Group project Machine Learning

Gabriel Alves
Salomé Benyamin
Cécile Bonnet
Ilona Karaboulkov

1. Analysis of the data and preprocessing

```
#Importing the file
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
import numpy as np

data = pd.read_csv('/content/drive/MyDrive/Churn.csv')
data2= data.drop(columns=['RowNumber', 'Surname','CustomerId'])
```

Firstly, we imported the file from our Google Drive. Loading the pandas library is essential to read a csv file. We also imported numpy a crucial package for our script.

We decided to exclude some columns that are the following: RowNumber, CustomerID and Surname. We used the drop function to remove these columns. We have chosen to remove them because they are not relevant to see if a customer will stay or not in the bank.

```
#Import the packages
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Model
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
```

Secondly, we imported the other packages that will be necessary to develop our model.

```
#Encoding
from sklearn.preprocessing import OrdinalEncoder
encoder = OrdinalEncoder()
value = encoder.fit_transform(data2['Geography'].values.reshape(-1, 1))
data2['Geography'] = value
encoder = OrdinalEncoder()
value = encoder.fit_transform(data2['Gender'].values.reshape(-1, 1))
data2['Gender'] = value
```

Thirdly, we encoded the geography and gender column so we can use them for later and as originally the values in these columns were stored in the string format which can not be

exploited directly. As the columns are not all in the same format, text and numbers, we encode them. This function is loaded in the sklearn.preprocessing package. The encoding enables to give a number value to a text. We used this for the Geography and Gender columns. Now we have 0 for France, 1 for Germany and 2 for Spain and for 1 for Male and 0 for Female.

```
#Get the target column
y= data2.iloc[:,len(data2.columns)-1]

#Splitting the Data into Train and Test Sets
data2_train, data2_test, y_train, y_test = train_test_split(data2.iloc[:,0:len(data2.columns)-1], y, test_size = 0.3, random_state = 42)

len(data2_train), len(data2_test), len(y_train), len(y_test)
```

Fourthly, it was necessary to get the target column which corresponds to the column of exited clients. We then splitted the data into train and test sets for our model.

```
#Normalization
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
data2_train = sc.fit_transform(data2_train)
data2_test = sc.transform(data2_test)
```

After, we need to do the normalization of the variables because they present different orders of magnitudes. The normalization then enables to have the same order for all the variables and a better plot.

2. Construction of the model

Then, we imported the tensorflow package because we will need it in order to develop the model thanks to keras.

We split the data into train and test sets. The first step is to define the target column in order to train the model. Here, our target column that we will predict, is the Exited column. We chose to have 25% tested and 75% of the model that is trained.

We model we created is based on the sequential Keras model. We have chosen to implement 64 layers thanks to the relu function. This function is an activation function an enables to change the way we see a data. The relu function gives x if x is superior to 0 or 0 otherwise. Then, the use of the sigmoid function is very important too because it is the one that will say if yes or no a client will churn. Indeed, this function gives a number that is 0 or 1. In the model, we have chosen to set the input_dim to 10. It corresponds to the number of colomns in the data2_train and data2_set.

3. Training of the model

We have decided to train the model for 20 epochs, which means we will make 20 iterations of the samples, and we will divide the data into training and test sets. We perform validation of 25% of the data and at the same time monitor the accuracy loss on the data we set apart. To run properly, the model.fit () command returns a history object, which is a dictionary that contains data about everything that happened during the training. This object will retrieve information on the two metrics, accuracy and loss, both on the validation and training processes.

4. Analyze the performance

We can see that the training loss decreases with every epoch and that the training accuracy increases.

Concerning the training loss, we have good results because it is the normal situation that it decreases. Indeed, by training the model many times, here 20 epochs, we have less and less losses. Both the training and the validation curves decrease.

On the contrary, the plotting of the accuracy shows an increasing curve. That is because when training and validating the model, it gets more and more accurate.

5. Evaluation on test data

Then, we train a new network from scratch only for 4 epochs and then evaluate it on the test data. Here are our final results:

```
[0.4075709581375122, 0.8296666741371155]
```

Ultimately, we generated the likelihood of clients leaving the bank by using the predict method:

```
model.predict(data2_test)
```

```
array([[0.1651065 ],
       [0.07807016],
       [0.31090605],
       ...,
       [0.08459404],
       [0.05536515],
       [0.1315397 ]], dtype=float32)
```