Introduction to the Theory and Practice of Machine Learning

Machine Learning Rapport

Hazim Benslimane
Christopher Jabea
Dylan Rachwal
Alexandre Thouvenot

December 15th, 2021

Contents

Introduction

We decided to work on the binary classification task. Our task consists ed to analyse two data-sets, one based on data extracted from banknotes and one based on physiological data possibly related to chronic kidney disease. In this report, we will analyse multiple methods of classification such as Support Vector Machine (SVM), Principal Component Analysis (PCA), Classification Tree and Forest and K-Nearest Neighbors (KNN) algorithms

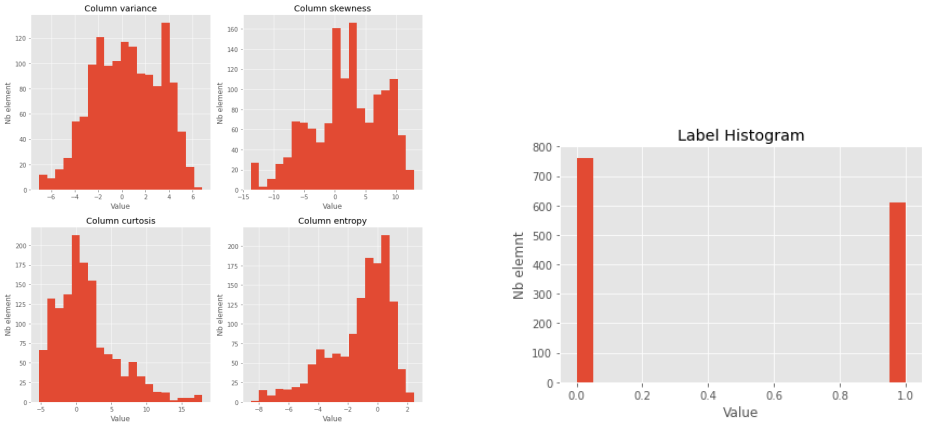
1 Banknote Authentication Dataset

1.1 Description of the Dataset

This is data-set is relatively simple and is composed of 4 features associated to one label (0 or 1). All the features are float values and there are no missing values in the data-set. We decided to normalize the data-set with min-max method before applying classification algorithm.

$$data = \frac{data - \min(data)}{\max(data) - \min(data)}$$

The figure below shows the histogram of each feature before the normalization.



(a) Data Histrogram (b) Label Histogram

Figure 1: Banknote data-set

1.2 The Different method used

We decided to try a dimensionality-reduction method like PCA to see if there exists interesting lower dimension space on which apply classification algorithm. We compared to type of PCA reduction, one based on EVD (eigen value decomposition) and one based on SVD (singular value decomposition).
We decided to try three different classification algorithms on this data-set. The

fisrt one is KNN (k-nearest neighbors), the second one is SVC (support vector classification) and the last are decision tree and decision forest.

1.3 Results

1.3.1 PCA

As we can see on the graph below, EVD method has a slightly higher explained variance than the SVD decomposition. The explained variance is high enough for 2 or 3 components to try to apply classification algorithm.

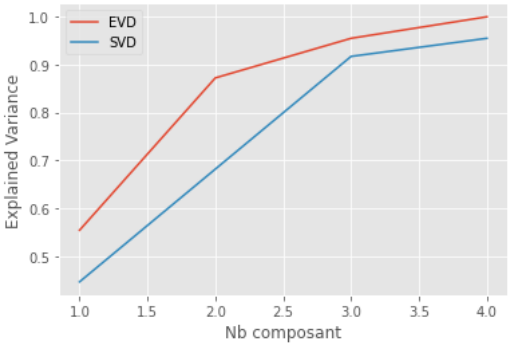
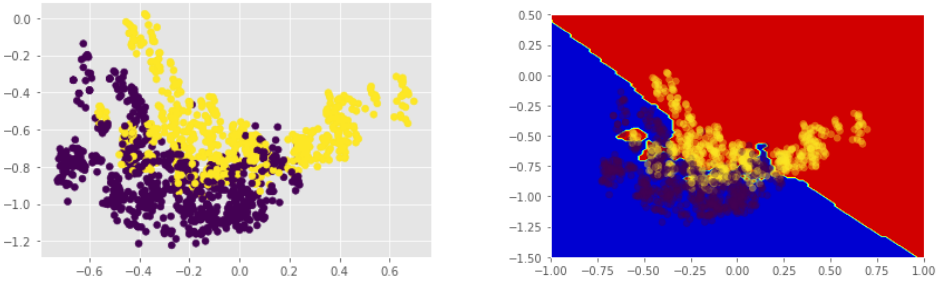


Figure 2: Comparison of PCA reduction method

1.3.2 KNN

On the 4 features of the data-set, a value of K = 10 and KFold cross validation procedure, the KNN algorithm gives good result (more than 99% of precision and recall). In addition, it is relevant to see the result with the PCA projection on two components.



(a) Data Histrogram (b) Label histogram

Figure 3: Banknote data-set

As we can see on the figure above, the decision boundary of the two clusters is not clear, and thus KNN performs not well on two components of PCA.

1.3.3 SVC

As we can see on the Figure 4, the prediction is highly depend of the choice of the kernel, the parameter which decide the shape of the hyperplane. Here, the only one that cannot be used is a sigmoid kernel as our data are not normalized between -1 and 1.

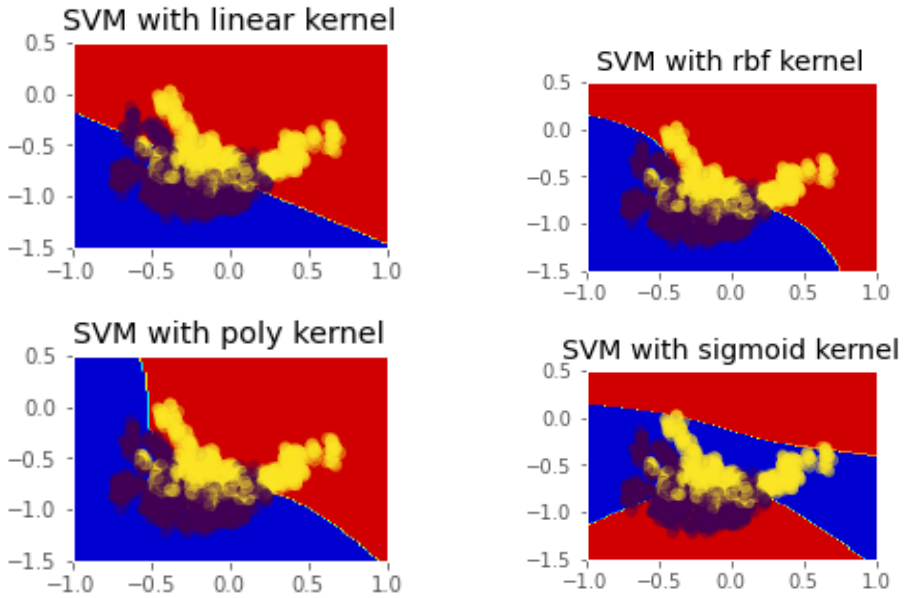


Figure 4: Comparison of SVM kernel choices

However when looking closely to the key metrics in Figure 5, the best kernel to choose is the rbf kernel.

```
SVM with linear kernel
For the metric fit_time, the mean is 0.025 and the standard deviation is 0.001
For the metric score_time, the mean is 0.004 and the standard deviation is 0.0
For the metric test_precision_macro, the mean is 0.781 and the standard deviation is 0.079
For the metric test_recall_macro, the mean is 0.772 and the standard deviation is 0.077
For the metric test_roc_auc, the mean is 0.888 and the standard deviation is 0.057
For the metric test_f1, the mean is 0.74 and the standard deviation is 0.093
------------------------------

SVM with poly kernel
For the metric fit_time, the mean is 0.03 and the standard deviation is 0.003
For the metric score_time, the mean is 0.004 and the standard deviation is 0.001
For the metric test_precision_macro, the mean is 0.819 and the standard deviation is 0.077
For the metric test_recall_macro, the mean is 0.818 and the standard deviation is 0.078
For the metric test_roc_auc, the mean is 0.926 and the standard deviation is 0.043
For the metric test_f1, the mean is 0.8 and the standard deviation is 0.088
------------------------------

SVM with rbf kernel
For the metric fit_time, the mean is 0.038 and the standard deviation is 0.005
For the metric score_time, the mean is 0.006 and the standard deviation is 0.001
For the metric test_precision_macro, the mean is 0.844 and the standard deviation is 0.07
For the metric test_recall_macro, the mean is 0.839 and the standard deviation is 0.072
For the metric test_roc_auc, the mean is 0.927 and the standard deviation is 0.042
For the metric test_f1, the mean is 0.818 and the standard deviation is 0.086
------------------------------
```

Figure 5: Metrics for SVM methods

1.3.4 Decision Tree

Here, we first use a cross validation procedure to tune the depth of the Classification Tree using the negative log loss metric. In the Figure 6, we can see that the best depth is clearly a depth of 3 which has the smaller values for 99 % of samples. Besides we can see that a higher depth will lead to overfitting with a high variance and a higher bias and for lower depths, a small variance but higher bias.

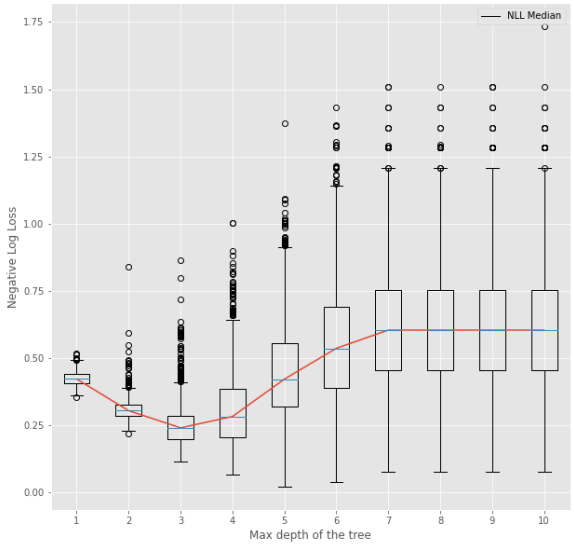


Figure 6: Comparison of the different number of depths using negative log loss

Using 3 as depth, we obtain the tree in Figure 7

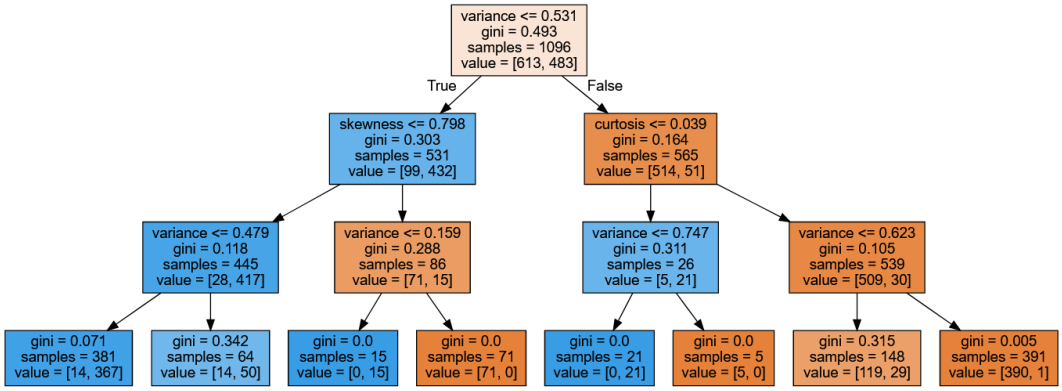
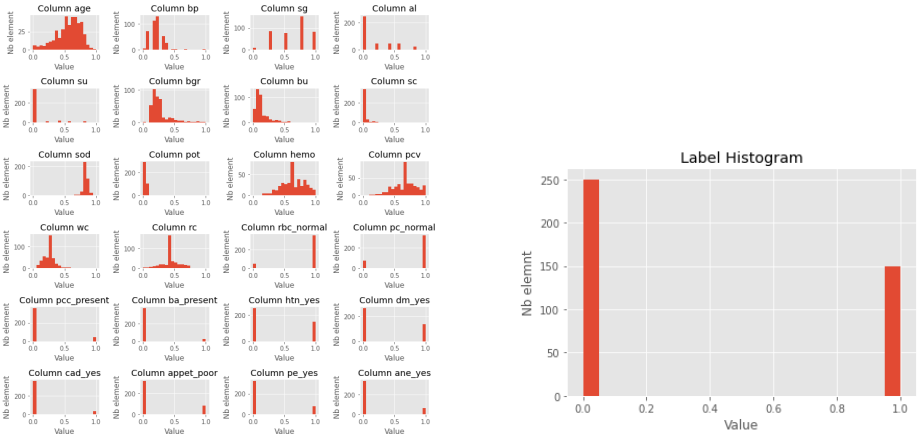


Figure 7: Tree with a depth of 3

2 Chronic Kidney Disease

2.1 Description of the Dataset

The second data-set is more difficult to analyse. It is composed of 24 features, which are physiological parameters of different patient. Each patient is associated to a label, 0 or 1, if he suffers from kidney disease. Some of the features are scalar, but some of them are string with to possible label. We decided to replace these string labels with integer value (0 or 1). Then we handled missing data in the data-set. Some of the scalar value can be easily replace by the mean of the other value. Nonetheless, some of the scalar values are in a discrete value space (for instance the age). In this case, we replace the missing value by a random value in the same feature. Because some of the values are one-hot encode (0 or 1), we decided to apply a min-max normalization to conserve this property of the data-set. The graphs below show the histogram of the features and the labels before the normalisation.



(a) Data Histrogram (b) Label histogram

Figure 8: Kidney disease data-set

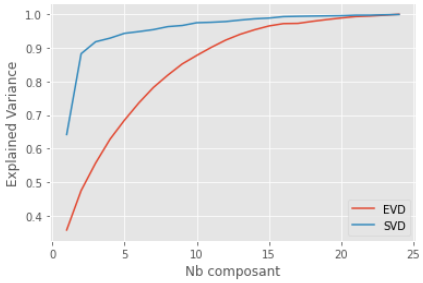
2.2 The Different method used

We chose to apply the same procedure as the banknote data-set. The method are the same, PCA, KNN, SVC, and decision tree.

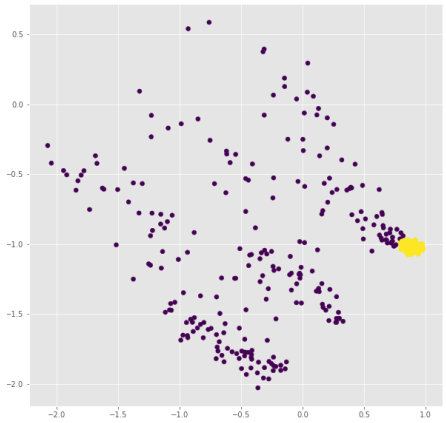
2.3 Results

2.3.1 PCA

PCA reduction with EVD gives interesting results. As we can see on the figure below, the explained variance of EVD is extremely high with only few components.



(a) Comparison of PCA



(b) Projection on 2 components with EVD PCA

Figure 9: PCA Algorithm

As we can see on the figure above, it is possible to distinguish to clusters corresponding to the two different labels.

2.3.2 KNN

KNN gives also quite good result on this data-set. On the full data-set (no reduction), the algorithm reach more than 93% of recall on precision.

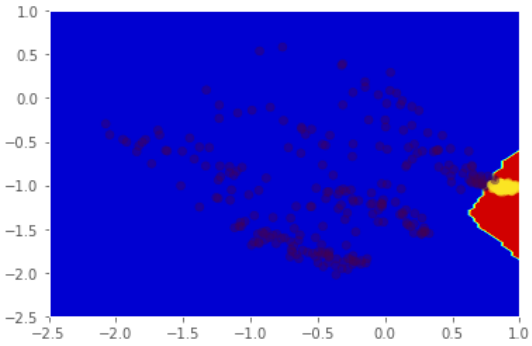


Figure 10: Decision boundary of KNN on PCA data

On the figure above, we can see that the decision boundary obtained with KNN algorithm on data reduced to 2 dimension with PCA separates quite well the two clusters.

2.3.3 SVC

After using the PCA to only have 2 dimensions of features, we can see on Figure 11 that every algorithms fit very well with the dataset. However when we look closer to Figure 12, the polynomial kernel of degree 3 is better than the others according to accuracy and recall (raw values and derivated values as f1-score or ROC curve) with 97% of precision and recall.

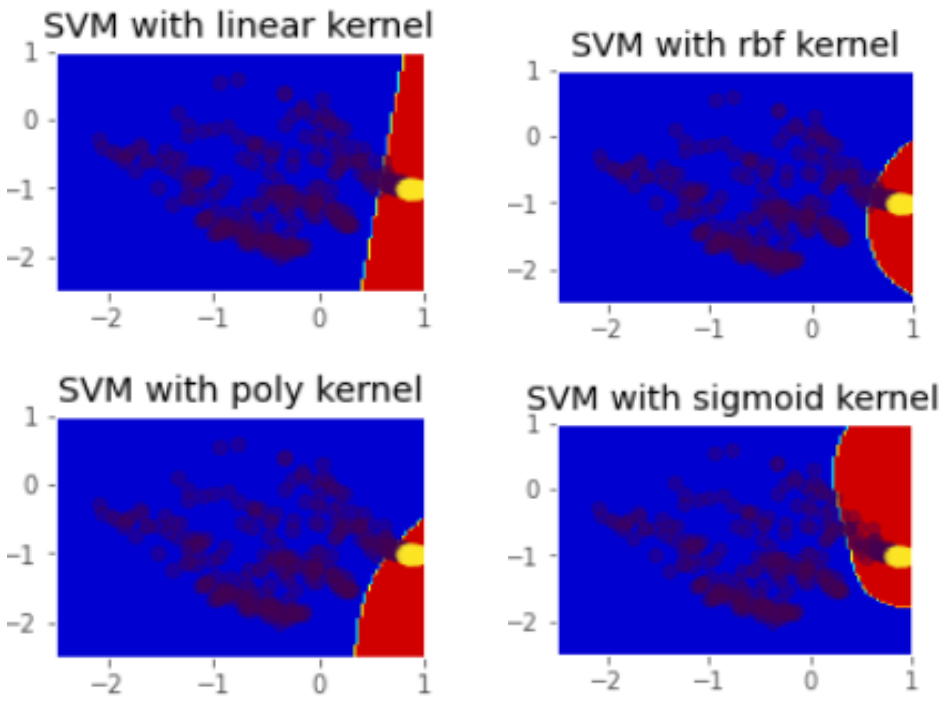


Figure 11: Comparison of SVM kernel choices

```
SVM with linear kernel
For the metric fit_time, the mean is 0.001 and the standard deviation is 0.002
For the metric score_time, the mean is 0.003 and the standard deviation is 0.002
For the metric test_precision_macro, the mean is 0.958 and the standard deviation is 0.061
For the metric test_recall_macro, the mean is 0.966 and the standard deviation is 0.051
For the metric test_roc_auc, the mean is 1.0 and the standard deviation is 0.0
For the metric test_f1, the mean is 0.952 and the standard deviation is 0.071
------------------------------

SVM with poly kernel
For the metric fit_time, the mean is 0.001 and the standard deviation is 0.001
For the metric score_time, the mean is 0.002 and the standard deviation is 0.002
For the metric test_precision_macro, the mean is 0.985 and the standard deviation is 0.041
For the metric test_recall_macro, the mean is 0.988 and the standard deviation is 0.032
For the metric test_roc_auc, the mean is 1.0 and the standard deviation is 0.0
For the metric test_f1, the mean is 0.983 and the standard deviation is 0.046
------------------------------

SVM with rbf kernel
For the metric fit_time, the mean is 0.001 and the standard deviation is 0.002
For the metric score_time, the mean is 0.002 and the standard deviation is 0.002
For the metric test_precision_macro, the mean is 0.961 and the standard deviation is 0.06
For the metric test_recall_macro, the mean is 0.968 and the standard deviation is 0.051
For the metric test_roc_auc, the mean is 1.0 and the standard deviation is 0.0
For the metric test_f1, the mean is 0.955 and the standard deviation is 0.07
------------------------------

SVM with sigmoid kernel
For the metric fit_time, the mean is 0.003 and the standard deviation is 0.002
For the metric score_time, the mean is 0.002 and the standard deviation is 0.002
For the metric test_precision_macro, the mean is 0.916 and the standard deviation is 0.088
For the metric test_recall_macro, the mean is 0.92 and the standard deviation is 0.094
For the metric test_roc_auc, the mean is 0.96 and the standard deviation is 0.069
For the metric test_f1, the mean is 0.898 and the standard deviation is 0.112
```

Figure 12: Metrics for SVM methods

2.3.4 Decision Tree

Here, we first use a cross validation procedure to tune the depth of the Classification Tree using the negative log loss metric. In the Figure 13, we can see that the best depth is clearly a depth of 2 which has the smaller values for 99 % of samples. We obtain with a such depth and using a Classification Forest 98% of precision and recall.

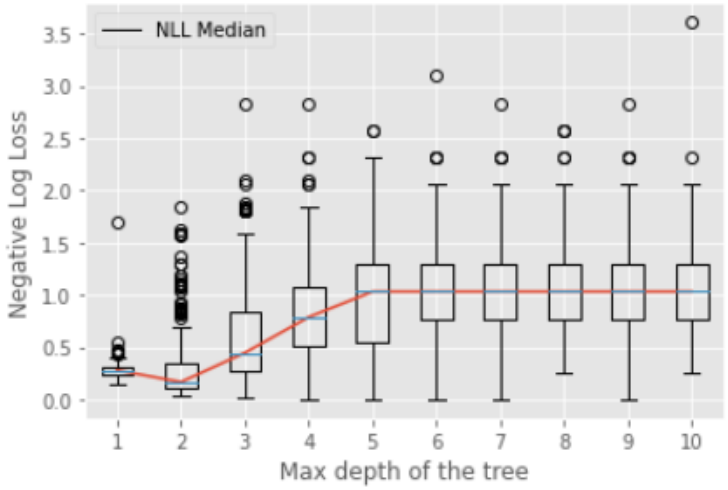


Figure 13: Comparison of the different number of depths using negative log loss

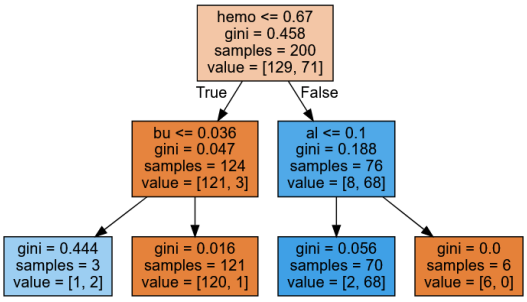


Figure 14: Tree with a depth of 3

3 Good Programming Practices

3.1 Git

Git is a tool to control the version of an application or program. It's a great collaborative tool to access the different versions and has a history of all the changes made from the first version of your application.

3.1.1 Git WorkFlow

When working with Git, you can think of it as a tree and the different branches correspond to the different versions of your app. The main version is located in a branch called master.

3.1.2 Branch and Merging

If a developer wants to modify the code, the wrong thing to do is to modify the code that is located in branch master directly because if you modify the code of master, it can cause some issues on the application, especially if it's hosted somewhere and millions of people have access to it. The correct way is to create a new branch from master, that way you will have the latest version of the code and you will be able to modify freely, without interrupting the app. When you have finished your work on the branch that you have created, you can merge it to master and that way master will have the latest modification. It's one of the common ways to change a branch, especially if you want to add big features to your application.

3.1.3 Testing

It's important before merging a branch in master to test your branch correctly by making some unit tests. Unit test is important before merging and deploying. Usually, there is a file called test that contains all the tests your code should pass before merging and deploying. They are popular libraries that can help you make these unit tests like unittest in Python.

3.1.4 Commit and Pushing

When you are modifying your branch and want to add your latest modifications, you commit your code. Commit is like a snapshot of your application, it saves all the code modifications that you made at that commit. When making a commit, you need to attach a message to it so that other developers can know exactly what you did at that commit.

Commit messages are written in a specific way, called "Conventional Commits" and it should be structured this way.

1. First we define the type of the commit , there are different types but the most used are feat, fix and BREAKING CHANGES.

2. Then we write a small description of the changes that we added.

3.2 Clean Code

3.2.1 Functions,Variables and Comments

A good piece of code is a code that is understandable by machines and humans. Naming functions and variables with appropriate and reasonable names can be important especially if you are working as a team.
When you are programming, it is recommended to automate your code. You can achieve that by writing different functions that have specific roles. This way, it can speed up the debugging process. Comments are important, it helps others to comprehend and understand what your function, code does.

3.2.2 Code Review

After finishing your code and before uploading it, one common practice in a team is to get your code reviewed by other developers. Getting your code reviewed by others can help you improve the quality of your code, detect some bugs that you didn't notice, and make sure everything works fine.

Appendix

Github

Here you can find our project on Github at the web address : github.com/dylrachwal/MCE_Project
A part of every commit is avalaible on Figure 15

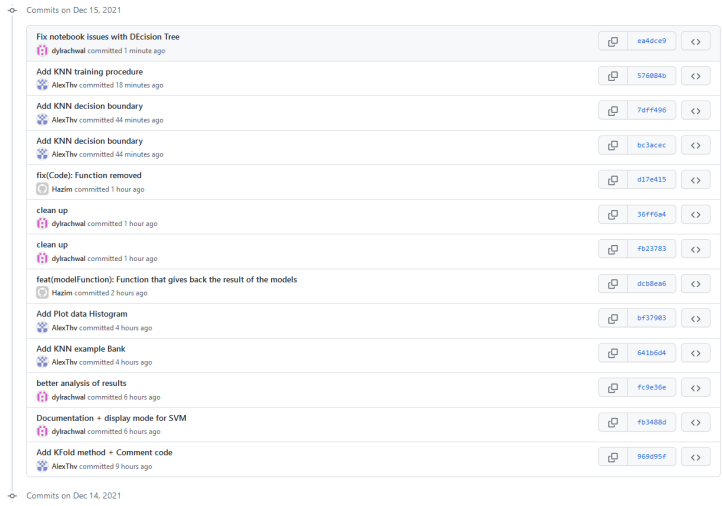


Figure 15: History of Commits

Code Python

```
# Python script for every functions used
import pandas as pd
from sklearn.model_selection import train_test_split,
                                    cross_validate
from sklearn import svm
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix,
                                    classification_report,
                                    plot_confusion_matrix,
                                    recall_score,  roc_auc_score,
                                    log_loss, f1_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score, ShuffleSplit

from sklearn.tree import export_graphviz
from graphviz import Source


def load_data(file_name, preprocess=False, columns_name=None):
    """
    Code by : Dylan
    Summary
```

```
    This function load the data and separate the labels and the
                                        features
    Parameters
    ----------
    param1 : string
        Name of the file required
    param2 : Boolean (Optional)
        Allow pre_processing or not (by Default disabled)
    Returns
    -------
    pd.Dataframe  :
        Dataframe of the different features
    np.array :
        The one-hot encoded label
    np.array :
        The different unique labels
    """
    df = pd.read_csv(file_name, sep=",", names=columns_name)
    if (preprocess):
        return pre_process(df)
    Y = df.pop(df.columns[-1]).values
    X = df
    class_labels = np.unique(Y)
    return X, Y, class_labels


def split_data(X, Y, test_size):
    """
    Code by : Dylan Rachwal
    Summary
    Split the dataset in a training and test parts
    Parameters
    ----------
    param1 : pd.Dataframe
        Dataframe
    param2 : np.array
        Labels
    param3 : float
        ratio of the test_size over the total size of features
    Returns
    -------
    pd.Dataframe  :
        Dataframe of the training part
    pd.Dataframe  :
        Dataframe  of the testing part
    np.array :
        Labels of the training part
    np.array :
        Labels of the testing part
    """
    return train_test_split(X, Y, test_size=test_size)


def predict_SVC(X_train, X_test, Y_train, kernel='linear'):
```

```
    """
    Code by : Dylan Rachwal
    Summary
    create and predict using SVM algorithm
    Parameters
    ----------
    param1 : pd.Dataframe
        Dataset for training
    param2 : pd.Dataframe
        Dataset for testing
    param3 : np.array
        labels for training
    param4 : string
        Kernel for the SVM. Can be 'linear', 'poly', 'rbf', '
                                            sigmoid', 'precomputed'
    Returns
    -------
    sklearn.svm.SVC :
        model of the SVM
    np.array :
        predicted labels of the test dataset
    """
    svc = svm.SVC(kernel=kernel)
    svc.fit(X_train, Y_train)
    return svc, svc.predict(X_test)


def predict_Random_Tree_Classification(X_train, X_test, Y_train,
                                       depth):
    """
    Code by : Christopher Jabea
    Summary
    create and predict using a Decision Tree
    Parameters
    ----------
    param1 : pd.Dataframe
        Dataset for training
    param2 : pd.Dataframe
        Dataset for testing
    param3 : np.array
        labels for training
    param4 : int
        maximal depth of each tree
    Returns
    -------
     sklearn.tree.DecisionTreeClassifier :
        model
    np.array :
        predicted labels of the test dataset
    """
    class_tree = DecisionTreeClassifier(max_depth=depth)
    class_tree.fit(X_train, Y_train)
    return class_tree, class_tree.predict(X_test)
```

```
def predict_Random_Forest_Classification(X_train, X_test, Y_train
                                         , depth, n_estimators):
    """
    Code by : Christopher Jabea
    Summary
    create and predict using a DecisionForest
    Parameters
    ----------
    param1 : pd.Dataframe
        Dataset for training
    param2 : pd.Dataframe
        Dataset for testing
    param3 : np.array
        labels for training
    param4 : int
        maximal depth of each tree
    param5 : int
        Number of Trees in the Forest
    Returns
    -------
     sklearn.tree.RandomForestClassifier :
        model
    np.array :
        predicted labels of the test dataset
    """
    class_forest = RandomForestClassifier(
        max_depth=depth, n_estimators=n_estimators)
    class_forest.fit(X_train, Y_train)
    return class_forest, class_forest.predict(X_test)


def precision_recall_multilabels(y_true, y_pred, labels):
    """
    Code by : Hazim Benslimane
    Summary
    returns the precision recall
    Parameters
    ----------
    param1 :y_true
        Real Values of Y
    param2 : y_pred
        Predicted Values of y
    param3: labels
        label of class
    Returns
    -------
    values : precision,recall
    """
    recalls = np.zeros((1, 2))
    precisions = np.zeros((1, 2))
    for label in labels:
        label_array = np.full((y_true.shape), label)
        real_association = y_true == label_array
```

```
        pred_association = label_array == y_pred
        TP = np.count_nonzero(np.logical_and(
            real_association, pred_association))
        P = np.count_nonzero(real_association)
        FN = np.count_nonzero(pred_association)
        if FN == 0:
            recalls[0, label] = 1
        else:
            recalls[0, label] = TP/FN
        if P == 0:
            precisions[0, label] = 0
        else:
            precisions[0, label] = TP/P

    return precisions, recalls


def plot_decision_tree(model, feature_names):
    """
    Code by : Christopher Jabea
    Summary
    create and display the graph of the decision tree
    Parameters
    ----------
    param1 : DecisionTreeClassifier
        Model of the Decision Tree
    param2 : List[str]
        Name of the features
    Returns
    -------
    graphviz :
        graph of the model
    """
    plot_tree = export_graphviz(
        model, out_file=None, feature_names=feature_names, filled
                                            =True)
    graph = Source(plot_tree)
    graph.render("class_tree")

    # Plot the tree
    graph
    return graph


def find_best_depths(X, Y, n_depths=10, cv=False):
    """
    Code by : Christopher Jabea
    Summary
    Algorithms to find the best depths of a decision tree
                                        classification
    Parameters
    ----------
    param1 : pd.Dataframe
        Dataframe
```

```
    param2 : np.array
        Labels
    param3 : int
        maximum depth
    param4 : Boolean
        cross validation procedure, By default None
    Returns
    -------
    np.array  :
        scores of each depth using negative log loss
    """
    cvp = None
    if(cv):
        # Define the cvp (cross-validation procedure) with random
                                          1000 samples, 2/3
                                          training size, and 1/3
                                          test size
        cvp = ShuffleSplit(
            n_splits=X.shape[0], test_size=1/3, train_size=2/3,
                                              random_state=0)

    # Define the max depths between 1 and 10
    depths = np.linspace(1, 10, n_depths)

    # Loop on the max_depth parameter and compute negative cross
                                       entropy loss.
    tab_log_tree = []
    for i in range(n_depths):
        class_tree = DecisionTreeClassifier(max_depth=depths[i])
        tab_log_tree.append(-cross_val_score(class_tree, X,
                               Y, scoring='neg_log_loss', cv=cvp))
    return tab_log_tree


def pre_process(dataframe):
    """
    Code by : Dylan
    Summary
    This function is used for the pre processing of the dataset
    Parameters
    ----------
    param1 : pd.Dataframe
        DataFrame which is cleaned and normalized
    Returns
    -------
    pd.Dataframe  :
        Dataframe of the different features
    np.array :
        The one-hot encoded label
    np.array :
        The different unique labels
    """
    nodataval = '?'
```

```
    num_samples, num_features = dataframe.shape

    # Handle nodata values
    dataframe = dataframe.replace(nodataval, np.nan)

    # Convert remaining false string columns
    other_numeric_columns = ["sg", "al", "su"]
    dataframe[other_numeric_columns] = dataframe[
                                       other_numeric_columns].apply
                                       (
        pd.to_numeric)

    # convert false sting to numeric    a voir
    false_strings = ["pcv", "wc", "rc"]
    for column in false_strings:
        dataframe[column] = pd.to_numeric(dataframe[column],
                                          errors='coerce')

    # Replace missing values
    fillna_mean_cols = pd.Index(
        set(dataframe.columns[dataframe.dtypes ==
            "float64"]) - set(other_numeric_columns)
    )
    fillna_most_cols = pd.Index(
        set(dataframe.columns[dataframe.dtypes == "object"]) |
                                          set(
            other_numeric_columns)
    )
    dataframe[fillna_mean_cols] = dataframe[fillna_mean_cols].
                                       fillna(
        dataframe[fillna_mean_cols].mean())
    dataframe[fillna_most_cols] = dataframe[fillna_most_cols].
                                       fillna(
        dataframe[fillna_most_cols].mode().iloc[0])

    # Clear '/t' and ' yes' or ' no' values
    cat_cols = [
        col for col in dataframe.columns if dataframe[col].dtype
                                          == 'object']
    for col in cat_cols:
        dataframe[col] = dataframe[col].astype(
            str).apply(lambda s: s.replace('\t', ''))
        dataframe[col] = dataframe[col].astype(
            str).apply(lambda s: s.replace(' ', ''))

    # randomize order
    dataframe = dataframe.sample(frac=1).reset_index(drop=True)

    dataframe = pd.get_dummies(dataframe, drop_first=True)
    y = dataframe.pop(dataframe.columns[-1]).values
    class_labels = np.unique(y)
    # one hot encode

    return dataframe.iloc[:, 1:], y, class_labels
```

```
def min_max_normalization(dataframe):
    """
    Code by : Alexandre Thouvenot
    Summary
    Compute KNN algorithm
    Parameters
    ----------
    param1 : pd.Dataframe
        DataFrame
    Returns
    -------
    pd.Dataframe
        Normalised dataframe
    """
    return (dataframe - dataframe.min()) / (dataframe.max() -
                                        dataframe.min())


class PCA_dec:
    """
    Code by : Alexandre Thouvenot
    Summary
    Class used to compute PCA decomosition with EVD method
    """

    def __init__(self, data):
        self.data = data
        self.cov_matrix = np.cov(np.transpose(self.data))
        self.eig_val, self.eig_vect = np.linalg.eig(self.
                                            cov_matrix)

    def exp_variance(self):
        return np.cumsum(self.eig_val) / sum(self.eig_val)

    def PCA_decomposition(self, dim):
        reduce_data = np.dot(np.transpose(
            self.eig_vect[:, :dim]), np.transpose(self.data))
        return np.transpose(reduce_data)


def KNN(X_train, Y_train, X_test, K):
    """
    Code by : Alexandre Thouvenot
    Summary
    Compute KNN algorithm
    Parameters
    ----------
    param1 : np.Array
        Array which contains training values
    param2 : np.Array
        Array which contains training values labels
    param3 : np.Array
```

```
        Array which contains test values
    param4 : Int
        Number of neighbors
    Returns
    -------
    List :
        List of index of training blocks
    List :
        List of index of test blocks
    """
    Y_test = []
    for x in X_test:
        index_list = np.argsort(np.linalg.norm((X_train-x), axis=
                                               1))[:K]
        label_list = Y_train[index_list]
        val, nb = np.unique(label_list, return_counts=True)
        Y_test.append(val[np.argmax(nb)])

    return np.array(Y_test)


def K_Fold(X, n_split):
    """
    Code by : Alexandre Thouvenot
    Summary
    Split data into n_split block to compute KFold cross
                                        validation
    Parameters
    ----------
    param1 : np.Array
        Array which contains training values
    param2 : Int
        Number of block
    Returns
    -------
    List :
        List of index of training blocks
    List :
        List of index of test blocks
    """
    n_data, _ = X.shape
    train_index_list = []
    test_index_list = []
    bloc_size = n_data // n_split
    for i in range(0, n_split):
        bloc_test = np.array(range(i * bloc_size, (i+1) *
                                           bloc_size))
        bloc_train = np.delete(
            np.array(range(0, bloc_size * n_split)), bloc_test)
        train_index_list.append(bloc_train)
        test_index_list.append(bloc_test)

    return train_index_list, test_index_list
```

```
def display_data_histogram(X, x_plot, y_plot):
    """
    Code by : Alexandre Thouvenot
    Summary
    Plot dataframe features histogram
    Parameters
    ----------
    param1 : pd.Dataframe
        DataFrame
    param2 : Int
        x size of subplot
    param2 : Int
        y size of subplot
    Returns
    -------
    plt.axs :
        Figure of multiple histogram
    """
    fig, axs = plt.subplots(x_plot, y_plot, figsize=(10, 10))
    for i, column in enumerate(X.columns):
        axs[i//y_plot, i % y_plot].hist(X[column].values, bins=20
                                        )
        axs[i//y_plot, i % y_plot].set_title('Column ' + column)
        axs[i//y_plot, i % y_plot].set(xlabel='Value', ylabel='Nb
                                       element')
    fig.tight_layout()
    return axs


def display_confusion_matrix(model, X_test, Y_test, Y_pred):
    """
    Code by : Dylan Rachwal
    Summary
    Plot and return the confusion matrix
    Parameters
    ----------
    param1 : sklearn.model
        model used to predict values
    param2 : pd.Dataframe
        Dataset for testing
    param3 : np.array
        labels for testing
    param4 : np.array
        labels predicted
    Returns
    -------
    np.array :
        Confusion_matrix of the model
    """
    confusion_matrix_test = confusion_matrix(Y_test, Y_pred)
    plot_confusion_matrix(model, X_test, Y_test)
    return confusion_matrix_test
```

```
def multiple_prediction_scores(model, X, Y, cv, scoring):
    """
    Code by : Dylan Rachwal
    Summary
    Return the wanted scores of the model using cross validation
    Parameters
    ----------
    param1 : sklearn.model
        model used to predict values
    param2 : pd.Dataframe
        Dataset
    param3 : np.array
        labels
    param4 : Int
        size of the cross validation
    param5 : List[str]
        List of the scores
    Returns
    -------
    np.array :
        Confusion_matrix of the model
    """
    return cross_validate(model, X, Y, cv=cv,  scoring=scoring)

def train_knn_kfold(X,Y,n_split,K, class_labels):
    """
    Code by : Alexandre Thouvenot
    Summary
    Train KNN with KFold procedure
    Parameters
    ----------
    param1 : np.array
        Input value
    param2 : np.array
        Label
    param3 : Int
        Block of KFold procedure
    param4 : Int
        K number of neighbour
    param5 : List
        Class label
    Returns
    -------
    np.array :
        Precision and recall of each label
    """
    train_index_list, test_index_list = K_Fold(X, n_split)
    accuracy = np.zeros((1,2))
    recall = np.zeros((1,2))
    for index_train, index_test in zip(train_index_list,
                                       test_index_list):
        X_train = X[index_train,:]
        Y_train = Y[index_train]
```

```
        X_test = X[index_test,:]
        Y_test = Y[index_test]
        Y_pred = KNN(X_train, Y_train, X_test, K)
        res = precision_recall_multilabels(Y_test, Y_pred,
                                           class_labels)
        accuracy += res[0]
        recall += res[1]
    return accuracy/n_split, recall/n_split
```