Development Project in Machine Learning

BENDOU Yassir
BENGHABRIT Ilyas
OULED ATTOU Hafsa

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

Contents

Introduction

This report presents the results of the mini-project in the class « Introduction to Machine Learning ».
In the following we show in detail the automation pipeline we have built which consists of data cleaning, feature selection, model selection, hyper-parameter tuning and model testing. We have automated our work in a way which allows for the user to pick any kind of model with any desired parameters and test its performance.

Datasets

We have two provided datasets. Both of them are meant for classification purposes. The first dataset is the Chronic kidney disease dataset and the second one is the banknote authentication dataset.

Chronic kidney disease dataset :

In this study, the chronic kidney disease dataset is data collected in a hospital in India over a period of about two months. The dataset is noisy and therefore needs cleaning.
The data are blood tests and other measures (Red blood cell count, hypertension, anemia…) of 400 patients (one row per patient). The first 25 columns are for the measures and the last one is to classify whether the patient has kidney disease or not.

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
155	155	50.0	70.0	1.020	3.0	0.0	abnormal	normal	present	present	...	36	4700	NaN	no	no	no	good	no	no	ckd
262	262	55.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	...	43	7200	5.4	no	no	no	good	no	no	notckd
236	236	65.0	80.0	NaN	NaN	NaN	NaN	NaN	notpresent	notpresent	...	25	NaN	NaN	yes	yes	yes	good	yes	no	ckd
228	228	60.0	70.0	NaN	NaN	NaN	NaN	NaN	notpresent	notpresent	...	NaN	NaN	NaN	yes	no	no	good	no	no	ckd
297	297	53.0	60.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	...	45	7700	5.2	NaN	NaN	NaN	good	no	no	notckd

5 rows × 26 columns

Figure 1: A sample of 5 rows of the dataset

	id	age	bp	sg	al	su	bgr	bu	sc	sod	pot	hemo
count	400.000000	391.000000	388.000000	353.000000	354.000000	351.000000	356.000000	381.000000	383.000000	313.000000	312.000000	348.000000
mean	199.500000	51.483376	76.469072	1.017408	1.016949	0.450142	148.036517	57.425722	3.072454	137.528754	4.627244	12.526437
std	115.614301	17.169714	13.683637	0.005717	1.352679	1.099191	79.281714	50.503006	5.741126	10.408752	3.193904	2.912587
min	0.000000	2.000000	50.000000	1.005000	0.000000	0.000000	22.000000	1.500000	0.400000	4.500000	2.500000	3.100000
25%	99.750000	42.000000	70.000000	1.010000	0.000000	0.000000	99.000000	27.000000	0.900000	135.000000	3.800000	10.300000
50%	199.500000	55.000000	80.000000	1.020000	0.000000	0.000000	121.000000	42.000000	1.300000	138.000000	4.400000	12.650000
75%	299.250000	64.500000	80.000000	1.020000	2.000000	0.000000	163.000000	66.000000	2.800000	142.000000	4.900000	15.000000
max	399.000000	90.000000	180.000000	1.025000	5.000000	5.000000	490.000000	391.000000	76.000000	163.000000	47.000000	17.800000

Figure 2: Statistics to describe the data overall

The disease attained 62,5% (250 patients)of the population while the 37,5% (150 patients) left remains unattained as we show in the table below

```
100*data['classification'].value_counts()/len(data)
```

```
ckd       62.5
notckd    37.5
Name: classification, dtype: float64
```

```
data['classification'].value_counts()
```

```
ckd       250
notckd    150
Name: classification, dtype: int64
```

Figure 3: Rate of the positive population

Banknote dataset

We have a dataset of 4 features and 1372 samples. The features are extracted from images taken of genuine and forged banknotes for inspection. The last column Class is for classification.

	variance	skewness	curtosis	entropy	class
354	3.5458	9.37180	-4.03510	-3.95640	0
221	2.4196	6.46650	-0.75688	0.22800	0
214	4.1962	0.74493	0.83256	0.75300	0
383	3.8117	10.14570	-4.04630	-4.56290	0
934	-1.6176	1.09260	-0.35502	-0.59958	1

Figure 4: A sample of 5 rows of the dataset

	variance	skewness	curtosis	entropy	class
count	1372.000000	1372.000000	1372.000000	1372.000000	1372.000000
mean	0.433735	1.922353	1.397627	-1.191657	0.444606
std	2.842763	5.869047	4.310030	2.101013	0.497103
min	-7.042100	-13.773100	-5.286100	-8.548200	0.000000
25%	-1.773000	-1.708200	-1.574975	-2.413450	0.000000
50%	0.496180	2.319650	0.616630	-0.586650	0.000000
75%	2.821475	6.814625	3.179250	0.394810	1.000000
max	6.824800	12.951600	17.927400	2.449500	1.000000

Figure 5: Statistics to describe the data overall

762(almost 55,54%) banknotes are classified as authentic and the remaining 610 as non authentic.

```
[14]: data['class'].value_counts()

t[14]: 0    762
       1    610
       Name: class, dtype: int64
```

```
Entrée [16]: 100*data['class'].value_counts()/len(data)

Out[16]: 0    55.539359
         1    44.460641
         Name: class, dtype: float64
```

Figure 6: Rate of the genuine banknotes

Preprocessing and data cleaning

For each dataset, the first steps to do were their cleaning and preprocessing. It consists of making the data exploitable to train the model on.

Data Cleaning:

By cleaning we mean replacing the miss-filled values with the correct ones, defining the right types of the dataset's variables, dropping some useless columns for the posterior treatment such as 'id' columns and encoding the output variable.

We found in the datasets some values that were miss-filled such as the ones containing '\t' or a space ' ' due to some bad filling by the data collectors. Such cases of values disturb the treatment of the data afterward. We then dealt with these cases by replacing those parts of the values with an empty string ''.
We also had some boxes with '?' that we considered as empty boxes and replaced them with NaN values so that we can treat them after just like the other empty values.

htn	dm	cad	appet	pe	ane
yes	\tyes	no	good	no	no

⇒

htn	dm	cad	appet	pe	ane
yes	yes	no	good	no	no

Figure 7: Cleaning of miss filled values

An important task at this stage was to define the type of the variables in the data so that we can perform on the right transformations in the next step of preprocessing. We then defined the type of numerical columns as floats in order to differentiate them from the categorical ones.

In the datasets, we also remarked the presence of some columns that should be dropped such as the 'id' columns. In fact, this column doesn't bring new information to the data and has a value that changes from one sample of the data to the other which brings a very high variance that we shouldn't take into consideration in the modelisation.

One last task in the cleaning of the data was to encode the output variable so that the training can once again be done.

Preprocessing:

In the preprocessing of the data, the goal was to transform the data in a shape that a model can train on. To do so we need to replace missing values, encode the features, center and normalize the data.

We began by replacing the missing values. To do so we need to define two different strategies for replacing, one for the numerical variables and the other one for the categorical ones. The most used ones are the replacing with the mean of the other values for numerical variables and the most frequent value for the categorical features.

rbc	pc	pcc	ba
NaN	normal	notpresent	notpresent
NaN	normal	notpresent	notpresent

⇒

rbc	pc	pcc	ba
normal	normal	notpresent	notpresent
normal	normal	notpresent	notpresent

Figure 8: Replacing of missing values (NaN)

The encoding of the categorical features is also necessary to let the training be possible. A model should surely train on data that contain only numerical values. Since the categorical features that we have in both datasets contain at most two modalities, we chose to encode with the preprocessing function in scikit-learn OrdinalEncoder.

rbc	pc	pcc
normal	normal	notpresent
normal	normal	notpresent
normal	normal	notpresent
normal	abnormal	present
normal	normal	notpresent

⇒

rbc	pc	pcc
1.0	1.0	0.0
1.0	1.0	0.0
1.0	1.0	0.0
1.0	0.0	1.0
1.0	1.0	0.0

Figure 9: Encoding of categorical features

Finally, we need to center and normalize the numerical features so that the training can become faster and more efficient. To do so we used the preprocessing function StandardScaler.

Feature Selection

In order to compress our features and remove linearity between them, we use the PCA algorithm. We fix the threshold ratio of the variance explained to 95% as shown if the plot below for the kidney disease dataset:

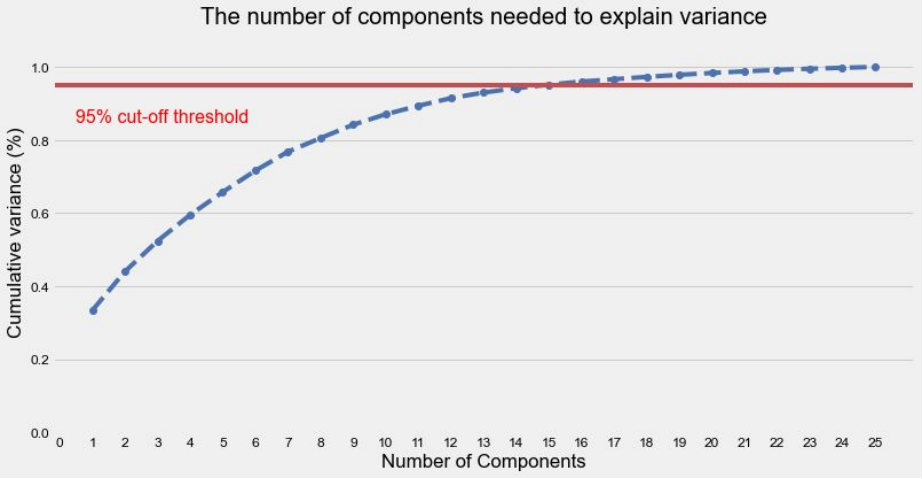

Figure 10: Cumulative variance per number of components

We only need 15 components to explain 95% of the variance of our dataset. We then use the coefficients of projections of our dataset on the principal components for the model training. The number of components is automatically selected for any dataset as long as the cut-off variance has been fixed.

Model Selection

Since the goal is to perform a binary classification on the datasets, the models chosen to do so were the classification models.

We selected five different classifiers : SVM, LogisticRegression, SGDClassifier, DecisionTreeClassifier, AdaBoostClassifier and RandomForestClassifier, and we let the user the possibility to choose the model with whom perform the training and classification. The model put by default is SVM.

A grid-search will be launched on this model to tune the parameters and a cross-validation will be done on its training.

Hyper-parameter tuning

Each model has some hyper-parameters that should be tuned so that it performs in the best way. To tune the hyper-parameters of a model, the combinaison allowing to have the best score should be chosen.
We define for every model that we mentioned earlier a set of many possible combinations.

```
'SVC':{'model':SVC,
       'parameters':{'kernel':['linear', 'rbf', 'sigmoid', 'poly'],
                                'C'     :[1, 10],
                                'degree': [2, 3],
                                'gamma' : ['scale', 'auto']
                     }
      },
'LogisticRegression':{'model':LogisticRegression,
                      'parameters':{'C':[1, 10],
                                'fit_intercept' : [True,False],
                                'intercept_scaling' : [1,10],
                                 }
                      },
```

Figure 11: grid search options

Then we run a grid-search to define the best combinaison among them to take.

Grid-Search:

We define all possible combinations to have among the possible set of parameters defined for the selected model. Then for each combination we do a cross-validation to compute a training score on the training data performed by the model with this combination of parameters.
We look then for the combination attending the maximum cross-validation score and we keep the corresponding hyperparameters.

Model Test

In this section, we will compare the performance of the different models that we used using the F1 score on the test datasets as a metric. The following table summarizes the results after running a gridsearch on the hyper-parameters :

Model	F1 Score (kidney dataset)	F1 Score (banknote dataset)
Support Vector Machine	99.11%	93.59%
Logistic Regression	100%	91.59%
Stochastic Gradient Descent Classifier	99.1%	84.77%
Decision Tree	100%	97.04%
Random Forest	99.11%	96.55%
AdaBoost Classifier	99.11%	94.07%

Table 1: Comparison of different models on the two datasets

We can see that the models struggle more on the banknote dataset and perform very well on the kidney dataset which has a lot of features. Surprisingly on the banknote dataset , the decision tree algorithm beats all the other models, while the random Forest and adaboost come second and third. One of the reasons that the Random Forest and the AdaBoost Classifier don't perform as well as the DecisionTree could be the lack of enough samples for the two models which induce a certain bias.

Process Automation

Our entire code has been written in two scripts, src/main.py which is our main script and src/utils.py which contains all the functions for the cleaning, the preprocessing, the training and the predicting. The functions work on both datasets and are modular, meaning each function has only one functionality.

The two datasets are stored in "data" folder and are fetched each time we run main.py

```
C:.
    .DS_Store
    .gitignore
    LICENSE
    README.md

├───data
    │   .DS_Store
    │
    ├───banknote
    │       data_banknote_authentication.txt
    │
    └───kidney_disease
            kidney_disease.csv

└───src
    │   main.py
    │   utils.py
    │
    └───__pycache__
            utils.cpython-38.pyc
```

Figure 12: Tree of the project source

Using a parser, our code has different options for the user to select from and can be triggered by running the command "python main.py" in "src" folder. In order to see the different possible scenarios, we can run "python main.py -h" which outputs the following :

```
(venv2) PS C:\Users\Yassir\Documents\TAFs\mce\mini-projet-ml-mce\src> python .\main.py -h
usage: main.py [-h] [--model MODEL] [--dataset DATASET] [--parameters KEY=VAL [KEY=VAL ...]] [--pca PCA]
               [-v CUT_OFF_VARIANCE]

Main-Code

optional arguments:
  -h, --help            show this help message and exit
  --model MODEL, -m MODEL
                        model to run on the dataset, options are {'SVC', 'LogisticRegression', 'SGDClassifier',
                        'RandomForestClassifier', 'AdaBoostClassifier'}
  --dataset DATASET, -d DATASET
                        dataset type, options are {'kidney','banknote'}, default is 'kidney'
  --parameters KEY=VAL [KEY=VAL ...], -p KEY=VAL [KEY=VAL ...]
                        model paramaters, dictionary of the parameters and their values, example for SVC : -p
                        kernel=rbf C=10 degree=2 gamma=auto
  --pca PCA             Apply PCA, if True applies PCA algorithm
  -v CUT_OFF_VARIANCE, --cut_off_variance CUT_OFF_VARIANCE
                        PCA cut-off variance or the threshold ratio of explained variance. Default is 0.95
(venv2) PS C:\Users\Yassir\Documents\TAFs\mce\mini-projet-ml-mce\src>
```

Figure 13: running the code with different parameters

An example would be to run the following command :
“python main.py -m=LogisticRegression -d=banknote --pca=True -v=0.98 -p kernel=rbf C=10 gamma=auto”
This command will run the code with a LogisticRegression with the following parameters (rbf kernel, C=10 and gamma=’auto’) on the banknote dataset. The data will also be transformed using PCA.

Another example is the following command : **“python main.py -m=AdaBoostClassifier --pca=False”**
This command will run the code with an AdaBoost Classifier on the kidney dataset (default dataset). The code will not run PCA algorithm and the code will also run a gridsearch to find the best parameters since the parameters were not specified by the user.

The output of the code is the score (F1 score) on the validation set and on the test set. If the user didn’t specify the model parameters, we also return the best parameters found by the grid search.

```
(venv2) PS C:\Users\Yassir\Documents\TAFs\mce\mini-projet-ml-mce\src> python .\main.py
params None
kidney dataset : Extracting data...
Cleaning data
number of features selected : 15
Chosen model and its different possible parameters: {'model': <class 'sklearn.svm._classes.SVC'>, 'parameters'
: {'kernel': ['linear', 'rbf', 'sigmoid', 'poly'], 'C': [1, 10], 'degree': [2, 3], 'gamma': ['scale', 'auto']}
}
running grid search to find the best parameters for the kidney dataset
Best parameters for SVC are {'kernel': 'rbf', 'C': 1, 'degree': 2, 'gamma': 'scale'}.
Best Score is 0.9849636554686217
Training with parameters : {'kernel': 'rbf', 'C': 1, 'degree': 2, 'gamma': 'scale'}
Cross Validation Score : 98.5%
F1 score on test data : 99.11%
```

Figure 14: Output of the code

Figure 8 shows a screenshot of the code’s output. In the output we can see that the Support Vector Machine model was run on the kidney disease dataset. The best parameters were

{'kernel': 'rbf', 'C': 1, 'degree': 2, 'gamma': 'scale'} with **98.5%** F1 score on the cross validation and **99.11%** score on the test accuracy.

Good programming practices

In this project, we tried our best to follow good programming practices such as factorising the code. For example, there is only one main part that calls all the needed functions only once, no matter the values of the parameters and whether they are given before executing the py file or not. We also followed modular programming where the main module is well structured and each coded function serves only one purpose. Moreover, we chose convenient non confusing names for our variables to facilitate their manipulation and smooth the flow of the code at each phase. Besides, we commented each line of the code that may seem ambiguous to a non contributor to the project to explain what is done clearly. We organised the comments to be interpreted by Sphinx to generate proper documentation which we included in the README.md file at the source of the project.

Conclusion

To conclude, we have built an end to end pipeline which allows a user to train a model and get its score on one of the two given datasets. We've used different coding techniques and collaboration techniques. Git has been a very important tool in our project and we have learnt how to manage a Machine Learning project remotely during this exercise.

Appendix

Github repository :

We collaborated on the project using github repository where we made commits that we agreed on at every progress we achieved. Sometimes, we would not push immediately until we discussed the matter in person.
The code is available on github at : https://github.com/ybendou/mini-projet-ml-mce
The following screenshot is the git log of our repository

```
(venv2) PS C:\Users\Yassir\Documents\TAFs\mce\mini-projet-ml-mce> git log --pretty=format:'%h %ad, message: %s'
9c27f61 Mon Dec 7 13:56:53 2020 +0100, message: Removing sphinx doc from README.md and adding it to the report
1c51ab5 Sun Dec 6 21:47:37 2020 +0100, message: Adding sphinx documentation support and generating documentation
0298b40 Sun Dec 6 20:52:12 2020 +0100, message: Update README.md
f50e814 Sun Dec 6 20:38:27 2020 +0100, message: Small fix on the SGDClassifier parameters
53b8acf Sun Dec 6 19:50:50 2020 +0100, message: adding small feature to the main
f159821 Sun Dec 6 19:27:28 2020 +0100, message: Fixing PCA feature selection and Adding sphinx documentation format
b5f4f07 Sun Nov 29 14:18:36 2020 +0100, message: Merge branch 'main' of https://github.com/ybendou/mini-projet-ml-mce in
to main
c1ab493 Sun Nov 29 14:17:36 2020 +0100, message: Creating main file and src folder and factorizing the code in the main
bad9282 Sun Nov 29 13:03:08 2020 +0100, message: Commenting the functions
8f3ce74 Tue Nov 24 23:19:14 2020 +0100, message: Adding training/croos_validation and grid search
ae74885 Tue Nov 24 23:08:46 2020 +0100, message: removing files to fix conflict
00df090 Tue Nov 24 21:08:33 2020 +0100, message: Adding Feature Selection/PCA
179c410 Mon Nov 23 19:01:07 2020 +0100, message: reorganizing the files
7373a5e Mon Nov 23 18:56:06 2020 +0100, message: preprocessing functions
3d196c9 Mon Nov 23 18:54:07 2020 +0100, message: Small modification on cleaning function
7ed192b Mon Nov 23 18:29:13 2020 +0100, message: Adding cleaning function
e088ec3 Mon Nov 23 17:13:09 2020 +0100, message: Adding jupyter notebook explore1
7a11abf Mon Nov 23 17:12:04 2020 +0100, message: adding a notebook to explore data
6eb1ca0 Mon Nov 23 17:07:41 2020 +0100, message: adding colomn names to banknote dataset
b4e7966 Mon Nov 23 17:04:25 2020 +0100, message: First push, creating the project template and adding the data
196308a Tue Nov 10 14:15:24 2020 +0100, message: Initial commit
```

Figure 15:logs of the git repository

Code :

We have two files : main.py and utils.py which we joined to this document. We also join a documentation of the different functions which we generate using sphinx, a powerful tool to generate documentation from python scripts.

```
##########################################
#                                        #
#              main.py                   #
#                                        #
##########################################

import pandas as pd
import numpy as np
from utils import *
import argparse
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
import json

#This is a particular class to include dictionaries in the parsers
class StoreDictKeyPair(argparse.Action):
    def __init__(self, option_strings, dest, nargs=None, **kwargs):
        self._nargs = nargs
        super(StoreDictKeyPair, self).__init__(option_strings, dest, nargs=nargs, **kwargs)
    def __call__(self, parser, namespace, values, option_string=None):
        my_dict = {}
        for kv in values:
            k,v = kv.split("=")
            my_dict[k] = v
        setattr(namespace, self.dest, my_dict)


# All the args arguments:
parser = argparse.ArgumentParser(description='Main-Code')
parser.add_argument('--model', '-m', type=str, default='SVC',help = "model to run on the dataset, options are \n{'SVC',
'LogisticRegression', 'SGDClassifier', 'RandomForestClassifier', 'AdaBoostClassifier','DecisionTreeClassifier}") # model choice
parser.add_argument('--dataset','-d', type=str, default='kidney', help="dataset type, options are {'kidney','banknote'}, default is
'kidney'") # dataset choice
parser.add_argument("--parameters","-p" , dest="my_dict", action=StoreDictKeyPair, nargs="+",
metavar="KEY=VAL",help="model paramaters, dictionary of the parameters and their values, example for SVC : -p kernel=rbf
C=10 degree=2 gamma=auto") # parameters choice, it's a dictionary
parser.add_argument('--pca', type=bool, default=True, help="Apply PCA, if True applies PCA algorithm") #Choice to keep the
original features or to use the PCA coefficients instead
parser.add_argument('-v', '--cut_off_variance',type=float, default=0.95, help="PCA cut-off variance or the threshold ratio of
explained variance. Default is 0.95") #Ratio threshold of variance explained


data_paths = {'kidney':'../data/kidney_disease/kidney_disease.csv',
          'banknote':'../data/banknote/data_banknote_authentication.txt'} # Dictionary of datasets paths

# Dictionary of models and their corresponding parameters for the gridsearch
models_dict = {'SVC':{'model':SVC, #Support vector Classifier
                 'parameters':{'kernel':['linear', 'rbf', 'sigmoid', 'poly'],
                               'C'     :[1, 10],
                               'degree': [2, 3],
                               'gamma' : ['scale', 'auto']
                              }
                },
           'LogisticRegression':{'model':LogisticRegression, # Logistic Regression
                         'parameters':{'C':[1, 10],
                               'fit_intercept' : [True,False],
                               'intercept_scaling' : [1,10],
                                }
                         },
            'SGDClassifier':{'model':SGDClassifier, # Stochastic gradient descent classifier
                         'parameters':{'loss':['hinge', 'log', 'modified_huber', 'squared_hinge', 'perceptron'],
                               'penalty':['l1', 'l2'],
```

```
                                'fit_intercept' : [True,False],
                                }
                        },
            'DecisionTreeClassifier':{'model':RandomForestClassifier,
                        'parameters':{'criterion':['gini', 'entropy'],
                                'max_depth' : [2,5,10,None],
                                }
                        },
            'AdaBoostClassifier':{'model':AdaBoostClassifier,
                        'parameters':{'n_estimators':[50, 100, 150],
                                'algorithm':['SAMME', 'SAMME.R'],
                                'learning_rate' : [0.1,0.5,1]
                                }
                        },
            'RandomForestClassifier':{'model':RandomForestClassifier,
                        'parameters':{'n_estimators':[50, 100, 150],
                                'criterion':['gini', 'entropy'],
                                'max_depth' : [2,5,10,None],
                                'bootstrap' : [True,False],
                                }
                        }
            }
def main():
    """
    @author : Yassir BENDOU
    main function of the program
    """
    args = parser.parse_args()  #Initiate the parser
    data_str  = args.dataset    # dataset choice
    model_str = args.model      # model choice
    params    = args.my_dict    # get the parameters dictionary provided by the user
    print('params',params)
    print(f'{data_str} dataset : Extracting data...')
    data = pd.read_csv(data_paths[data_str]) # Fetch the data
    print('Cleaning data')
    data = clean_noisy_data(data,classes = 2)  # Clean the dataset
    num_variables, categ_variables = detect_type(data.iloc[:,:-1]) # Detect numerical and categorical data
    data.iloc[:,:-1] = replace_missing(data.iloc[:,:-1], num_variables, categ_variables) # Replace missing values
    data.iloc[:,:-1] = center_encode(data, num_variables, categ_variables)  # scale values and ordinal ordinalencode them

    if args.pca:
        dataset = feature_selection(data,cut_off_variance=args.cut_off_variance)  # Compress the data using PCA

    X,y = dataset.values[:,:-1],dataset.values[:,-1]
    X_train, X_test, y_train, y_test = split_data(X,y)

    model_choice = models_dict[model_str]
    model = model_choice['model']
    if params == None : # If the user doesn't provide any parameters for the model, we run the gridsearch and pick the best ones
        print(f'Chosen model and its different possible parameters: {model_choice}')
        print(f'running grid search to find the best parameters for the {data_str} dataset')
        parameters = model_choice['parameters']
        params,best_score = select_params(model, parameters, X_train, y_train)
        print(f'Best parameters for {model_str} are {params}. \nBest Score is {best_score}')

    print(f'Training with parameters : {params}')
    score_val,clf = training(model, params, X_train, y_train) # Train the model
    print(f'Cross Validation Score : {np.round(100*score_val,2)}%')
    score_test = test_evaluate(clf,X_test,y_test) # Run the model on test data
    print(f'F1 score on test data : {np.round(100*score_test,2)}%')

if __name__ == '__main__':
    main()
```

```
##########################################
#                                        #
#              utils.py                  #
#                                        #
##########################################
import pandas as pd
import numpy as np
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, StandardScaler, OneHotEncoder
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
import itertools
from sklearn.metrics import f1_score

def clean_noisy_data(dataset,classes = 2):
    """
    @author : Yassir BENDOU
    Clean the data with replacing the miss-filled values by the correct ones and defining the right types for the dataset
    variables.

    Args:
        :param dataset: The dataset to clean.
        :type  dataset: pandas.DataFrame

    Returns:
        :param dataset: Cleaned dataset.
        :type dataset: pandas.DataFrame
    """

    if 'id' in dataset.columns :
        dataset = dataset.drop(columns = ['id']) # Drop id column as it's not relevent for predictions


    for c in dataset.columns :
        if dataset[c].dtype not in ['float32','float64','float','int32','int64','int']:
            dataset[c] = dataset[c].str.replace('\t','')
            dataset[c] = dataset[c].replace('?',np.nan)
            dataset[c] = dataset[c].str.replace(' ','')


    #Changing numerical data into float types
    output_column = dataset.columns[-1]

    string_columns = []
    numerical_columns = []
    output_is_string = False
    for c in dataset.columns :
        try :
            dataset[c] = dataset[c].astype(float) #transform numerical data that is stored as string to float type
            numerical_columns.append(c)
        except ValueError :
            if c == output_column:
                output_is_string = True
            string_columns.append(c)

    #Ordinal encoding of the output variable
    if output_is_string:
        outputs = dataset[output_column].unique()
        assert len(outputs) == classes, 'Error the number of output classes should be the same as the one in the dataset'
        dataset[output_column] = 1*(dataset[output_column] == outputs[0]) #return an ordinal encoding of the output variable
    dataset[output_column] = dataset[output_column].astype(int)

    dataset[numerical_columns] = dataset[numerical_columns].astype(float)
    return dataset
```

```
def detect_type(data):
    """
    @author : Ilyas BENGHABRIT
    Detecting the type of the variables numerical or categorical

    Args:
        :param data: The dataset.
        :type  data: pandas.DataFrame

    Returns:
        :param num_variables: List of columns with numerical values.
        :type num_variables: list

        :param categ_variables: List of columns with categorical values.
        :type categ_variables: list
    """

    num_variables = []
    categ_variables = []
    columns = list(data.columns)
    n = len(columns)
    for i in range(n):
        if data[columns[i]].dtype in  ['int32','int64','float32','float64','int','float']: #Checking if the variable is numerical
            num_variables.append(columns[i])
        else :
            categ_variables.append(columns[i])
    return num_variables, categ_variables


def replace_missing(data, num_variables, categ_variables, num_strategy = 'mean', categ_strategy = 'most_frequent'):
    """
    @author : Ilyas BENGHABRIT
    Replacing the missing values in categorical variables and numerical variables by 2 corresponding strategies
    (mean for numerical variables and the most frequent value for categorical varibles for example)

    Args:
        :param data: The dataset.
        :type  data: pandas.DataFrame

        :param num_variables: List of columns with numerical values.
        :type num_variables: list

        :param categ_variables: List of columns with categorical values.
        :type categ_variables: list

        :param num_strategy: The defined strategy to replace the numerical missing values, default is mean.
        :type num_strategy: str

        :param categ_strategy: The defined strategy to replace the categorical missing values, default is most_frequent.
        :type categ_strategy: str

    Returns:
        :param data_tr_table: A transformed dataset with missing values filled.
        :type data_tr_table: pandas.DataFrame
    """

    ct = ColumnTransformer([("categ_imput", SimpleImputer(missing_values = np.nan, strategy = categ_strategy),
categ_variables),
                            ("num_imput", SimpleImputer(missing_values = np.nan, strategy = num_strategy), num_variables)])
    data_transformed = ct.fit_transform(data) #Recuperate the transformed array
    columns = categ_variables + num_variables
    data_tr_table = pd.DataFrame(data_transformed, columns = columns)#Putting the transformation into a dataframe
    return data_tr_table

def center_encode(data, num_variables, categ_variables):
    """
```

```
    @author : Ilyas BENGHABRIT
    Centring and normalizing the data. Transforming the categorical variables.

    Args:
        :param data: The dataset.
        :type  data: pandas.DataFrame

        :param num_variables: List of columns with numerical values.
        :type num_variables: list

        :param categ_variables: List of columns with categorical values.
        :type categ_variables: list

    Returns:
        :param data_tr_table: A centered and encoded dataset.
        :type data_tr_table: pandas.DataFrame
    """
    cat_enc = OrdinalEncoder()
    center_norm = StandardScaler()
    if categ_variables != [] and num_variables != []:
        categ_data = data[categ_variables]
        num_data = data[num_variables]
        data_transformed_cat = cat_enc.fit_transform(categ_data)
        data_transformed_num = center_norm.fit_transform(num_data) # center and normalize numerical data
        categ_columns = categ_variables
        columns = list(categ_columns) + num_variables
        data_transformed = np.concatenate((data_transformed_cat, data_transformed_num), axis = 1)
        data_tr_table = pd.DataFrame(data_transformed, columns = columns)
    elif categ_variables != []:
        categ_data = data[categ_variables]
        data_transformed_cat = cat_enc.fit_transform(categ_data)
        columns = categ_variables
        data_tr_table = pd.DataFrame(data_transformed_cat, columns = columns)
    elif num_variables != []:
        num_data = data[num_variables]
        data_transformed_num = center_norm.fit_transform(num_data)
        columns = num_variables
        data_tr_table = pd.DataFrame(data_transformed_num, columns = columns)
    return data_tr_table


def feature_selection(dataset,cut_off_variance=0.95):
    """
    @author : Yassir BENDOU
    Applies Feature selection using PCA. We fix the number of the remaining vectors based on the number of components which
garantees 95% of the original variance of the dataset.

    Args:
        :param dataset: The dataset.
        :type  dataset: pandas.DataFrame

        :param cut_off_variance: The threshold ratio of the explained variance, takes values between 0 and 1, default is 0.95.
        :type cut_off_variance: float

    Returns:
        :param data_compressed: The compressed dataset.
        :type data_compressed: pandas.DataFrame
    """
    X,_ = dataset.values[:,:-1],dataset.values[:,-1]
    pca = PCA(n_components=len(dataset.columns)-1)
    pca.fit(X)
    y = np.cumsum(pca.explained_variance_ratio_) # get the cumulative explained variance

    number_of_features = len(dataset.columns) - sum(y>=cut_off_variance)
    print(f'number of features selected : {number_of_features}')

    new_X = pca.fit_transform(X) # get the projections on the eigen vectors of PCA.
```

```
    data_compressed = pd.DataFrame(new_X[:,:number_of_features]) # only keep the desired features according to the
cut_off_variance
    data_compressed[dataset.columns[-1]] = dataset.iloc[:,-1]

    return data_compressed

def split_data(X, y):
    """
    @author : Hafsa OULED ATTOU
    Split the data into training set and validation set

    Args:
        :param X: The features
        :type  X: numpy.array

        :param y: labels
        :type  y: numpy.array

    Returns: Training set and validation set of the data, training set and validation set of the class
    """
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)
    return X_train, X_test, y_train, y_test

def determine_combinations(parameters):
    """
    @author : Yassir BENDOU
    Determine all possible combinaisons to have from a defined set of parameters of a model

    Args:
        :param parameters:  Dictionary of parameters.
        :type  parameters: dict

    Returns:
        :param comb_parameters: List of all combinaisons, each combinaison of parameters defined as a dictionary.
        :type comb_parameters: list
    """
    parameters_values = list(parameters.values())
    combinations = list(itertools.product(*parameters_values))
    comb_parameters = []
    for c in combinations :
        d = {}
        keys = list(parameters.keys())
        for k in range(len(keys)):
            d[keys[k]] = c[k]
        comb_parameters.append(d)
    return comb_parameters

def training(model, parameters, X, y):
    """
    @author : Hafsa OULED ATTOU
    Train the model with defined parameters and returns the cross validation score

    Input : Model, parameters of the model, data, class
    Output : the score of the cross validation

    Args:
        :param model:  the defined model.
        :type  model: object

        :param parameters: parameters of the model.
        :type  parameters: dict

        :param X: features.
        :type  X: numpy.array

        :param y: labels.
        :type  y: numpy.array
```

```
    Returns:
        :param score: the score of the cross validation.
        :type score: float

        :param clf: the trained model.
        :type clf: object
    """
    clf = model(**parameters)
    scores = cross_val_score(clf, X, y, cv=5, scoring='f1_macro')
    clf.fit(X,y)
    score = np.mean(scores)
    return score,clf

def select_params(model, parameters, X, y):
    """
    @author : Ilyas BENGHABRIT
    Selecting the best parameters to take for the model

    Input : Model, dictionary of possible parameters, Data, class
    Output :

    Args:
        :param model:  the defined model.
        :type  model: object

        :param parameters: all possible parameters of the model for the gridsearch, format is a dictionary of lists. \nExample :
'parameters':{'loss':['log', 'squared_hinge', 'perceptron']}}
        :type  parameters: dict

        :param X: features.
        :type  X: numpy.array

        :param y: labels.
        :type  y: numpy.array
    Returns:
        :param chosen_params: Chosen combinaison of parameters, score of the cross validation with this combinaison.
        :type chosen_params: dict

        :param score_max: score of the chosen parameters, which is the highest score.
        :type score_max: float
    """

    comb_parameters = determine_combinations(parameters) # Get all the possible combinations
    total_scores = []
    for i in range(len(comb_parameters)):
        total_scores.append(training(model, comb_parameters[i], X, y)[0])
    score_max = np.max(total_scores)
    ind_max = np.argmax(total_scores)
    chosen_params = comb_parameters[ind_max] # return the parameters with the highest score.

    return chosen_params, score_max


def test_evaluate(model,X,y):
    """
    @author : Hafsa OULED ATTOU
    Evaluate the model on test data and returns the f1 score.
    """
    y_pred = model.predict(X)
    score = f1_score(y, y_pred,average='macro')
    return score
```

src

utils

`src.utils.center_encode`(*data*, *num_variables*, *categ_variables*) [source]

Centring and normalizing the data. Transforming the categorical variables.

Args:

param data: The dataset.
type data: pandas.DataFrame
param num_variables: List of columns with numerical values.
type num_variables: list
param categ_variables: List of columns with categorical values.
type categ_variables: list

Returns:

param data_tr_table: A centered and encoded dataset.
type data_tr_table: pandas.DataFrame

`src.utils.clean_noisy_data`(*dataset*, *classes=2*) [source]

Clean the data with replacing the miss-filled values by the correct ones and defining the right types for the dataset variables.

Args:

param dataset: The dataset to clean.
type dataset: pandas.DataFrame

Returns:

param dataset: Cleaned dataset.
type dataset: pandas.DataFrame

`src.utils.detect_type`(*data*) [source]

Detecting the type of the variables numerical or categorical

Args:

param data: The dataset.
type data: pandas.DataFrame

Returns:

param num_variables: List of columns with numerical values.
type num_variables: list
param categ_variables: List of columns with categorical values.
type categ_variables: list

`src.utils.determine_combinations`(*parameters*) [source]

Determine all possible combinaisons to have from a defined set of parameters of a model

Args:

param parameters: Dictionary of parameters.
type parameters: dict

Returns:

param comb_parameters: List of all combinaisons, each combinaison of parameters defined as a dictionary.
type comb_parameters: list

`src.utils.feature_selection`(*dataset*, *cut_off_variance=0.95*) [source]

Applies Feature selection using PCA. We fix the number of the remaining vectors based on the number of components which garantees 95% of the original variance of the dataset.

Args:

param dataset: The dataset.
type dataset: pandas.DataFrame
param cut_off_variance: The threshold ratio of the explained variance, takes values between 0 and 1, default is 0.95.
type cut_off_variance: float

Returns:

param data_compressed: The compressed dataset.
type data_compressed: pandas.DataFrame

`src.utils.replace_missing`(*data*, *num_variables*, *categ_variables*, *num_strategy='mean'*, *categ_strategy='most_frequent'*) [source]

Replacing the missing values in categorical variables and numerical variables by 2 corresponding strategies (mean for numerical variables and the most frequent value for categorical varibles for example)

Args:

param data: The dataset.
type data: pandas.DataFrame
param num_variables: List of columns with numerical values.
type num_variables: list
param categ_variables: List of columns with categorical values.
type categ_variables: list
param num_strategy: The defined strategy to replace the numerical missing values, default is mean.
type num_strategy: str
param categ_strategy: The defined strategy to replace the categorical missing values, default is most_frequent.
type categ_strategy: str

Returns:

param data_tr_table: A transformed dataset with missing values filled.
type data_tr_table: pandas.DataFrame

`src.utils.select_params`(*model*, *parameters*, *X*, *y*) [source]

Selecting the best parameters to take for the model

Input : Model, dictionary of possible parameters, Data, class Output :

Args:

param model: the defined model.
type model: object
param parameters: all possible parameters of the model for the gridsearch, format is a dictionary of lists.

Example : *'parameters':{'loss':['log', 'squared_hinge', 'perceptron']}*

type parameters: dict
param X: features.
type X: numpy.array
param y: labels.
type y: numpy.array

Returns:

param Chosen combinaison of parameters, score of the cross validation
chosen_params: with this combinaison.
type chosen_params: dict
param score_max: score of the chosen parameters, which is the highest score.
type score_max: float

`src.utils.`**`split_data`**(*X*, *y*) [source]

Split the data into training set and validation set

Args:

param X: The features
type X: numpy.array
param y: labels
type y: numpy.array

Returns: Training set and validation set of the data, training set and validation set of the class

`src.utils.`**`test_evaluate`**(*model*, *X*, *y*) [source]

Evaluate the model on test data and returns the f1 score.

`src.utils.`**`training`**(*model*, *parameters*, *X*, *y*) [source]

Train the model with defined parameters and returns the cross validation score

Input : Model, parameters of the model, data, class Output : the score of the cross validation

Args:

param model: the defined model.
type model: object
param parameters: parameters of the model.
type parameters: dict
param X: features.
type X: numpy.array
param y: labels.
type y: numpy.array

Returns:

param score: the score of the cross validation.
type score: float
param clf: the trained model.
type clf: object