IMT Atlantique
Technopôle de Brest-Iroise - CS 83818
29238 Brest Cedex 3
Téléphone : +33 (0)2 29 00 13 04
Télécopie : +33 (0)2 29 00 10 12
URL : **www.imt-atlantique.fr**

Mini project report

Machine Learning : Binary Classification

CIOCARLAN Alina
PYTHOUD Axel

Date d'édition : 6 décembre 2020
Version : 1

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

Sommaire

1. Introduction

This project aims to create a machine learning workflow applicable to different binary classification problems and datasets with minimal manual processing. Our workflow is ordered this way :

1. Import the dataset
2. Clean the dataset
3. Split the dataset
4. Train the model
5. Validate the model

The first step allows us to import the files of the data set. There may be multiple files, and the files may be of different types. Then, we clean the dataset by adding missing values, normalizing the data, removing unnecessary columns... Once we have a cleaned dataset, we can split it, which allows us to train our model. We chose to train deep neural networks, with feature reduction through Principal Component Analysis. Some hyperparameters will be chosen through grid search. After the training of the model, we have to validate it, which will allow us to see how well it performs. This is done with k-fold cross validation.

We expect that manual input will always happen at step 1, at the very least to give the file name of the dataset, and perhaps to take into account other file types and implement importing multiple files into the same dataset. Manual input may also happen at step 2, in order to standardize the format of the dataset, and at step 4, in order to fine tune the model and obtain better results.

We will test our workflow on two datasets.

1. Banknote Authentification : 1372 instances x 5 features. Four features have been obtained from pictures of genuine and forged banknotes, with the last one being the classification between real and fake. All values are numeric, and none are missing. Available here : https ://archive.ics.uci.edu/ml/datasets/banknote+authentication
2. Chronic Kidney Disease : 400 instances x 25 features. Twenty four features have been obtained from patient affected and unaffected by chronic kidney disease, with the last one being diagnosis of chronic kidney disease. Some values are numeric, others are text booleans. Some values are missing. Available here : https ://archive.ics.uci.edu/ml/datasets/Chronic_Kidney_Disease

2. Data preprocessing

Our goal in this part is to import the dataset into a clean Pandas dataframe. This choice is motivated by the ease of use of Pandas for data science, and its popularity which means there are a lot of guides online.

2.1. Importing the original dataset

For some tasks, it is possible, and easier, to modify the data file rather than the dataframe. This means that we will have an original dataset, and a modified dataset. Once we have computed the modified dataset, we will not need to compute it again, and can simply import the modified dataset instead of the original one.

The chronic kidney disease file is a .csv file with commas between its values, and a description. It already has an id column that will be automatically added by Pandas.

The banknote authentication file is a .txt file. However, it is formatted exactly as a CSV file, with commas between its value. The description is missing.

This means we can import both as CSV files, which is done by a Pandas function.

2.2. Standardising the dataset

The import of the datasets into Pandas dataframes has a few issues.

1. The banknote authentication dataset has no description. This means that its first instance is considered as a description by Pandas. As this would remove one of our data points, and make it harder to understand what the data means, we add a description to the file.
2. The chronic kidney file has an id column which is also added by Pandas. This means we should remove it, especially considering that the id is ordered by the class of the instance, and it would lead our model to just put a threshold based on the id value and call it a day.
3. The chronic kidney file has some characters that should not exist in its values. Most missing values are null character strings, which is expected, but some are spaces, tabs, or question marks. These characters are sometimes added to actual values. As there is no use for any of them in the file, we remove all of them.

We save these modifications, and we will only import the file containing those from this point forward.

2.3. Cleaning the dataset

Our goal here is to clean the dataset to make it usable by the deep learning model.

1. If a column has more than 30% of its values missing, we drop the column.
2. We find the mean of each column, or the most present value for text columns, and replace the missing values of the column by its mean.
3. We find the variance of each numeric column to center and reduce it.
4. We replace the text values by numerical values, as text values are not compatible with the rest of the process.

It would be possible to save the dataset here instead of earlier, and it would save some time for subsequent attempts on the same dataset. However, our reason for saving the dataset earlier was not to gain time, but because standardising the text was easier than standardising the dataframe. Therefore, we did not choose to save here again.

3. Machine learning workflow

We are going to describe the Machine Learning workflow we chose. We will in particular explain our learning strategy and the metrics we used to evaluate our models.

3.1. Feature reduction

As we can see on our datasets, especially the Chronic Kidney Disease's (CKD) one, there are a lot of features. Indeed, there are 24 columns for the features. This may be a problem for our neural network (or any classification algorithm) : having a high search dimension increases the complexity of the algorithm and its run-time. The neural network will also require a huge amount of data to converge without under-fitting. Therefore, it is essential to define a strategy to reduce the features' space. We have already dropped one column during the preprocessing phase, as more than 30% of its lines where empty. We will use a Principal Component Analysis (PCA) algorithm to drop redundant informations. We already have a standardize dataset thanks to the pre-processing step, which is essential to apply a PCA algorithm without suffering from high variance.

PCA is an unsupervised technique which aims to make the high variability of the data more visible by rotating the axes. It consists in the calculation of the eigenvalues and eigenvectors of our feature matrix.

After that, it calculates the variance ratio of each rotated feature and ranges it in a decreasing order. We then select *n* principal components, depending on how much relevant information we want to keep. In our case, we defined a threshold of 90% of cumulative variance ratio. Performing a PCA on the CKD dataset yields to the selection of 10 principal components, which allows us to reduce a lot the complexity of our dataset. For the Bank Note Dasaset, it reduces the features' dimension of 1.

3.2. Neural network model

As we deal with numerical data (nor images or temporal data), one of the most appropriate layers to use here is the dense layer (also called fully-connected layer). In order to get a better precision in the predictions, we are going to use 3 dense layers. The first 2 layers will use "ReLU" as an activation function, which is the most commonly used activation function in hidden layers. The last dense layer, composed of only one neuron (binary classification), will be followed by the common sigmoid activation function.

In our model (see annex X), you can notice 2 particular layers : the batch normalization one and the dropout one. Batch normalization standardizes the input (batch) of the layer, while the dropout layer allows us to ignore the prediction of a part of the neurons (only during training), 15% of them here. They both help our algorithm to avoid overfitting, to be more stable and to better generalize the results on another dataset.

Some parameters will be chosen later through a gridsearch (see next part). We decided to perform this strategy on the number of epoch, the batch size, and the optimizer. We won't perform gridsearch on the activation function for example as they may not have a very significant impact compared with the number of epochs or the batch size.

3.3. Training and testing strategy

A common practice is to split the dataset into a training, a validation and a test set. Each dataset has a precise function :

- Training set : train the weights of the neural network model chosen.

- Validation set : used as a test set to fine-tune the model (ie choose the best parameters or training strategy in order to get the best results).

- Test set : only used to test the final model, once we had optimize the training parameters with the validation test. It allows use to confirm the actual performance of our network.

Our strategy is here to divide the dataset in two : 75% for training/validation set, and 25% for the test set. We decided to take a bigger test set in order to challenge our model and to see if it can easily generalize its results on an unseen dataset. On the training/validation set, we are going to train and fine-tune a model. We will do that in 2 main steps :

- Step 1 : K-Fold cross-validation and Gridsearch (Sklearn function) in order to find which parameters lead to the highest accuracy and the lowest loss. We will split the set in 10 folds, as it is a common value used for this kind of cross-validation. The Gridsearch will be performed on 3 relevant parameters : the number of epochs, the batch size and the kind of optimizer. Playing on the number of epochs and on the batch size will have an impact on the quality of the convergence and also on its speed. The choice of the optimizer depends on the dataset's characteristics.

- Step 2 : K-Fold cross-validation using the best parameters found in first step. We will keep the weights of the model which achieved the best accuracy and the lowest lost.

Once we have found our best model, we are going to train it on the whole dataset (it was trained on only 90% of the training/validation set) and then we are going to test it on the unseen test set to evaluate it.

3.4. Metrics

One of the most common metrics is the accuracy. In our network, this metric is easy to access through the sklearn functions. The highest the accuracy of a network is, the more its answers are correct. This metric is simple and intuitive. However, it has many drawbacks when we consider a dataset with unbalanced classes. Therefore, we need to look at other metrics to evaluate our model. We will look at the loss of our network, and at the precision, recall and F1-score. A small lost indicates that the network has a good convergence, so we want to minimize this parameter. The other 3 metrics' meaning depend on the numbers FP (false positive), FN (false negative), TN (true negative) and TP (true positive).

For example, in the CDK dataset, we want to minimize the FN, ie the number of person not considered as sick, while they really are sick. Therefore, it is essential for us to maximize the recall (which is equal to TP/(TP+FN). It is also good to maximize precision (ie minimize FP). As F1-score is a mean of precision and recall, it's also important to maximize F1-score.

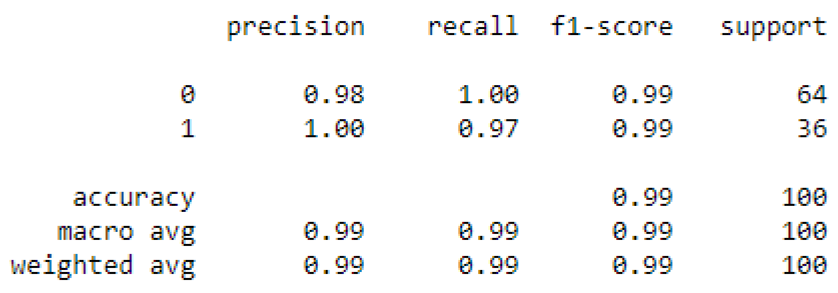
4. Results : best model validation

We have tested our workflow using a RTX 2060 and having 8 Gb of memory. As our CPU is limited in memory, there are some tasks that we couldn't parallelize some tasks, for example the Gridsearch. Therefore, the fine-tuning phase takes a longer time. Applying our workflow takes approximately 10 minutes for each dataset.

4.1. CKD Dataset

For each step of our training phase, we obtained the following results :

1. Fine-tuning : 100 for the best epoch number, 40 for the batch size and the best optimizer is 'Adamax'.
2. Best model : 99% of accuracy and a loss of 0.038. We also obtain the following scores :

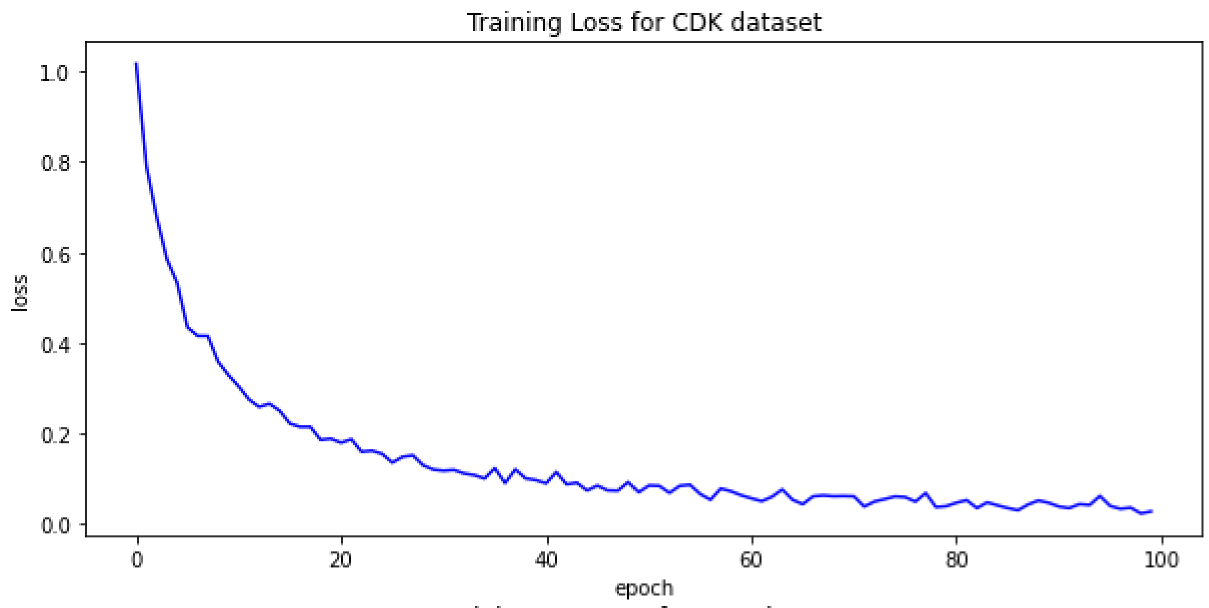


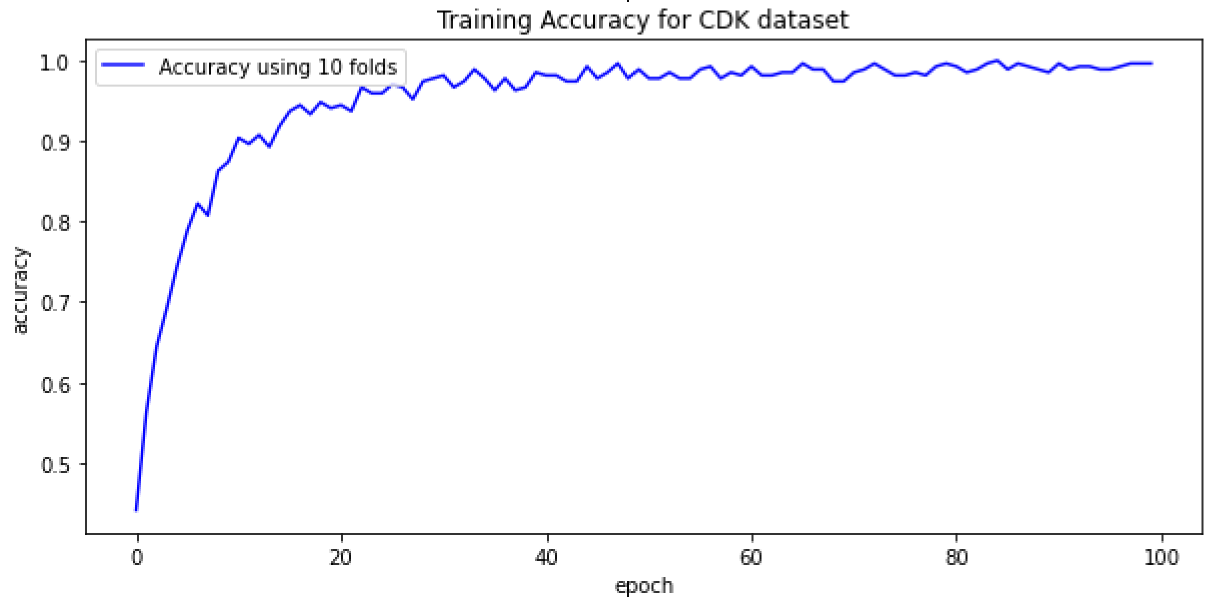
```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        64
           1       1.00      0.97      0.99        36

    accuracy                           0.99       100
   macro avg       0.99      0.99      0.99       100
weighted avg       0.99      0.99      0.99       100
```

Our neural network performed very well on this dataset. The accuracy is very high. Moreover, when we look at the row 0 of the classification report, which corresponds to the label "ckd", we notice hat the recall is equal to 1, which is a good news because this means that we never misclassified someone that has a CKD. Thus, we can conclude that this neural network is very powerful for this dataset.





If we take a look at the learning curves, we can see that both loss and accuracy have converged to their final value. That means that we do not need to train our network on this dataset anymore (we won't get significantly better results by doing so).

4.2. Banknote Dataset

For each step of our training phase, we obtained the following results :

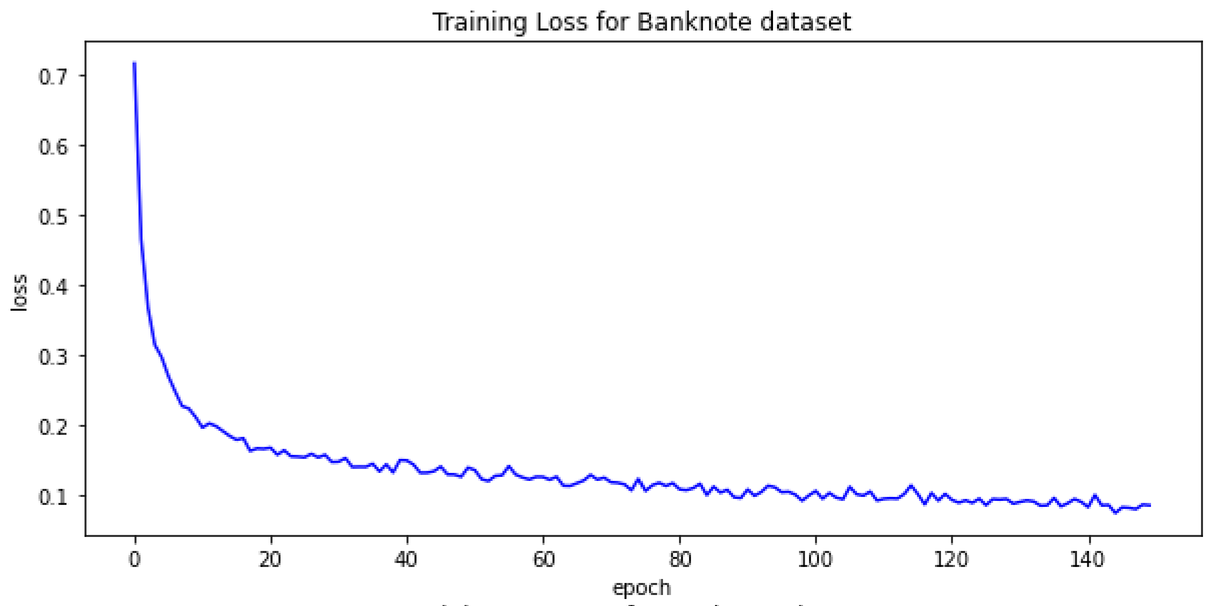
1. Fine-tuning : 150 for the best epoch number, 100 for the batch size and the best optimizer is 'Adam'.
2. Best model : 95% of accuracy and a loss of 0.08. We also obtain the following scores :

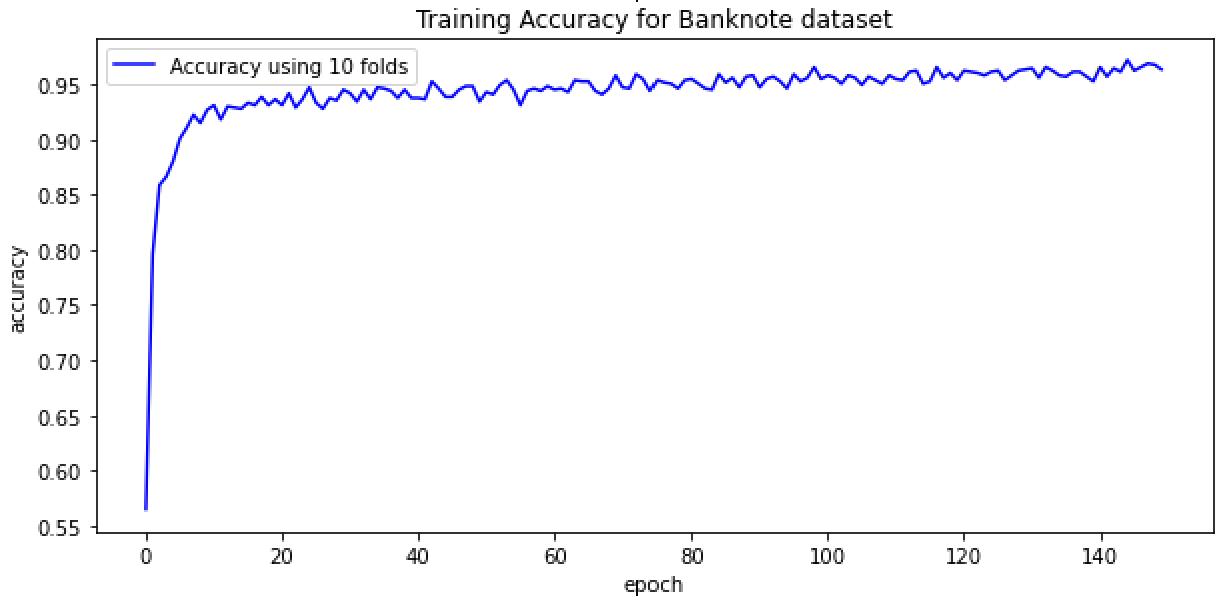
```
              precision    recall  f1-score   support

           0       0.98      0.93      0.95       191
           1       0.91      0.98      0.95       152

    accuracy                           0.95       343
   macro avg       0.95      0.95      0.95       343
weighted avg       0.95      0.95      0.95       343
```

Our neural network did not perform that well on this dataset. We expected a better accuracy, even if it is not that bad. The scores from the classification report are correct, but it could have been better.



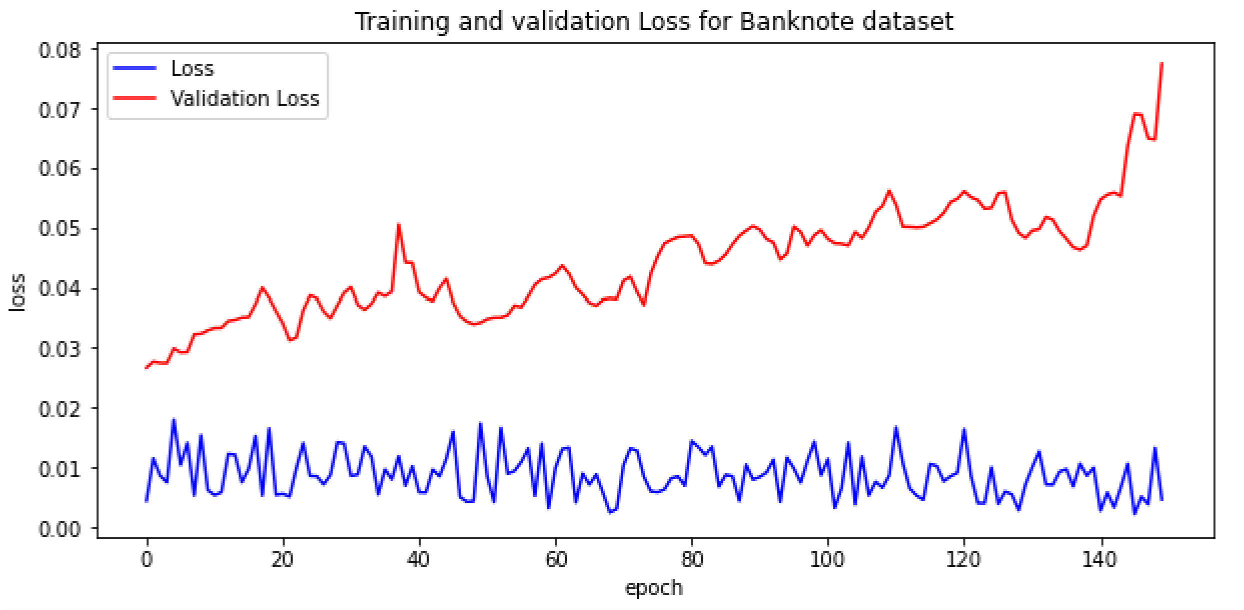


If we take a look at the learning curves, we can see that both loss and accuracy haven't totally converged.

Indeed, the curves are still increasing when approaching epoch 150. This means that our network may need a little more time to learn.

Therefore, we tried to launch again the training phase for 1000 epochs. The accuracy is better (97,3%) and the scores are also better. This confirms the fact that the network needed more time to learn. However, we think that the network can be modified in order to make it converge faster. Moreover, even if we keep increasing the number of epochs, we don't get a better accuracy than 97%. We can probably find a model that will be more powerful for this dataset.

We can also improve the analyse of our network for this dataset. Indeed, we can see that the training accuracy (99%) is higher than the test accuracy. We can wonder whether our network is overfitting or not. To do that, we need to modify our function *best-model-fit-eval()* and add a validation step (so we also need to create a validation set). Then, we need to compare the training and the validation curve. When the validation loss begins to increase, that means that the model is overfitting. Let's apply this to our banknote strategy. We obtain the following curves :



That is exactly what is happening here. We can see that before training on the whole dataset, the validation loss already is above the training loss. Thus, our model is severely overfitting.

This highlights one of the major problems of our model. Indeed, it may lead to overfitting. Maybe we can solve this by adding more dropouts or by making a simpler model (less layers for example).

5. Conclusion and good practices to adopt

Our aim was to create a machine learning workflow for binary classification problems. As we can see, aside from the preprocessing part, we applied the exact same process for both of the datasets. However, as planned, the model is not perfect for the bank authentication dataset, as the No Free Lunch theorem states that no algorithm is the best for all problems. This means that we could get a better result by manually fine tuning the model.

As we worked as two people, the use of git was important to share our work. Here, it was easily splittable, so we did not face the trouble of merges. The code also wasn't particularly long or hard to write, so reverts were not important either. However, it is easy to see how git would be even more crucial if we had a longer, harder project with more people. Moreover, we should think on using several branches on git for larger project. In any case, it is essential to discuss with the whole team before beginning the project on how we should organize the work and the working space.

Aside from the technical part, an understanding of the end goal and good communication are important. Each step of the workflow should be done thinking of the next, so that it will fit easily without too much trouble. In the worst case scenario, we would have to write an adapter for functions within the same file. Here, the basic conception was already done, and we discussed how we were going to implement the workflow.

It is also important to properly separate the code into small understandable parts. A large function that does everything is hard to test, hard to debug, and hard to understand. A large function that calls several smaller functions is easier to understand, and each part can be tested and debugged individually before testing the larger function. Comments and function description (input and output variables) are also important to help communicate the intent of a function. We can refer to the PEP8 standard. Finally, writing a README file to explain how to run the project is a must.

An important thing to take into account is the libraries' versions we are using and the material we use to test our neural network, especially for projects you want to share with a company or with the scientific community. They will have an impact on the efficiency of the code, and it will also make it easier to re-use without having errors. Moreover, it is important to create different environments for each related project on our local computers in order to avoid libraries' conflicts.

It would have been a good idea to have tests to check if everything is working properly, instead of testing on the fly. We can consider that our test on both datasets constitutes the global test of our code, however we do not have individual tests. In a larger project, testing only on the fly would not be possible, and would result in many problems.

Annexes

Annexe 1 – Logs

Git repository :

`https://github.com/apythoud/imt-ml-project-group15`

Annexe 2 – Codes

2.1. Imports

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pandas.api.types import is_numeric_dtype
import os.path
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout, BatchNormalization
from tensorflow.keras.losses import binary_crossentropy
from sklearn.model_selection import GridSearchCV
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
```

2.2. Preprocessing functions

```
def hf_add_definitions(input_file_name, output_file_name, definition) :  # Axel PYTHOUD
    input_file = open(input_file_name,"r")
    text = definition + "\n" + input_file.read()
    input_file.close()

    output_file = open(output_file_name, "w")
    output_file.write(text)
    output_file.close()


def hf_remove_characters(text, characters) :  # Axel PYTHOUD

    for character in characters :
        text = text.replace(character,'')
    return text

def hf_remove_id(input_file_name) :  # Axel PYTHOUD
    input_file = open(input_file_name,"r")
    text = ""
    for line in input_file :
        text += line[line.index(',') + 1:]
    input_file.close()
    return text

def hf_prepare_kidney_file(input_file_name, output_file_name) :  # Axel PYTHOUD
    text = hf_remove_id(input_file_name) #This removes the id column
    text = hf_remove_characters(text, ["\t", "?", ' ']) #This removes tabs,
    # question marks, and spaces from the file
    output_file = open(output_file_name, "w")
    output_file.write(text)
    output_file.close()

unprepared_banknote_file_name = "data_banknote_authentication.txt"
banknote_file_name = "data_banknote_authentication_with_def.csv"
if not(os.path.exists(banknote_file_name)) :
    hf_add_definitions(unprepared_banknote_file_name, banknote_file_name,
    "variance,skewness,curtosis,entropy,class") #This adds a definition to the file

unprepared_kidney_file_name = "archive/kidney_disease.csv"
kidney_file_name = "archive/kidney_disease_cleaned.csv"
if not(os.path.exists(kidney_file_name)) :
    hf_prepare_kidney_file(unprepared_kidney_file_name, kidney_file_name) #This
    #removes tabs, question marks, and spaces from the file


## Import the files into pandas dataframes

def import_file(file_name, separator) :     # Axel PYTHOUD
    pd_data = pd.read_csv(file_name, sep = separator)
    return pd_data

banknote_pd_data = import_file(banknote_file_name, ",")
kidney_pd_data = import_file(kidney_file_name, ",")


## Clean dataframes
# Add missing values
# Center and reduce columns

def hf_get_mean_value(column) :  # Axel PYTHOUD
    #Help function that returns the mean value of the column.
    #For non numeric data types, returns the most frequent value.
    if is_numeric_dtype(column) :
        return column.mean()
    else :
        values = pd.value_counts(column)
        return values.idxmax()

def drop_column(pd_data):  #Alina CIOCARLAN
```

```
    """ Objective : drop columns that have more than 30% of missing values
        Input : dataframe to pre-process (type : pandas dataframe)
        Output : dataframe (type : pandas dataframe)"""
    c = pd_data.columns
    #calculate the percrentage of each column that is empty
    empty = (pd_data.isnull().sum() / len(pd_data)) * 100

    for i in range(len(empty)):
        if empty[i] > 30: #drop column if percentage of missing values is above 30%
            c_to_drop = c[i]
            pd_data = pd_data.drop(c_to_drop, 1)
    return pd_data


def clean_dataframe(pd_data) :  # Axel PYTHOUD
    pd_data = drop_column(pd_data)

    column_names = pd_data.columns
    number_of_columns = column_names.size
    means = []

    for col_name in column_names :
        means.append(hf_get_mean_value(pd_data[col_name]))

    # We have the means of each column of the dataset
    # Now we find the cells that are not filled, and replace them with the
    # mean value of the column.
    null_data = np.where(pd.isnull(pd_data))
    for i in range(len(null_data[0])) :
        row = null_data[0][i]
        col_id = null_data[1][i]
        col = column_names[col_id]
        pd_data.at[row,col] = means[col_id]

    # Our cells are all filled now.
    # We can center and reduce the values of the numeric columns
    L=list(pd_data.columns)[:-1] #function applied on all numerical values
    # except the last class
    pd_data[L] = pd_data[L].apply(lambda x : (x - x.mean()) / np.sqrt(x.var() + 10**-9)
    if is_numeric_dtype(x) else x) #The value 10**-9 is a safety to
    # ensure we don't divide by 0.

    return pd_data

def label_encoding(pd_df): #Alina CIOCARLAN
    """ Objective : encode the columns that have non-numerical values
        Input : dataframe to pre-process (type : pandas dataframe)
        Output : dictionnary with the labels that were changed into
        numerical values (type : dictionnary)"""
    c=pd_df.columns
    dict_class=dict()
    types=list(pd_df.dtypes)
    for i in range(len(c)):
        if types[i] == 'O': #there are only 2 types here, if it's of type 'O',
        #we have to transform it into numerical
            lab_encod = LabelEncoder()

            pd_df[c[i]] = lab_encod.fit_transform(pd_df[c[i]])
            dict_class[c[i]]=lab_encod.classes_ #keep a dictionnary of the
            # labels to keep track of them if needed
    return dict_class
```

2.3. Feature reduction

```
def PCA_on_df(df,seuil): #Alina CIOCARLAN
    """ Objective : Apply PCA to reduce dimension
        Inputs : - dataframe to pre-process (type : pandas dataframe)
                 - threshold of the PCA (type : float between 0 and 1)
        Output : transformed dataframe (type : pandas dataframe)"""
    L=list(df.columns)[:-1] # we do not apply the PCA on the column "labels"
    df_pca=df[L]
    pca=PCA()
    pca.fit(df_pca)
    df_pca=pca.transform(df_pca)
    cumulative_pca=pca.explained_variance_ratio_.cumsum()
    n=0
    for e in cumulative_pca: #take the columns that lead to the highest variance ratio
        if e<seuil:
            n+=1
    df_pca = df_pca[:, :n]
    return pd.DataFrame(df_pca)
```

2.4. Training functions

2.4.1. Split the dataset

```
# we split the dataset in 2 : one part for training/validation set (we'll
# do K-fold validation right after), the other for test set
def data_split(dataX,dataY,train_ratio): #Alina CIOCARLAN
    """ Objective : Split the dataset in 2
        Inputs : - dataframe to split (type : pandas dataframe)
                 - training set ratio (type : float between 0 and 1)
        Outputs : - training feature (type : numpy array)
                    - test feature (type : numpy array)
                    - training labels (type : numpy array)
                    - test labels (type : numpy array) """
    x_train, x_test, y_train, y_test = train_test_split(dataX,dataY ,
    test_size=1 - train_ratio)

    return x_train, x_test, y_train, y_test
```

2.4.2. Fine tune the model

This step is performed using a gridsearch.

```
def get_best_epoch_batch(x_train, y_train): # Alina CIOCARLAN
    """ Objective : Get the best epoch and batch size with GridSearch
        Inputs : - training features (type : numpy array)
                - training labels (type : numpy array)
        Outputs : - batch size (type : int)
                    - epoch number (type : int) """
    # define the model
    def create_model():
        model=Sequential()
        model.add(Dense(64, input_dim=x_train.shape[1], activation='relu'))
        model.add(BatchNormalization())
        model.add(Dense(16, activation='relu'))
        model.add(Dropout(.15))
        model.add(Dense(1, activation='hard_sigmoid'))
        model.compile(loss='binary_crossentropy', optimizer='Nadam', metrics=['accuracy'])
        return model

    # gridsearch through Keras wrapper
    model_grid_search = KerasClassifier(build_fn=create_model, verbose=0)

    # grid search parameters
    if x_train.shape[0]<500: #if small dataset, we can perform more grid
    # search and we may need less epochs to converge
        batch_size = [20, 40, 60, 80]
        epochs = [75, 100, 125]

    else: #otherwise, perform less gridsearch and take more epoch
        batch_size = [40, 70, 100]
        epochs = [75, 100, 125, 150]

    param_grid = dict(batch_size=batch_size, epochs=epochs)

    print("==================================================")
    print("Looking for best epoch number and batch size.......")

    grid = GridSearchCV(estimator=model_grid_search, param_grid=param_grid, n_jobs=1)
    # n_jobs=-1 parallélise les opérations en distribuant les tâches sur les
    # coeurs du GPU. A utiliser seulement si on a un bon GPU et assez de RAM *
    # (j'ai i5 9, 8Go de RAM, et ça a planté). Sinon utiliser n_job=1.

    #perform grid search and 5-fold crossval
    grid_result = grid.fit(x_train, y_train)

    # summarize results
    print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
    return grid_result.best_params_['batch_size'],grid_result.best_params_['epochs']
```

```
def get_best_optimizer(x_train, y_train): # Alina CIOCARLAN
    """ Objective : Get the best optimizer with GridSearch
        Inputs : - training features (type : numpy array)
                 - training labels (type : numpy array)
        Output : best optimizer (type : string) """
    def create_model_opt(optimizer='adam'):
        model=Sequential()
        model.add(Dense(64, input_dim=x_train.shape[1], activation='relu'))
        model.add(BatchNormalization())
        model.add(Dense(16, activation='relu'))
        model.add(Dropout(.15))
        model.add(Dense(1, activation='sigmoid'))
        model.compile(loss='binary_crossentropy', optimizer=optimizer,
        metrics=['accuracy'])
        return model
    model_grid_search_opt = KerasClassifier(build_fn=create_model_opt,epochs=100,
    batch_size=40, verbose=0)

    # grid search parameters

    print("====================================================")
    print("Looking for the best optimizer.......")

    optimizer_list = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax',
    'Nadam']
    param_grid = dict(optimizer=optimizer_list)

    grid = GridSearchCV(estimator=model_grid_search_opt, param_grid=param_grid,
    n_jobs=1) #perform grid search and K-fold
    grid_result = grid.fit(x_train, y_train)

    # summarize results
    print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
    return grid_result.best_params_['optimizer']
```

2.4.3. Best model selection

We select the best model with best parameters found before using a K-fold cross-validation.

```
def K_fold_cross_val(x_train,y_train, batch, epoch, optimizer,folds): # Alina CIOCARLAN
    """ Objective : Get the best model with the best parameters with a
    K-fold cross-valisation
        Inputs : - training features (type : numpy array)
                 - training labels (type : numpy array)
                 - batch (type : int)
                 - epoch number (type : int)
                 - best optimizer (type : string)
                 - number of folds for cross-val (type : int)

        Outputs : - best model (type : Keras object)
                    - history of the best model (type : Keras object) """
    kf = KFold(n_splits=folds,shuffle=True)
    # lists eo save the scores
    acc_per_fold = []
    loss_per_fold = []

    fold_nb = 1

    # store the models
    models = []
    history=[]

    print("============================================================")
    print("Looking for the best model using the best parameters.......")
    for train, test in kf.split(x_train, y_train):
      # Define the model
        model=Sequential()
        model.add(Dense(64, input_dim=x_train.shape[1], activation='relu'))
        model.add(BatchNormalization())  # add batch normalization to avoid overfitting
        model.add(Dense(16, activation='relu'))
        model.add(Dropout(.15)) # same effect as batch normalization but the concept
        # is different : we don't count the answer of 15% of the neurons
        # (randomly chosen) during the training only
        model.add(Dense(1, activation='sigmoid'))

        model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
        models.append(model)

        print('------------------------------------------------------------------------')
        print(f'Training for fold {fold_nb} ...')

      # Fit data to model using train index of the current fold
        history.append(models[fold_nb-1].fit(x_train[train],
        y_train[train],epochs=epoch, batch_size=batch))

      # evaluate and get metrics using test index of the current fold
        scores = models[fold_nb-1].evaluate(x_train[test], y_train[test], verbose = 0)
        y_pred=models[fold_nb-1].predict_classes(x_train[test])

        #print de scores for this fold
        acc_per_fold.append(scores[1] * 100)
        loss_per_fold.append(scores[0])

      # Increase fold number
        fold_nb = fold_nb + 1
    index_min=loss_per_fold.index(np.min(loss_per_fold))
    return models[index_min], history[index_min]
```

2.4.4. Train the best model on the whole dataset

```
def best_model_fit_eval(model,x_train, y_train, x_test, y_test, batch, epoch,
optimizer): # Alina CIOCARLAN
```

```
    """ Objective : Fit the best model
        Inputs :- best model (type : Keras object)
                - training features (type : numpy array)
                - training labels (type : numpy array)
                - test features (type : numpy array)
                - test labels (type : numpy array)
                - batch (type : int)
                - epoch number (type : int)
                - best optimizer (type : string)

        Outputs :- loss of the network (type : list)
                 - accuracy of the network (type : list) """
    loss=[]
    acc=[]
    # Fit data to model
    print("===============================================")
    print("Training and evaluating the best model.......")
    history_best_model = model.fit(x_train, y_train,epochs=epoch, batch_size=batch)
    loss.append(history_best_model.history['loss'])
    acc.append(history_best_model.history['accuracy'])


      # evaluate and get metrics
    scores = model.evaluate(x_test, y_test, verbose=0)
    y_pred=model.predict_classes(x_test)

    print("===============================================")
    print("FINAL SCORE")
    print(f'Score for best model: {model.metrics_names[0]} of {scores[0]};
    {model.metrics_names[1]} of {scores[1]*100}%')

    print(classification_report(y_test, y_pred))

    print("===============================================")
    return loss, acc
```

2.5. Tests on our 2 datasets

2.5.1. Tests on CDK dataset

```
# the tests were done using => CPU : intel i5 9th gen, RAM : 8 Go, GPU : RTX 2060
# cannot parallelize gridsearch on CPU so takes more time.

# TESTS ON CHRONIC KIDNEY DISEASE DATASET
# Alina CIOCARLAN

#pre processing

kidney_pd_data = clean_dataframe(kidney_pd_data)
dict_class_encod = label_encoding(kidney_pd_data)

# feature selection
```

```
df_proj=PCA_on_df(kidney_pd_data,0.9)

# split dataset kidney disease
dataX=np.array(df_proj)
dataY=np.array(kidney_pd_data['classification'])
train_ratio = 0.75
folds=10

# fine tuning
# get best params
kdn_x_train, kdn_x_test, kdn_y_train, kdn_y_test=data_split(dataX,dataY,train_ratio)
batch, epoch = get_best_epoch_batch(kdn_x_train,kdn_y_train)
optimizer = get_best_optimizer(kdn_x_train,kdn_y_train)

# get best model performing K-fold cross validation
best_model, history = K_fold_cross_val(kdn_x_train,kdn_y_train, batch, epoch,
optimizer,folds)

# evaluate this model
cdk_loss_10,cdk_acc_10 = best_model_fit_eval(best_model,kdn_x_train,kdn_y_train,
kdn_x_test,kdn_y_test, batch, epoch, optimizer)


# Plot the learning curves /  Alina CIOCARLAN

loss= history.history['loss']
acc= history.history['accuracy']

# plot figures to compare performance between 10-fold crossval or 5-fold crossval
plt.figure(figsize = (10, 10))
plt.subplot(2, 1, 1)
plt.xlabel('epoch')
plt.ylabel('loss')
plt.plot(loss, 'b', label='Loss using 10 folds')
plt.title("Training Loss for CDK dataset")

plt.subplot(2, 1, 2)
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.plot(acc, 'b', label='Accuracy using 10 folds')
plt.title("Training Accuracy for CDK dataset")
plt.legend()

plt.show()
```

2.5.2. Tests on the banknote dataset

```
# TESTS ON BANKNOTE DATASET

# Alina CIOCARLAN
# pre processing
banknote_pd_data = clean_dataframe(banknote_pd_data)

# feature selection
df_proj_bank=PCA_on_df(banknote_pd_data,0.97)

# split dataset kidney disease
dataX=np.array(df_proj_bank)
dataY=np.array(banknote_pd_data['class'])
train_ratio = 0.75
folds=10
# fine tuning
# get best params
kdn_x_train, kdn_x_test, kdn_y_train, kdn_y_test=data_split(dataX,dataY,train_ratio)
batch, epoch = get_best_epoch_batch(kdn_x_train,kdn_y_train)
optimizer = get_best_optimizer(kdn_x_train,kdn_y_train)

# get best model performing K-fold cross validation
best_model, bank_history = K_fold_cross_val(kdn_x_train,kdn_y_train, batch, epoch,
optimizer,folds)

# evaluate this model
bank_loss_10, bank_acc_10 = best_model_fit_eval(best_model,kdn_x_train,kdn_y_train,
kdn_x_test,kdn_y_test, batch, epoch, optimizer)

# plot the curves
history_best_model = best_model.fit(kdn_x_train, kdn_y_train, validation_split=0.33,
epochs=epoch, batch_size=batch)
loss = history_best_model.history['loss']
acc=history_best_model.history['accuracy']
val_loss=history_best_model.history['val_loss']
val_acc=history_best_model.history['val_accuracy']

plt.figure(figsize = (10, 10))
plt.subplot(2, 1, 1)
plt.xlabel('epoch')
plt.ylabel('loss')
plt.plot(loss, 'b', label='Loss')
plt.plot(val_loss, 'r', label='Validation Loss')
plt.title("Training and validation Loss for Banknote dataset")
plt.legend()
```

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	**Campus de Nantes**	**Campus de Rennes**	**Site de Toulouse**
Technopôle Brest-Iroise	4, rue Alfred Kastler	2, rue de la Châtaigneraie	10, avenue Édouard Belin
CS 83818	CS 20722	CS 17607	BP 44004
29238 Brest Cedex 3	44307 Nantes Cedex 3	35576 Cesson Sévigné Cedex	31028 Toulouse Cedex 04
France	France	France	France
T +33 (0)2 29 00 11 11	T +33 (0)2 51 85 81 00	T +33 (0)2 99 12 70 00	T +33 (0)5 61 33 83 65
F +33 (0)2 29 00 10 00	F +33 (0)2 99 12 70 08	F +33 (0)2 51 85 81 99	

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

© IMT Atlantique, 2019
Imprimé à IMT Atlantique
Dépôt légal : Septembre 2017
ISSN : 2556-5060