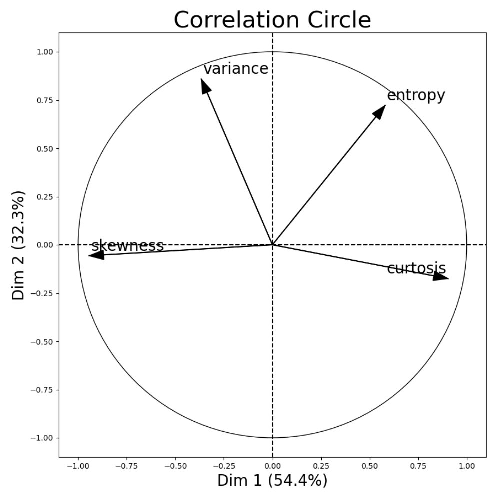
At this point, the normalization of the numerical features was performed and the categoricals were casted to *True* or *False* with pandas *get_dummies()* method. Finally, the classes *y* (diseased or not) were extracted and return as a separate vector.

3. Principal Component Analysis

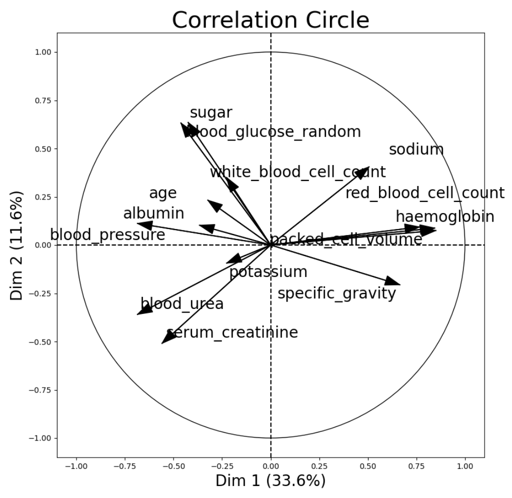
Principal Component Analysis (PCA) is an unsupervised learning method that simplifies the complexity of high-dimensional sample spaces while preserving their information.

Using *plot_pca_correlation_graph* it was possible to obtain the correlation circles of the different datasets, which are shown in Figure 1.

The easiest case to interpret due to the number of features is the Banknote Authentication dataset. It is evident that skewness and curtosis can be described by the first dimension of PCA. The same can be said for variance and entropy looking at the second component. Furthermore, by analyzing the direction of the



(a) Banknote Authentication dataset.



(b) Kidney Disease dataset.

Figure 1: Correlation circles for both datasets.

arrows it is possible to understand the correlations of the features. Finally, the percentage of the information (variance) represented by each of the two components can be read on the axes.

PCA was done in a very similar way in both datasets, except that in Kidney Disease categorical variables were not processed because many authors recommend excluding them in this regard.

After gaining an intuition regarding how the features and their correlations behave, different component values for the PCA algorithm were tested. It was chosen to retain more than 85% of the variance in both cases. In particular:

- For the Banknote Authentication dataset 2 components that represent 0.87 of the original variance were considered.
- For the Kidney Disease dataset 10 numerical components that represent 0.92 of the original variance were considered.

New datasets with dimension reduction were obtained at the output, being ready to be treated with the Machine Learning algorithms that will be described below.

4. K-Nearest Neighbors

The K-Nearest Neighbors classifier is an algorithm based on a set of data for training and for evaluation of a model, of a supervised type. The latter means that the training data includes the desired solution (labels).

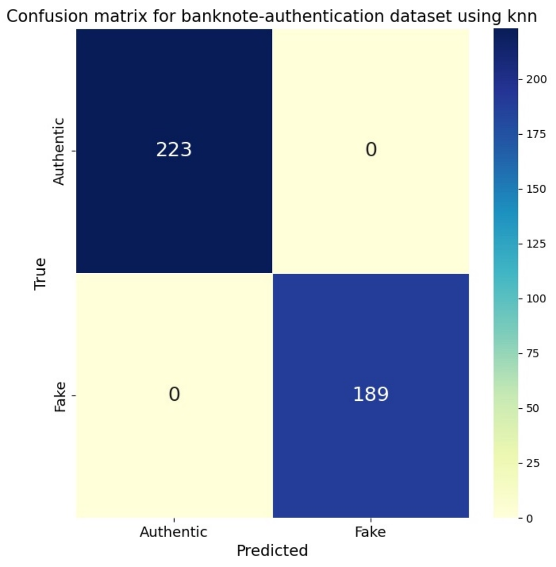
This classifier is a method that considers the samples closest to the one to be predicted. Then, from these samples, it classifies the new data of interest based on the majority of data around it. This closeness depends on the variable K (integer), which refers to the number of neighbors to be considered.

In order to choose the best value of K, the accuracy of the model was tested for $K = 1$ to $K = 7$. The value of K that yielded the best results was used for the final model. These results are presented in the following Table 1:

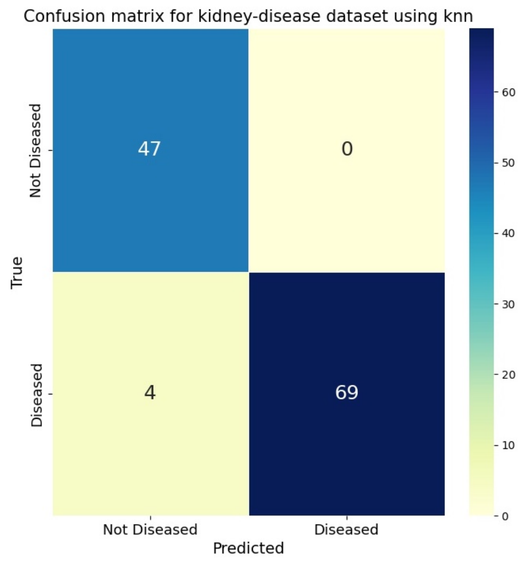
Taking into consideration the results presented above, the best value of K appears to be $K = 5$. The confusion matrix for each dataset is shown in Figure 2.

	K=1	K=2	K=3	K=4	K=5	K=6	K=7
banknote-authentication	100.00	100.00	100.00	100.00	**100.00**	99.51	99.51
kidney-disease	95.00	95.00	95.00	95.00	**96.67**	95.00	95.00

Table 1: Accuracy in both datasets of KNN for different values of K.



(a) Banknote Authentication dataset.



(b) Kidney Disease dataset.

Figure 2: Confusion matrices for classification of both datasets using KNN ($K = 5$).

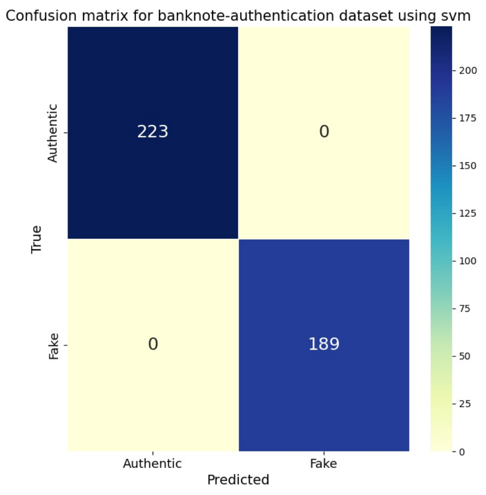
This model correctly classifies all examples of the banknote authentication dataset and almost all of the kidney disease dataset. However, these 4 misclassified examples of the kidney disease are not unimportant, since they represent 4 patients that were diagnosed as not diseased when they actually had a kidney disease. For this type of applications, these kind of errors are very problematic. Therefore, it is to be concluded that while this model appears to be excellent for proving the authenticity of banknotes, it is not very good for diagnosing patients with kidney disease.

5. Support Vector Machines

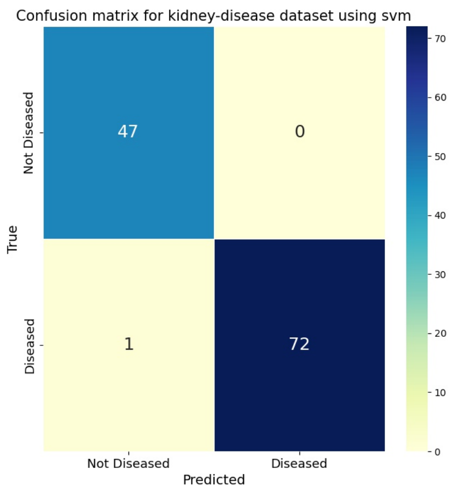
Support Vector Machines is a supervised learning technique that searches for a hyperplane to separate two classes of points, including a margin to maximize the minimal distance. It is a quite simple calculation as it depends just on the support points. Given the case in which the clouds are not linearly separable, we can limit the error by adding the ξ_i coefficients that represent the distance between wrongly classified individuals and the margin we are willing to accept. Another option is to introduce the *Kernel Trick* and work with transformations to spaces of larger dimensions.

The scikit-learn library was used and the model was fitted by using `sklearn.svm.SVC()` with default parameters.

The confusion matrix for each dataset is shown in Figure 3.



(a) Banknote Authentication dataset.



(b) Kidney Disease dataset.

Figure 3: Confusion matrices for classification of both datasets using SVM.

The results are fairly acceptable due to the fact that there are hits for both datasets in all predictions, except in one case corresponding to a false negative in Kidney Disease.

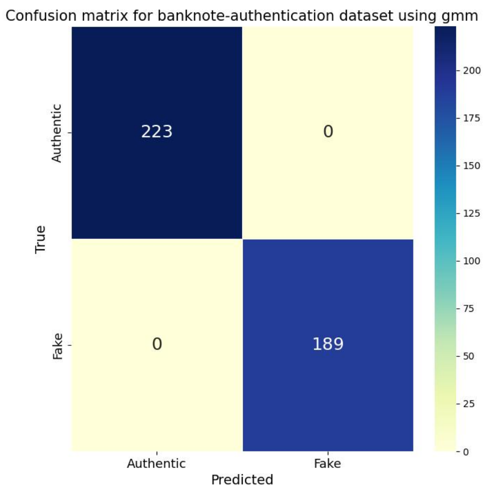
6. Gaussian Mixture Models

We applied the technique known as Gaussian Mixture Models (GMM) to both datasets. This is an unsupervised model that fits data to a mixture of Gaussian distributions using the expectation-maximization algorithm, but a supervised approach was taken: two Gaussian mixture model are trained, one for each class in order to model their probability distribution. In order to make a prediction, the test samples' likelihood are evaluated using both models and the one with a greater likelihood is chosen.

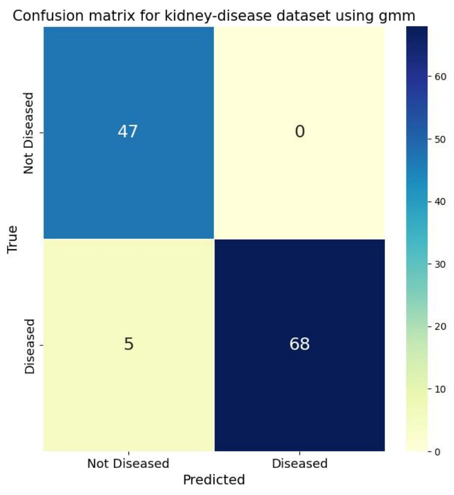
The resulting parameter of this classifier is the number of mixture components used to model the data in each class. In the case of the BA dataset, the number used was 2 mixture components, where as only one sufficed for the KD dataset (i.e. each class is modelled as a Gaussian distribution).

The model was implemented using the class `GaussianMixture` from the scikit-learn module `sklearn.mixture`.

The resulting confusion matrices for this model are shown in Figure 4. We conclude this is a fairly good result, given the simplicity of this classifier.



(a) Banknote Authentication dataset.



(b) Kidney Disease dataset.

Figure 4: Confusion matrices for classification of both datasets using GMM.

7. Neural Networks

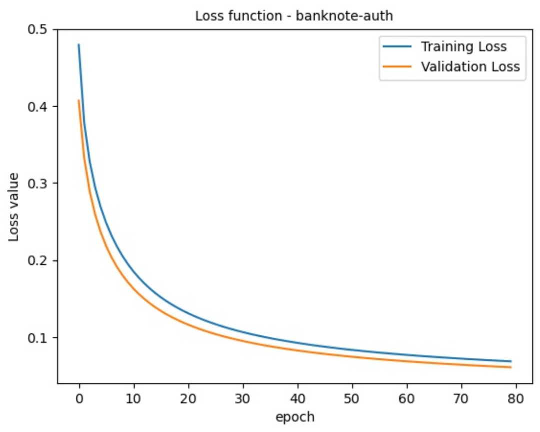
Another method used for classification is the application of neural networks. For both datasets (CKD and BA) the data consists of a set of features without any apparent spatial relationship, therefore fully connected neural networks are used and not convolutional or other types.

In general, such methods tend to tackle much more complex and higher dimensional problems. Yet we can easily implement them for this classification problem.

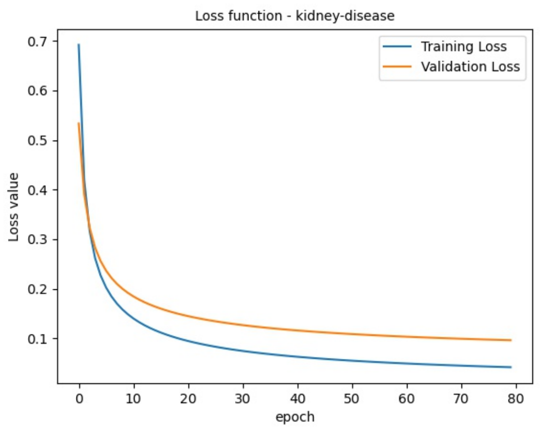
The most basic hyperparameters to determine in this type of network are the number of layers to be used and the number of nodes or neurons per layer. This is by no means a simple task in general and in most cases the answer is a bit of experience and experimentation.

Since the problem to be solved is relatively simple, it is tested with a single layer, whose number of inputs will be the number of features according to the dataset and with a single output. The number of parameters to train is equal to the number of features plus one, the latter corresponding to the bias of the output neuron. In this approach we are trying to separate the data by a hyperplane in feature space.

For both datasets the binary cross-entropy is used as loss function and the optimisation method used is stochastic gradient descent. The number of epochs was set at 80 for both datasets with a learning rate of 0.05. A validation set of size 25% of the training set is used to determine the number of epochs. Figure 5 shows the losses over the training and validation sets for each epoch for each dataset.



(a) Banknote Authentication dataset.



(b) Kidney Disease dataset.

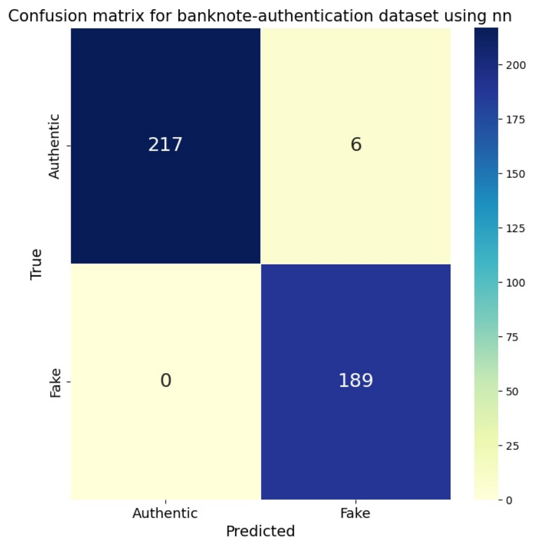
Figure 5: Loss over training and validation sets for each epoch.

The number of epochs is chosen approximately where the loss function does not decrease considerably with increasing number of epochs. Normally we would observe that at some point the validation loss increases while the training loss continues to decrease slightly, at this point we would start to overfit the data and we should stop, but we will not see that in this case, as the model is too simple for overfitting.

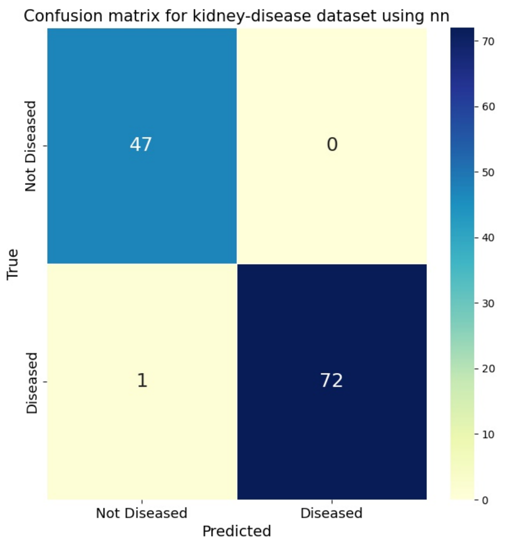
	Accuracy
banknote-authentication	98.00
kidney-disease	99.00

Table 2: Accuracy in both datasets for NN classification.

The results of this method are presented in Table 2 and the confusion matrix is shown in Figure 6. The two neural networks correctly classify data from both datasets, even with the simplest possible network model.



(a) Banknote Authentication dataset.



(b) Kidney Disease dataset.

Figure 6: Confusion matrices for classification of both datasets using NN.

8. Results

In order to evaluate the performance of the four models used to classify, the confusion matrices were generated (as seen in each section). Also, precision, recall and the F1-scores were computed. These scores are presented in the following Table 3:

	Banknote Authentication				**Kidney Disease**			
	KNN	**SVM**	**GMM**	**NN**	**KNN**	**SVM**	**GMM**	**NN**
Precision	1.00	1.00	1.00	0.97	1.00	1.00	1.00	1.00
Recall	1.00	1.00	1.00	1.00	0.95	0.99	0.93	0.96
F1-score	1.00	1.00	1.00	0.98	0.97	0.99	0.96	0.98

Table 3: Scores used to evaluate the classifiers.

As it can be seen in the table above, all models performed very well in both datasets.

9. Good Programming Practices

This work included a collaborative development of Machine Learning functions with an active use of Git. For the code to be readable, easily interpretable by members of the same group and third parties, and for the different modules to have a reasonable organization, it was necessary to adopt good programming practices. These will be detailed hereafter:

To make the code look as homogeneous as possible, we decided to use the PEP 8 Python Code Style Guide[1] before starting to program. Among other things, there one can see how to place function arguments, the correct format for loops, naming conventions and imports.

On the other hand, in order to make each function instantly understandable, a long comment was included in each of them detailing: what is its role, which are the input arguments and what it returns.

As already mentioned, the project was divided into modules in terms of code. It includes a main program *main.py* that uses *clean_normalize.py* to clean and normalise the input datasets, *ml_functions.py*

[1]https://www.python.org/dev/peps/pep-0008/

contains the functions associated with the Machine Learning methods, *tools.py* carries general tools such as the lines of code that have to do with plotting the confusion matrices. This organisation made it easy to work through Git in a collaborative way. In this sense, making recurring and explanatory commits was the adopted approach.

In addition, a *.txt* file with the Python modules required to run the project was included as well as a *README.md* to guide users on being able to get results on the command line using our code.

10. Conclusions

In this work, two classification problems were solved by applying different machine learning techniques. First, a study of both problems was carried out in order to understand them and to be able to propose different solutions. Both problems consist of binary classification of data having corresponding real values, i.e. it is a supervised learning problem. The objective is to generate a function or inference rule that allows to assign a class to a new data point never seen before, and ideally that this class is the correct one.

The first step consists of handling the data, which may be encoded in a csv file or other type of file, it is necessary to clean and normalize the data, then we should analyse its structure in order to propose methods to solve the problem. The method applied is PCA, principal component analysis, it gives us the best projection on a lower dimensional subspace in terms of information maintenance, i.e. maximising the variance of the data. In this way we can reduce the number of relevant dimensions of the data by reducing the number of parameters of the solutions to be applied and by reducing the computational effort.

Four methods are proposed to solve both problems: K-Nearest Neighbors, Support Vector Machines, Gaussian Mixture Models and Neural Networks. All the methods used gave relatively good results. It is important to understand the notion of the term "relatively good", because depending on the problem we can tolerate a given number of errors of one of the two types. For example in the case of kidney disease screening, false alarm errors could be tolerated but not miss errors (this is just an example without medical basis); nor would we want to declare a fake bank note as authentic. If we wanted to determine this rate with greater precision we would require a larger number of data points.

Another crucial axis of this project, and perhaps the most important in academic terms, is the adaptation to collaborative work and version control through the git tool. This project allowed us to practice our teamwork skills and to reinforce good programming practices in this type of collaborative work.

Appendices

Appendix 1 – Git Log

```
commit 5771c9ff3797ec1a5aafa79307505c91b08c736f
Author: Martina Balbi <martina.balbi14@gmail.com>
Date:   Wed Dec 15 12:25:20 2021 +0100

    implemented f1 scores

commit 220ee7f6de5a9688e9e8b85f91c6fc156d82d121
Merge: afd3be8 655db45
Author: Martina Balbi <martina.balbi14@gmail.com>
Date:   Wed Dec 15 11:51:18 2021 +0100

    Merge branch 'main' of https://github.com/kevinmicha/ML-IMTA-Project

commit afd3be81c47e37de9887b3c6c7d7cacbe2031998
Author: Martina Balbi <martina.balbi14@gmail.com>
Date:   Wed Dec 15 11:48:55 2021 +0100

    implemented f1 scores

commit 655db45e6f88195d3f372dfd26d2122261ec033b
Author: Martina Balbi <martina.balbi14@gmail.com>
Date:   Wed Dec 15 11:48:55 2021 +0100

    Merge branch 'main' of https://github.com/kevinmicha/ML-IMTA-Project

commit ebb3d16cc89f35b3b6bb8bce791b21b8a0e1844e
Author: Mateo Bentura <mateo.bentura-larregui@imt-atlantique.net>
Date:   Tue Dec 14 18:12:46 2021 +0100

    supervised GMM implementation

commit 478ba8678d787544932198efe6d407adb6852146
Author: Mateo Bentura <mateo.bentura-larregui@imt-atlantique.net>
Date:   Mon Dec 13 17:54:00 2021 +0100

    fixed function comment

commit 361a74c99107fc438c7eb9266747dd06abf0efea
Author: Mateo Bentura <mateo.bentura-larregui@imt-atlantique.net>
Date:   Mon Dec 13 17:47:26 2021 +0100

    plotted GMM covariances using PCA to gain insight into bad results

commit 1f3a9030359b3e03d0153c333829afacd435b4a9
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Fri Dec 10 12:12:30 2021 +0100
```

```
    new plots

commit 22a44717b046a01b31b9c05041fd982b24e3f629
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Fri Dec 10 12:12:20 2021 +0100

    confusion matrix pipeline is ready

commit fc231078ab63f3a2e20a97726e013e1467edc82e
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Fri Dec 10 11:56:15 2021 +0100

    adding loss plots for nn

commit a0647b6229eaab6c2c0ec83fa4fa44fd20fb9bbe
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Fri Dec 10 11:55:57 2021 +0100

    adding loss plots for nn

commit b22833cdab62050e6a98f05b7b974f0f538fbbd6
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Fri Dec 10 11:43:57 2021 +0100

    adding confusion matrices and solving typos

commit 884bee6b24dbba83c66da1552e7a3b725a35de88
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Fri Dec 10 11:28:41 2021 +0100

    added seaborn to requirements

commit 27e3effd8312307c97c829e44e229af7582e1b17
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Fri Dec 10 11:24:58 2021 +0100

    uncommenting svm calls in main

commit d3a43fc8c7b8432d45662ad4b236eae125fa2fae
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Fri Dec 10 11:23:59 2021 +0100

    svm function finished

commit e8897781da60fbe088f83e6ac9fb116596fa0344
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Fri Dec 10 11:20:37 2021 +0100

    created svm function
```

```
commit 79fd6562211d4b5f1fa2958ab578944acd0c4d2d
Author: [Ezequiel Centofanti] <[ezecentofanti@gmail.com]>
Date:   Fri Dec 10 00:00:50 2021 +0100

    Crossvalidation testig added

commit e2f1b4f3f51ec2067a7ead5d75b91600aea602bd
Author: Mateo Bentura <mateo.bentura-larregui@imt-atlantique.net>
Date:   Wed Dec 8 13:31:53 2021 +0100

    added Gaussian mixture model classifier

commit e5f86631d5d3c7335e0a0ba2ab6a3e39c30d0d0d
Author: Martina Balbi <martina.balbi14@gmail.com>
Date:   Mon Dec 6 19:02:44 2021 +0100

    typo correction

commit 4f4e66929b08b8f884e349b149ba0ca6a16388f1
Author: Martina Balbi <martina.balbi14@gmail.com>
Date:   Mon Dec 6 18:24:58 2021 +0100

    confusion matrixes for knn

commit b19326365664c1f56740d0407a1c11c3cef55da7
Author: Martina Balbi <martina.balbi14@gmail.com>
Date:   Mon Dec 6 18:15:01 2021 +0100

    updated plot_confusion_matrix function

commit b1ac5e90dee969761d4e725681db6534858464ea
Merge: 73a4c01 0e0e5d4
Author: Martina Balbi <martina.balbi14@gmail.com>
Date:   Mon Dec 6 17:56:36 2021 +0100

    Merge branch 'main' of https://github.com/kevinmicha/ML-IMTA-Project

commit 73a4c010886a49ae63206fcd2d9aaaef6ac7b610
Author: Martina Balbi <martina.balbi14@gmail.com>
Date:   Mon Dec 6 17:54:55 2021 +0100

    added lib folder

commit 0e0e5d4420d11a58bdc240e32e36b47aca473867
Author: Martina Balbi <73940356+martibalbi@users.noreply.github.com>
Date:   Mon Dec 6 17:54:35 2021 +0100

    Delete .DS_Store
```

```
commit 8d86f539b170585346d15123477da39adb6d7288
Author: Martina Balbi <martina.balbi14@gmail.com>
Date:   Mon Dec 6 16:30:28 2021 +0100

    added knn implementation and confustion matrix plot functions

commit 6816368d06db2aa704d1109e04de7ea50f919388
Author: Martina Balbi <martina.balbi14@gmail.com>
Date:   Mon Dec 6 11:30:47 2021 +0100

    removed pycache

commit 53c879c77a854414edaf02f8ee1bcb464972ce37
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Mon Dec 6 00:17:44 2021 +0100

    changed typo in an import

commit ee0e8d4c5304cc74d983f53df1bf18b7515d6ff7
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Mon Dec 6 00:13:31 2021 +0100

    adding torch to requirements

commit 1ccab017902caa9fd0e09f8089bc836e9e1a9fbc
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Mon Dec 6 00:09:30 2021 +0100

    some pep8 details in nn_util.py

commit 7f500be9e71fcaaf497d8d5ca5b1c2ea1c663a73
Author: Kevin Michalewicz <44092360+kevinmicha@users.noreply.github.com>
Date:   Mon Dec 6 00:04:29 2021 +0100

    renaming file NN -> nn

commit bc95181d073ca6f503853e8805efb34c5a16bd56
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Mon Dec 6 00:03:49 2021 +0100

    adding author in pca function

commit 8b2f498cb22f8299536696ce88560adfd141c6a5
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Mon Dec 6 00:03:14 2021 +0100

    NN -> nn in file/function names

commit 26e58abb181e913f85d2b5ff6e842644a1f9d2d7
Author: [Ezequiel Centofanti] <[ezecentofanti@gmail.com]>
```

```
Date:   Sun Dec 5 23:43:29 2021 +0100

    added neural networ classifier

commit c1424deb93a92bd097ac87fe689a6d89896eb7da
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Sun Dec 5 15:05:34 2021 +0100

    changing import locations

commit a2b430f6a32f817952cccd28ba1800cd5920f02f
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Sun Dec 5 15:01:31 2021 +0100

    added right location of datasets

commit 2c9e8a31326dcc03c0ffd353296c02e816fcbdd7
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Sun Dec 5 14:53:01 2021 +0100

    replacing some 'kd' and 'ba' for 'dataset'

commit 6878d0efe5e29dfa4140497a7e9debccdc0aa950
Author: Kevin Michalewicz <44092360+kevinmicha@users.noreply.github.com>
Date:   Sun Dec 5 14:43:47 2021 +0100

    creating requirements file

commit c4e90b41a3d5f49c1c3240997eb0fba7c7d5281b
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Sun Dec 5 14:27:31 2021 +0100

    added functions imports to main

commit 45c68a098dfdc86be5abac05630490b0228dcd74
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Sun Dec 5 14:26:21 2021 +0100

    deleting clean_ba, included elsewhere

commit 853eee3e47ed0e6a60ef90d03b2285e8077b624e
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Sun Dec 5 14:25:47 2021 +0100

    renaming and merging cleaning files

commit d4a0c340232839a31f3950b717f7dc02339751bd
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Sun Dec 5 14:24:41 2021 +0100
```

```
    adding main file

commit e801387d4d97a68f5213cf70f200d92ffaf0919c
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Sun Dec 5 14:15:01 2021 +0100

    uploading ml functions file

commit 3b3c0636b13a587d0c68f4b61e6b646be934f4bb
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Sun Dec 5 13:48:59 2021 +0100

    replaced v1...v4 for actual names

commit ae7832e4c6c07a000a53dd9aa6d48f9b5f62fcfa
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Sun Dec 5 13:45:46 2021 +0100

    adding column names

commit 1ae25f21d4b61a36afbeeedb5f20a1f676a3803c
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Sun Dec 5 13:39:23 2021 +0100

    changed two 'kd' to 'ba'

commit 87bff379426e9891fe5bc0e7e4cdd37685cb2e82
Author: Martina Balbi <martina.balbi14@gmail.com>
Date:   Sat Dec 4 18:46:48 2021 +0100

    added clean_ba function description

commit 37ad4724f9e195faa7d8f9eb4841db38151cd3fb
Merge: b20e52e 002141c
Author: Martina Balbi <martina.balbi14@gmail.com>
Date:   Sat Dec 4 18:44:47 2021 +0100

    Merge branch 'main' of https://github.com/kevinmicha/ML-IMTA-Project

commit b20e52eb11990d2f759fb592d7a979186294236a
Author: Martina Balbi <martina.balbi14@gmail.com>
Date:   Sat Dec 4 18:42:59 2021 +0100

    inlcude banknote authentication dataset cleaning function

commit 002141c0fb91d6db998a146fa48711a3c9fe45f6
Author: kevinmicha <kmichalewicz@fi.uba.ar>
Date:   Sat Dec 4 05:40:01 2021 +0100

    updating basic info of clean_kd func
```

```
commit efc0671d6ff7dba1fc34180062916381b8f89006
Author: Kevin Michalewicz <44092360+kevinmicha@users.noreply.github.com>
Date:   Tue Nov 30 19:17:36 2021 +0100

    first commit! - script cleaning KD dataset

commit e68253e4293d46687e56f9fb9339eb1f39db0f04
Author: Kevin Michalewicz <44092360+kevinmicha@users.noreply.github.com>
Date:   Sat Nov 27 18:41:55 2021 +0100

    Initial commit
```

Appendix 2 – Python files

2.1. main.py

```
# ======================================================================
# This is the main file of a classification project made at IMT
# Atlantique. Authors: Martina BALBI, Mateo BENTURA, Ezequiel
# CENTOFANTI and Kevin MICHALEWICZ.
# ======================================================================
from lib.clean_normalize import *
from lib.ml_functions import *
from lib.nn_util import *
from sklearn.model_selection import train_test_split
from lib.tools import *
import matplotlib.pyplot as plt

# Import and clean datasets
ba = pd.read_csv("datasets/data_banknote_authentication.txt")
kd = pd.read_csv("datasets/kidney_disease.csv").set_index('id')
kd, y_kd = clean_normalize_kd(kd)
ba, y_ba = clean_normalize_ba(ba)

# Perform PCA for the kidney-disease dataset
kd = pca(kd, 'kidney-disease')
# Perform PCA for banknote-authentication dataset, only to be used by GMM
ba_pca = pca(ba, 'banknote-auth')

# Select dataset to fit
data_set_df = ba
target_df = y_ba

# Split datasets into Train and Test
X_train_ba, X_test_ba, y_train_ba, y_test_ba = train_test_split(ba, y_ba,
                                              test_size=0.3, random_state=48)
X_train_kd, X_test_kd, y_train_kd, y_test_kd = train_test_split(kd, y_kd,
                                              test_size=0.3, random_state=48)
```

```
# K Nearest neighbors
y_pred_knn_ba = fit_knn(X_train_ba, X_test_ba, y_train_ba, y_test_ba,
                                                    'banknote-auth')
y_pred_knn_kd = fit_knn(X_train_kd, X_test_kd, y_train_kd, y_test_kd,
                                                    'kidney-disease')

#Support Vector Machines
y_pred_svm_ba = fit_svm(X_train_ba, X_test_ba, y_train_ba, y_test_ba,
                                                    'banknote-auth')
y_pred_svm_kd = fit_svm(X_train_kd, X_test_kd, y_train_kd, y_test_kd,
                                                    'kidney-disease')

# Gaussian Mixture Model
y_pred_gmm_ba = fit_gmm(X_train_ba, X_test_ba, y_train_ba, y_test_ba,
                                                    'banknote-auth')
y_pred_gmm_kd = fit_gmm(X_train_kd, X_test_kd, y_train_kd, y_test_kd,
                                                    'kidney-disease')

# Neural network
y_pred_nn_ba = fit_nn(X_train_ba, X_test_ba, y_train_ba, y_test_ba,
                                                    'banknote-auth')
y_pred_nn_kd = fit_nn(X_train_kd, X_test_kd, y_train_kd, y_test_kd,
                                                    'kidney-disease')

# Plot confusion matrices and get f1 scores
models = ['knn', 'svm', 'gmm', 'nn']
y_pred = [[y_pred_knn_ba, y_pred_knn_kd], [y_pred_svm_ba, y_pred_svm_kd],
          [y_pred_gmm_ba, y_pred_gmm_kd], [y_pred_nn_ba, y_pred_nn_kd]]
y_test = [y_test_ba, y_test_kd]
datasets = ['banknote-authentication', 'kidney-disease']

for i in range(len(models)):
    for j in range(len(y_test)):
        plot_confusion_matrix(y_test[j], y_pred[i][j], models[i], datasets[j])
        get_scores(y_test[j], y_pred[i][j], models[i], datasets[j])
```

2.2. clean_normalize.py

```
# ============================================================================
# This file contains functions that clean and normalize two particular
# datasets for a classification project at IMT Atlantique. Authors:
# Martina BALBI, Mateo BENTURA, Ezequiel CENTOFANTI and Kevin MICHALEWICZ.
# ============================================================================
import pandas as pd
import numpy as np

def clean_normalize_kd(kd):
    '''
    Cleaning and normalizing the Kidney Disease dataset

    INPUT
    kd: The kidney disease dataframe

    OUTPUT
    kd: A cleaned version of the kidney disease dataframe
    y: A vector containing the classes

    AUTHOR
    Kevin Michalewicz
    '''
    # kd = pd.read_csv("kidney_disease.csv").set_index('id')

    kd[['pcv', 'rc', 'wc']] = kd[['pcv', 'rc', 'wc']].apply(pd.to_numeric,
                                                            errors='coerce')

    kd.columns = ['age', 'blood_pressure', 'specific_gravity', 'albumin',
                  'sugar', 'red_blood_cells', 'pus_cell', 'pus_cell_clumps',
                  'bacteria', 'blood_glucose_random', 'blood_urea',
                  'serum_creatinine', 'sodium', 'potassium', 'haemoglobin',
                  'packed_cell_volume', 'white_blood_cell_count',
                  'red_blood_cell_count', 'hypertension', 'diabetes_mellitus',
                  'coronary_artery_disease', 'appetite', 'peda_edema', 'anemia',
                  'classification']

    # Remove NaN
    num_col = kd.columns[kd.dtypes=='float64']
    cat_col = kd.columns[kd.dtypes=='object']

    kd[num_col] = kd[num_col].fillna(kd[num_col].mean())
    kd[cat_col] = kd[cat_col].fillna(kd[cat_col].mode().iloc[0])
    kd[cat_col] = kd[cat_col].replace(to_replace={'\t': '', '␣': ''},regex=True)

    # Normalizing the data
    kd[num_col] = (kd[num_col] - kd[num_col].mean()) / (kd[num_col].std())

    # Cast labels to True or False
    kd = pd.get_dummies(kd, drop_first=True)

    # Extracting classes
    y = kd["classification_notckd"]
    kd = kd.drop(columns="classification_notckd")
    y = np.logical_xor(y,1).astype(int)

    return kd, y

def clean_normalize_ba(ba):
    '''
    Cleaning and normalizing the Banknote Authentification dataset

    INPUT
    ba: the banknote authentication dataset
    OUTPUT
    ba: clean banknote authentication dataset
    y: labels

    AUTHOR
    Martina Balbi
    '''

    # Adding column names
    ba.columns = ['variance', 'skewness', 'curtosis', 'entropy', 'class']

    # Get labels
    y = ba['class']
    ba.drop(columns='class', inplace=True)

    # Normalize data
    ba = (ba - ba.mean())/ba.std()

    return ba, y
```

2.3. ml_functions.py

```
# ==========================================================================
# This file contains some Machine Learning functions used for
# a classification project at IMT Atlantique. Authors: Martina
# BALBI, Mateo BENTURA, Ezequiel CENTOFANTI and Kevin MICHALEWICZ.
# ==========================================================================
from numpy.testing._private.utils import KnownFailureException
import pandas as pd
import matplotlib.pyplot as plt

from lib.nn_util import *

from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from mlxtend.plotting import plot_pca_correlation_graph
from sklearn.mixture import GaussianMixture
from sklearn import svm
from sklearn.metrics import accuracy_score


def pca(dataset, dataset_name):
    '''
    Principal Component Analysis (PCA) function

    INPUT
    dataset: A pandas DataFrame after the cleaning step
    dataset_name: A string containing the dataset name

    OUTPUT
    dataset_after_pca: A PCA-transformed version of the original dataset

    AUTHOR
    Kevin Michalewicz
    '''
    ncomp_kd = 10  # number of components for kidney disease
    ncomp_ba = 2  # number of components for banknote authentication

    if dataset_name == 'kidney-disease':
        numerical_columns = dataset.columns[dataset.dtypes == 'float64']
        categorical_columns = dataset.columns[dataset.dtypes == 'uint8']
        pca = PCA(n_components=ncomp_kd)
        pca.fit(dataset[numerical_columns])
        print('{}␣components␣represent␣{:.2f}␣of␣the␣variance'.format(
            ncomp_kd, sum(pca.explained_variance_ratio_)))
        print('---------------------------------')
        kd_tf_numerical = pca.transform(dataset[numerical_columns])
        dataset_after_pca = pd.concat([pd.DataFrame(
            data=kd_tf_numerical, index=dataset.index),
            dataset[categorical_columns]], axis=1)

    elif dataset_name == 'banknote-auth':
        pca = PCA(n_components=ncomp_ba)
        pca.fit(dataset)
        print('{}␣components␣represent␣{:.2f}␣of␣the␣variance'.format(
            ncomp_ba, sum(pca.explained_variance_ratio_)))
        print('---------------------------------')
        dataset_after_pca = pd.DataFrame(
            pca.transform(dataset), index=dataset.index)

    return dataset_after_pca


def fit_nn(X_train, X_test, y_train, y_test, dataset_name):
    '''
    Neural Network Classifier

    INPUT
    X_train: features for training
    X_test: features for testing
    y_train: targets for training
    y_test: targets for testing
    dataset_name: A string containing the dataset name

    OUTPUT
    y_predicted: predicted labels for the testing set

    AUTHOR
    Ezequiel Centofanti
    '''
    if dataset_name == 'kidney-disease':
        nb_features = 20

    elif dataset_name == 'banknote-auth':
        nb_features = 4

    # Create data-loaders
    train_loader, val_loader, test_loader = create_torch_dataset(
        X_train, X_test, y_train, y_test)

    # Initialize the neural network
    model_ = Net1(nb_features)

    # Specify loss function (categorical cross-entropy)
    criterion = nn.BCELoss()

    # Specify optimizer (stochastic gradient descent) and learning rate
    optimizer = torch.optim.SGD(model_.parameters(), lr=0.05)

    # Train model
    n_epochs = 80  # number of epochs to train the model
    train_losses_1, valid_losses_1 = training(
        n_epochs,
        train_loader,
        val_loader,
        model_,
        criterion,
        optimizer)

    # Plot loss over training
    plot_losses(train_losses_1, valid_losses_1, n_epochs, dataset_name)

    return evaluation(model_, test_loader, criterion, dataset_name)


def fit_knn(X_train, X_test, y_train, y_test, dataset_name):
    '''
    K-Nearest Neighbors classifier

    INPUT
    X_train: features for training
    X_test: features for testing
    y_train: targets for training
    y_test: targets for testing
    dataset_name: A string containing the dataset name

    OUTPUT
    y_predicted: predicted labels for the testing set

    AUTHOR
    Martina Balbi
    '''
    K = 5

    knn = KNeighborsClassifier(n_neighbors=K)
    knn.fit(X_train, y_train)
    y_predicted = knn.predict(X_test)
    accuracy = knn.score(X_test, y_test)

    print('K-Nearest␣neighbors␣test␣accuracy␣for␣dataset␣%s:␣%2d%%␣(%2d/%2d)' %
          (dataset_name, accuracy * 100, accuracy * len(y_test), len(y_test)))
    print('---------------------------------')

    return y_predicted


def fit_gmm(X_train, X_test, y_train, y_test, dataset_name):
    '''
    Gaussian mixture model classifier

    INPUT
    X_train: features for training
    X_test: features for testing
    y_train: targets for training
    y_test: targets for testing
    dataset_name: A string containing the dataset name

    OUTPUT
    y_predicted: predicted labels for the testing set

    AUTHOR
    Mateo Bentura
    '''
    classes = y_test.nunique()

    if dataset_name == 'banknote-auth':
        K = 4

    if dataset_name == 'kidney-disease':
        K = 2

    log_likelyhood = np.zeros(y_test.shape+(classes,))
    for n in range(classes):
        gm = GaussianMixture(n_components=K)
        gm.fit(X_train[y_train==n])
        log_likelyhood[:,n] = gm.score_samples(X_test)

    y_predicted = log_likelyhood.argmax(axis=1)
    accuracy = accuracy_score(y_test, y_predicted)
    print('Gaussian␣mixture␣model␣test␣for␣dataset␣%s:␣%2d%%␣(%2d/%2d)' %
            (dataset_name, accuracy * 100, accuracy * len(y_test), len(y_test)))
    print('--------------------------------')

    return y_predicted

def fit_svm(X_train, X_test, y_train, y_test, dataset_name):
    '''
    Support Vector Machine (SVM) classifier

    INPUT
    X_train: features for training
    X_test: features for testing
    y_train: targets for training (**not used: unsupervised method**)
    y_test: targets for testing
    dataset_name: A string containing the dataset name

    OUTPUT
    y_predicted: predicted labels for the testing set

    AUTHOR
    Kevin Michalewicz
    '''
    clf = svm.SVC()
    clf.fit(X_train, y_train)
    y_predicted = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_predicted)
    print('Support Vector Machine test for dataset %s: %2d%% (%2d/%2d)' %
            (dataset_name, accuracy * 100, accuracy * len(y_test), len(y_test)))
    print('--------------------------------')

    return y_predicted
```

2.4. tools.py

```
# ==================================================================
# This file contains some useful tools for a classification
# project at IMT Atlantique. Authors: Martina BALBI, Mateo BENTURA,
# Ezequiel CENTOFANTI and Kevin MICHALEWICZ.
# ==================================================================
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import seaborn as sns
import numpy as np

def plot_confusion_matrix(y_test, y_pred, model_name, dataset_name):
    '''
    Function to plot and save a confusion matrix
    INPUT
    y_test: test labels
    y_pred: predicted labels
    model_name: name of the classifier used to predict the labels
    dataset_name: name of the dataset

    AUTHOR
    Martina Balbi
    '''
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(8,8))
    heatmap = sns.heatmap(cm,cmap="YlGnBu",linewidths=.5, annot=True,
                          annot_kws={"size": 18}, fmt="d")
    if dataset_name == 'banknote-authentication':
        heatmap.set_xticklabels(['Authentic', 'Fake'], fontsize=13)
        heatmap.set_yticklabels(['Authentic', 'Fake'], fontsize=13)
    elif dataset_name == 'kidney-disease':
        heatmap.set_xticklabels(['Not Diseased', 'Diseased'], fontsize=13)
        heatmap.set_yticklabels(['Not Diseased', 'Diseased'], fontsize=13)
    plt.xlabel('Predicted', fontsize=14)
    plt.ylabel('True', fontsize=14)
    plt.title('Confusion matrix for %s dataset using %s' % (dataset_name,
                                               model_name), fontsize=15)
    plt.savefig("plots/confusion_matrices/Confusion_Matrix_%s_%s.jpg" %
                                               (dataset_name, model_name))

def get_scores(y_test, y_pred, model_name, dataset_name):
    '''
    Function to calculate scores of the model (precision, recall, F1-score)

    INPUT
    y_test: test labels
    y_pred: predicted labels
    model_name: name of the classifier used to predict the labels
    dataset_name: name of the dataset

    AUTHOR
    Martina Balbi
    '''

    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    score = f1_score(y_test, y_pred)
    print('%s F1 Score for dataset %s: %.2f, precision: %.2f, recall: %.2f' %
          (model_name, dataset_name, score, precision, recall))
    print('-------------------------------')

    return
```

2.5. nn_util.py

```
# Import PyTorch
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.utils.data as data_utils
from torch.utils.data.sampler import SubsetRandomSampler

from sklearn.model_selection import train_test_split

from lib.clean_normalize import *
from lib.ml_functions import *

# Define network architecture
class Net1(nn.Module):
    def __init__(self, nb_features):
        super(Net1, self).__init__()
        self.fc1 = nn.Linear(nb_features, 1)
    def forward(self, x):
        out = self.fc1(x)
        out = torch.sigmoid(out)
        return out

def create_torch_dataset(X_train, X_test, y_train, y_test, batch_size=20):
    '''
    Making a torch-type Dataset

    INPUT
    X_train: features for training
    X_test: features for testing
    y_train: targets for training
    y_test: targets for testing
    batch_size: number of samples processed before the model is updated

    OUTPUT
    train_loader: pytorch data-loader for training
    val_loader: pytorch data-loader for validation
    test_loader: pytorch data-loader for testing

    AUTHOR
    Ezequiel Centofanti
    '''
    validation_size = 0.2 # Fraction of the training set

    X_train, X_val, y_train, y_val = train_test_split(
        X_train, y_train, test_size=validation_size, random_state=1)

    X_train_t = torch.Tensor(np.array(X_train))
    y_train_t = torch.Tensor(np.array(pd.DataFrame(y_train)))
    y_train_t = y_train_t.type(torch.LongTensor)
    y_train_t = y_train_t.to(torch.float32)
    data_train = data_utils.TensorDataset(X_train_t, y_train_t)
    train_loader = torch.utils.data.DataLoader(data_train,
                                        batch_size=batch_size)

    X_val_t = torch.Tensor(np.array(X_val))
    y_val_t = torch.Tensor(np.array(pd.DataFrame(y_val)))
    y_val_t = y_val_t.type(torch.LongTensor)
    y_val_t = y_val_t.to(torch.float32)
    data_val = data_utils.TensorDataset(X_val_t, y_val_t)
    val_loader = torch.utils.data.DataLoader(data_val,
                                        batch_size=batch_size)

    X_test_t = torch.Tensor(np.array(X_test))
    y_test_t = torch.Tensor(np.array(pd.DataFrame(y_test)))
    y_test_t = y_test_t.type(torch.LongTensor)
    y_test_t = y_test_t.to(torch.float32)
    data_test = data_utils.TensorDataset(X_test_t, y_test_t)
    test_loader = torch.utils.data.DataLoader(data_test,
                                        batch_size=batch_size)

    return train_loader, val_loader, test_loader

def training(n_epochs, train_loader, val_loader, model,
                                        criterion, optimizer):
    '''
    Training a torch model

    INPUT
    n_epochs: number of training loops over all the dataset
    train_loader: pytorch data-loader for training
    val_loader: pytorch data-loader for validating
    model: torch model to train
    criterion: loss criterion
    optimizer: optimizer method

    OUTPUT
    train_losses: array of train losses at each epoch
    valid_losses: array of validation losses at each epoch

    AUTHOR
    Ezequiel Centofanti
    '''
    train_losses, valid_losses = [], []

    for epoch in range(n_epochs):
        train_loss, valid_loss = 0, 0 # monitor losses
        # train the model
        model.train() # prep model for training
        for data, label in train_loader:
            optimizer.zero_grad() # clear the gradients of
                                  #all optimized variables
            output = model(data) # forward pass: compute
                                 # predicted outputs by
                                 # passing inputs to the model
            loss = criterion(output, label) # calculate the loss
            loss.backward() # backward pass: compute gradient
                            # of the loss with respect to model
                            # parameters
            optimizer.step() # perform a single optimization
                             # step (parameter update)
            train_loss += loss.item() * data.size(0)
                            # update running training loss

        # validate the model
        model.eval()
        for data, label in val_loader:
            with torch.no_grad():
                output = model(data)
            loss = criterion(output,label)
            valid_loss += loss.item() * data.size(0)

        # Calculate average loss over an epoch
        train_loss /= len(train_loader.sampler)
        valid_loss /= len(val_loader.sampler)
        train_losses.append(train_loss)
        valid_losses.append(valid_loss)


    return train_losses, valid_losses


def evaluation(model, test_loader, criterion, dataset):
    '''
    Evaluating a torch model and printing the accuracy
    over the test-set

    INPUT
    model: torch model to train
    test_loader: pytorch data-loader for testing
    criterion: loss criterion

    OUTPUT
    y_predicted: predicted labels for the testing set

    AUTHOR
    Ezequiel Centofanti
    '''
    # initialize values to monitor test accuracy
    pred_correct = 0
    pred_total = 0

    y_predicted = []

    model.eval() # prep model for evaluation
    for data, label in test_loader:
        with torch.no_grad():
            output = model(data) # forward pass: compute
                                 # predicted outputs by
                                 # passing inputs to the model
        pred = output > 0.5
        y_predicted.extend([int(np.array(pred)[i][0])
                            for i in range(len(pred))])
        correct = np.squeeze(pred) == (np.squeeze(label)==1)
        # calculate test accuracy for each batch
        for i in range(len(label)):
            pred_correct += correct[i].item()
            pred_total += 1

    # Calculate and print avg test accuracy
    print('Neural␣network␣test␣accuracy␣for␣dataset␣%s:␣%2d%%␣(%2d/%2d)'
       % (dataset,100 * pred_correct / pred_total, pred_correct, pred_total))
    print('--------------------------------')
    return y_predicted


def plot_losses(train_losses, valid_losses, n_epochs, dataset_name):
    '''
    Ploting training and validation losses to monitor the training

    INPUT
    train_losses: array of train losses at each epoch
    valid_losses: array of validation losses at each epoch
    n_epochs: number of training loops over all the dataset
    dataset_name: A string containing the dataset name

    AUTHOR
    Ezequiel Centofanti
    '''
    plt.figure()
    plt.plot(range(n_epochs), train_losses)
    plt.plot(range(n_epochs), valid_losses)
    plt.legend(['Training␣Loss', 'Validation␣Loss'], prop={'size': 10})
    plt.title('Loss␣function␣-␣%s' % (dataset_name), size=10)
    plt.xlabel('epoch', size=10)
    plt.ylabel('Loss␣value', size=10)
    plt.savefig("plots/nn_loss/loss_%s.jpg" % (dataset_name))
```

References

[1] *Banknonte Authentication dataset*. URL: https://archive.ics.uci.edu/ml/datasets/banknote+authentication.

[2] *Chronic Kidney Disease Dataset*. URL: https://archive.ics.uci.edu/ml/datasets/Chronic_Kidney_Disease.

[3] Lucas Drumetz. *Lecture notes on Bayesian Classification and Gaussian Mixture Models*. IMT Atlantique.

[4] Lucas Drumetz. *Lecture notes on Neural Networks*. IMT Atlantique.

[5] Lucas Drumetz. *Lecture notes on Principal Component Analysis*. IMT Atlantique.

[6] Yannis Haralambous. *Lecture notes on Support Vector Machines*. IMT Atlantique.

[7] Dominique Pastor. *Lecture notes on Kernel methods*. IMT Atlantique.

OUR WORLDWIDE PARTNERS UNIVERSITIES - DOUBLE DEGREE AGREEMENTS

3 CAMPUS, 1 SITE

IMT Atlantique Bretagne–Pays de la Loire – **http://www.imt-atlantique.fr/**

Campus de Brest
Technopôle Brest-Iroise
CS 83818
29238 Brest Cedex 3
France
T +33 (0)2 29 00 11 11
F +33 (0)2 29 00 10 00

Campus de Nantes
4, rue Alfred Kastler
CS 20722
44307 Nantes Cedex 3
France
T +33 (0)2 51 85 81 00
F +33 (0)2 99 12 70 08

Campus de Rennes
2, rue de la Châtaigneraie
CS 17607
35576 Cesson Sévigné Cedex
France
T +33 (0)2 99 12 70 00
F +33 (0)2 51 85 81 99

Site de Toulouse
10, avenue Édouard Belin
BP 44004
31028 Toulouse Cedex 04
France
T +33 (0)5 61 33 83 65

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

© IMT Atlantique, 2021
Imprimé à IMT Atlantique
Dépôt légal : Septembre 2017
ISSN : 2556-5060