**Examen : Les bases de Machine Learning**

**Meggouri Ismail**

**Exercice 1 :**

1. Comment peut-on distinguer entre un problème de régression et un problème de classification ?

**Réponse :**

**La variable cible continue pour la régression et discrète pour la classification.**

1. Donner le modèle mathématique pour un modèle de régression linéaire pour un dataset ayant 3 inputs et un Output.

**Réponse :**

**Supposons que nos trois entrées soient notées x1,x2,x3 et que la sortie (output) soit noté y.**

**De plus nous aurons des coefficients associés a chaque entrée, notés w1,w2 et w3 ,ainsi qu’un biais**

**y=x1\*w1+x2\*w2+x3\*w3+bias**

1. Donner le schéma d’un réseau de neurones pour un modèle de régression (avec un seul neurone) pour un dataset ayant 3 inputs et un Output.

**Réponse :**

**W1**

**i=3**

**W2**

**W3**

1. Donner le schéma d’un réseau de neurones pour un modèle de classification pour un dataset ayant 3 inputs et un Output pour un problème linéairement séparable.

**Réponse :**

X1

W1

W2

W3

**ws**

**F(ws)**

**yhat**

X2

X3

bias

**ws= b+ x1\*w1+x2\*w2+x3\*w3 et F(x)=1/(+e^(-x))**

1. A quel moment il est nécessaire de standardiser les données ?

**Réponse :**

**Il est nécessaire de standardiser les données lorsque les différentes caractéristiques (features) du dataset sont mesurées dans des unités différentes ou ont des échelles différentes. La standardisation permet de mettre toutes les variables à la même échelle, ce qui facilite la convergence des algorithmes d'apprentissage automatique, en particulier ceux qui sont sensibles aux échelles, comme les réseaux de neurones.**

**Exercice 2 : Neural Network (one neurone) from scratch**

Dans cet exercice, nous allons utiliser le dataset data1.csv se trouvant dans googleclassromm/examen\_final

1. **Data**
   1. Écrire le code source Afficher le dataset (X,y) puis afficher les principales informations relatives au dataset ?

Votre code source :

import numpy as np

from sklearn.model\_selection import train\_test\_split

import pandas as pd

from google.colab import files

uploaded = files.upload()

df = pd.read\_csv("data1.csv")

df.head(10)

df.columns df.tail(10)

df[['y']]

* 1. Pourquoi il est nécessaire de standardiser les données ?

**Les données ne sont pas sur la même échelle, il est donc nécessaire de standardiser le dataset et uniquement la partie features (X).**

Votre code source pour standardiser les données :

data = np.array(df)

#extract the features :

X=np.array([data[i,:len(df.columns) - 1] for i in range(len(data))])

y=np.array([data[i,len(df.columns) - 1] for i in range(len(data))])

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X = scaler.fit\_transform(X)

* 1. À l’aide de train\_test\_split de sklearn.model\_selection, diviser le dataset en training dataset et test dataset (80% pour training\_data et 20% pour test\_data)

Votre code source pour split:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=44)

* 1. Pourquoi il faut séparer les données en données d’apprentissage et en données de test ?

**Réponse :**

**Pour évaluer le modèle sur des données non utilisées dans la phase d’apprentissage : généralisation**

* 1. Afficher le nombre d’enregistrements pour le training et le nombre d’enregistrement pour le test

Votre code source:

Len(X\_train) = 614

Len(X\_test) = 154

1. En explorant les données, est ce qu’il s’agit d’un problème de classification ou de régression ?

**Réponse :**

**En explorant ces données, il s’agit d’un problème de classification**

1. **Implémentation, Création, Entrainement, utilisation et évaluation du Modèle**

En se basant sur le programm en annexe, compléter le code source de la métode sigmoid et de la classe NeuralNetwork afin de mettre en place une application from scratch basé sur un seul neurone pour faire la classification. Et ce en tenant en compte la précision suivante :

* L’application doit être capable d’accepter n’importe quelle taille d’entrées

import numpy as np import matplotlib.pyplot as plt

import pandas as pd from sklearn.model\_selection

import train\_test\_split from google.colab

import files from sklearn import metrics

uploaded = files.upload()

df = pd.read\_csv("data1.csv")

data=np.array(df)

X=np.array([data[i,:len(df.columns)-1] for i in range(len(data))])

y=np.array([data[i,len(df.columns)-1] for i in range(len(data))])

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.2,random\_state=44)

class NeuralNetwork:

    def \_\_init\_\_(self):

        self.learning\_rate = 0.001

        self.num\_iterations = 500

        self.weights = None

        self.bias = None

    def sigmoid(self, z):

        return 1 / (1 + np.exp(-z))

    def fit(self, X, y):

        m, n = X.shape

        self.weights = np.zeros(n)

        self.bias = 0

        for i in range(self.num\_iterations):

            z = np.dot(X, self.weights) + self.bias

            predictions = self.sigmoid(z)

            dw = (1 / m) \* np.dot(X.T, (predictions - y))

            db = (1 / m) \* np.sum(predictions - y)

            self.weights -= dw \* self.learning\_rate

            self.bias -= db \* self.learning\_rate

    def predict(self, X):

        z = np.dot(X, self.weights) + self.bias

        predictions = self.sigmoid(z)

        return np.round(predictions)

confusion\_matrix = metrics.confusion\_matrix(y\_test, yhat)

cm\_display = metrics.ConfusionMatrixDisplay(

    confusion\_matrix=confusion\_matrix, display\_labels=[False, True]

)

cm\_display.plot()

1. Créer un schéma qui montre l’architecture du modèle

Votre schéma ici

1. Créer le modèle à partir de la classe créée

Votre code source :

model=NeuralNetwork()

1. Faire l’apprentissage

Votre code source :

model.fit(X\_train,y\_train)

1. Afficher les paramètres du modèle obtenus après apprentissage

Votre code source pour afficher les paramètres:

|  |
| --- |
| model.weights model.bias |

weights: array([ 0.08024334, 0.01299786, -0.02869338, 0.00377393, 0.00077223, -0.00397859, 0.04778198, -0.00512331])

bias: -0.35103285490085034

**Les valeurs des paramètres**

|  |  |  |
| --- | --- | --- |
| Itération 100 | Itération 1000 | Fin apprentissage |
| Weights: array([  1.13945299e-03, 6.44400573e,  -1.77127629e,2.55597330e-03,  1.27451545e-03, 3.61245935e,  1.89511661e-05, -1.14873002e-03])  bias: -0.00044852336923857666 | Weights: array([  0.01272191, 0.01152792,  -0.02934249, 0.00290592,  0.00081234, 0.00684456,  0.00038328, 0.00330223])  bias :  -0.0037914591820825687 | Weights : array([  0.08024334, 0.01299786,  -0.02869338, 0.00377393,  0.00077223, -0.00397859,  0.04778198, -0.00512331])  bias : -0.35103285490085034 |

1. Prédire l’output qui concerne le dataset de test

Votre code source :

yhat = model.predict(X\_test)

**Output :**

array(

    [

        0.0,1.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,

        0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0,0.0,1.0,

        0.0,0.0,1.0,1.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,

        0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,

        0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,

        0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0

    ]

)

1. **Evaluation de la performance du modèle**
2. Pourquoi il est indispensable d’évaluer la performance du modèle

**Réponse :**

**Nous permet de mesurer à quel point il est bon avant de l’utiliser pour des données jamais vues.**

1. Quelles sont les bonnes pratiques à suivre pour évaluer un modèle

**Réponse :**

**Séparation des données**

**Validation croisée (cross-validation)**

**Métriques de performance**

**Comparaison de modèles**

**Analyse des erreurs**

1. En utilisant les librairies qu’il faut, évaluer la performance du modèle

Votre code source :

#confusion matrix

y\_predicted=model.predict(X\_test)

confusion\_matrix = metrics.confusion\_matrix(y\_test,y\_predicted)

cm\_display = metrics.ConfusionMatrixDisplay(confusion\_matrix = confusion\_matrix, display\_labels = [False, True])

cm\_display.plot()

plt.show()

#accuracy

(45+19)/(45+19+82+8)= 0.42

#precision

19/(19+8)=0.7

#f1score

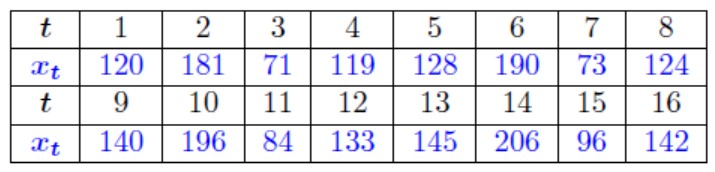
19/(19+82)=0.19

#recall

2\*precision\*recall/(precision+recall) = 2\*0.7\*0.19/(0.7+0.19)=0.3

**Exercice 3 : Time series analysis and Neural Network (one neurone) from scratch**

Le tableau ci-dessous donne le chiffre d'affaires d'une entreprise sur la période 1994 à 1997.



1. Définir l’objet data qui contient la série temporelle.

Votre code source :

import matplotlib.pyplot as plt

import tensorflow as tf

import numpy as np

data = {

    1: 120,

    2: 181,

    3: 71,

    4: 119,

    5: 128,

    6: 190,

    7: 73,

    8: 124,

    9: 140,

    10: 196,

    11: 84,

    12: 133,

    13: 145,

    14: 206,

    15: 96,

    16: 142,

}

1. Représentez cette série graphiquement.

Votre code source :

# Extraire les valeurs de t et X(t) pour le graphe

t\_values = list(data.keys())

X\_values = list(data.values())

# Créer le graphe

plt.figure(figsize=(10, 6))

plt.plot(t\_values, X\_values, marker='o', linestyle='-', color='b')

# Ajouter des titres et des étiquettes d'axe

plt.title("Série temporelle de chiffre d'affaires")

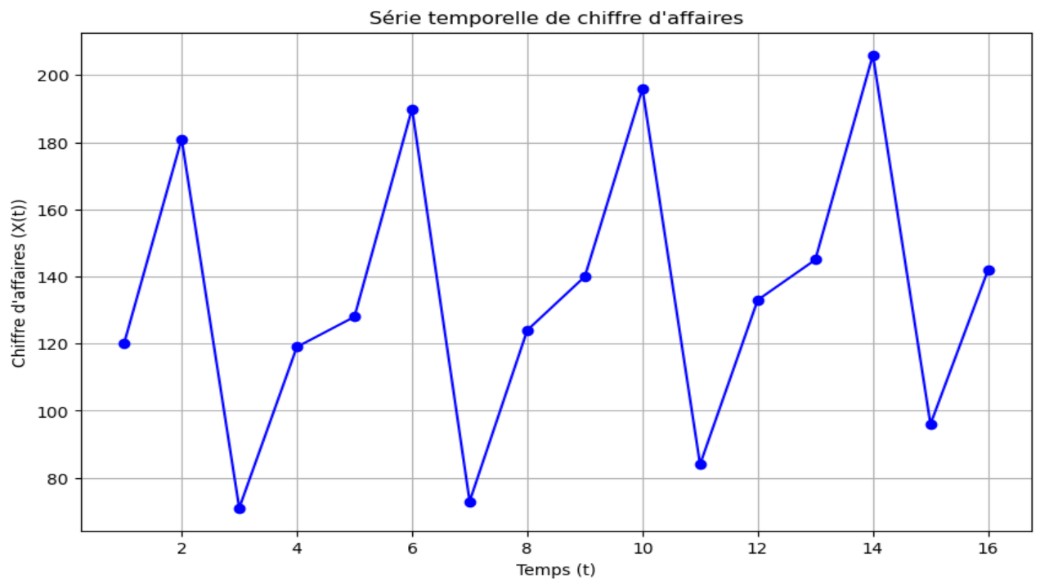
plt.xlabel('Temps (t)')

plt.ylabel("Chiffre d'affaires (X(t))")

# Afficher le graphe

plt.grid(True)

plt.show()



A-t-elle une composante saisonnière ? si oui, de quelle période ?

**Réponse :**

**Période 2.**

1. Quel est le type de la série ? pourquoi ?

**Réponse :**

**Réponse : série temporelle avec tendance et saisonnalité, car elle présente a la fois une tendance a la hausse et une variation périodique régulière**

1. On considère comme valeur initiale une moyenne des 3 premières valeurs observées et pour 𝛼 = 0.3 et 𝛼 = 0.7. Trouver une estimation par la méthode Lissage Exponentiel Simple (LES).

Votre code source :

def simple\_exponential\_smoothing(data, alpha, initial\_value):

    forecasts = []

    for t in sorted(data.keys()):

        if t == 1:

            forecast = initial\_value

        else:

            forecast = alpha \* data[t-1] + (1 - alpha) \* forecasts[-1]

            forecasts.append(forecast)

    return forecasts

# Coefficients de lissage

alpha\_1 = 0.3

alpha\_2 = 0.7

# Estimations avec alpha=0.3 et valeur initiale comme moyenne des 3 premières valeurs

estimations\_alpha\_1 = simple\_exponential\_smoothing(data, alpha\_1, (data[1] + data[2] + data[3]) / 3)

# Estimations avec alpha=0.7 et valeur initiale comme moyenne des 3 premières valeurs

estimations\_alpha\_2 = simple\_exponential\_smoothing(data, alpha\_2, (data[1] + data[2] + data[3]) / 3)

# Affichage des estimations

print("Estimations avec alpha = 0.3:", estimations\_alpha\_1)

print("Estimations avec alpha = 0.7:", estimations\_alpha\_2)

Output :

Estimations avec alpha = 0.3: [124.0, 122.8, 140.26, 119.48199999999999, 119.33739999999997,

                               121.93617999999998, 142.355326, 121.5487282, 122.28410973999999,

                               127.598876818, 148.1192137726, 128.88344964082, 130.118414748574,

                               134.5828903240018, 156.00802322680124, 138.00561625876088]

Estimations avec alpha = 0.7: [124.0, 121.2, 163.06, 98.618, 112.8854, 123.46562,

                            170.03968600000002, 102.11190580000002, 117.43357174,

                            133.230071522, 177.16902145659998, 111.95070643698,

                            126.68521193109399, 139.5055635793282, 186.05166907379845, 123.01550072213954]

1. Après l’évaluation entre les deux séries estimées, Afficher et représenter graphiquement la série LES qui retourne la meilleur estimation.

Votre code source :

def calculate\_mape(actual, forecast):

    actual = np.array(actual)

    forecast = np.array(forecast)

    return np.mean(np.abs((actual - forecast) / actual)) \* 100

mape\_alpha\_1 = calculate\_mape(list(data.values()), estimations\_alpha\_1)

mape\_alpha\_2 = calculate\_mape(list(data.values()), estimations\_alpha\_2)

# Choix de la meilleure série estimée en fonction du MAPE

best\_estimation = estimations\_alpha\_1 if mape\_alpha\_1 < mape\_alpha\_2 else estimations\_alpha\_2

# Affichage du MAPE pour chaque série estimée

print("MAPE avec alpha = 0.3:", mape\_alpha\_1)

print("MAPE avec alpha = 0.7:", mape\_alpha\_2)

Output :

MAPE avec alpha = 0.3: 31.89065425754557

MAPE avec alpha = 0.7: 44.22126060904764

1. Quel est le rôle de lissage expenetiel.

**Réponse :**

**Modélisation de tendances et de niveaux**

**Prévisions à court terme**

**Réduction du bruit et identification de tendances régulières**

1. Définir une fonction CoefSaison() de la série des coefficients saisonniers. Donner cette série.

Votre code source :

alpha = 0.3

initial\_value = (data[1] + data[2] + data[3]) / 3

# Estimation par la méthode LES pour obtenir la composante saisonnière

estimations = simple\_exponential\_smoothing(data, alpha, initial\_value)

seasonal\_component = np.array(list(data.values())) - np.array(estimations)

# Fonction pour calculer les coefficients saisonniers

def CoefSaison(data, seasonal\_component):

    n = len(data)

    period = len(seasonal\_component)

    coef\_saison = {}

    for i in range(1, period+1):

        coef\_saison[i] = np.mean(data[i:n:period] / seasonal\_component[:n//period])

    return coef\_saison

# Calcul des coefficients saisonniers

coefficients\_saisonniers = CoefSaison(list(data.values()), seasonal\_component)

# Affichage de la série des coefficients saisonniers

print("Coefficients saisonniers:")

print(coefficients\_saisonniers)

1. Ajuster la série **LES** par une droite via **un neurone (voir exercice2).**

Votre code source :

# Estimation par la méthode LES pour obtenir la série LES

estimations = simple\_exponential\_smoothing(data, alpha, initial\_value)

# Conversion des données en tableaux NumPy

X = np.array(list(range(1, len(data)+1)))

y = np.array(list(data.values()))

# Création du modèle de neurone (régression linéaire)

model = tf.keras.Sequential([tf.keras.layers.Dense(units=1, input\_shape=[1])])

# Compilation du modèle avec une fonction de perte (Mean Squared Error) et un optimiseur (Adam)

model.compile(loss='mean\_squared\_error', optimizer='adam')

# Entraînement du modèle sur les données

model.fit(X, y, epochs=1000, verbose=0)

# Obtenir le poids entraîné par le modèle

weight = model.get\_weights()[0][0]

bias = model.get\_weights()[1][0]

# Prédictions du modèle sur la série LES

predictions = model.predict(X)

# Affichage des résultats

print("(weight):", weight)

print("(bias):", bias)

# Représentation graphique de la série LES ajustée par une droite

plt.figure(figsize=(10, 6))

plt.plot(x, y, marker='o', linestyle= '-', color='b', label='Données réelles')

plt.plot(X, predictions, marker='o', linestyle='-', color='r', label='Prédictions (Droite ajustée)')

plt.title("Série temporelle de chiffre d'affaires et droite ajustée par un neurone")

plt.ylabel("Chiffre d'affaires (x(t))")

plt.xlabel('Temps (t)')

plt.legend()

plt.grid(True)

plt.show()

A graph with blue lines and red dots

Description automatically generated

(weight): [2.21848]

(bias): 0.9833726

1. Trouver la série de prédiction par la fonction predict().

Votre code source :

# Conversion des données en tableaux NumPy

X = np.array(list(range(1, len(data)+1)))

y = np.array(list(data.values()))

# Création du modèle de neurone (régression linéaire)

model = tf.keras.Sequential([tf.keras.layers.Dense(units=1, input\_shape=[1])])

# Compilation du modèle avec une fonction de perte (Mean Squared Error) et un optimiseur (Adam)

model.compile(loss='mean\_squared\_error', optimizer='adam')

# Entraînement du modèle sur les données

model.fit(X, y, epochs=1000, verbose=0)

# Prédiction de la série de prédiction

predictions = model.predict(X)

# Affichage des résultats

print("Série de prédiction:")

print(predictions.flatten())

Série de prédiction:

[2.8408275, 4.697875, 6.554923, 8.41197, 10.269018,

 12.126066, 13.983113, 15.840161, 17.697208, 19.554256, 21.411304, 23.268353

25.1254, 26.982447, 28.839495, 30.696543 ]

1. Donnez une prévision pour t=17 et 18 nombre.

Votre code source :

# Prévision pour t=17 et t=18

prediction\_t\_17 = model.predict(np.array([17]))[0][0]

prediction\_t\_18 = model.predict(np.array([18]))[0][0]

# Affichage des prévisions

print("Prévision pour t=17:", prediction\_t\_17)

print("Prévision pour t=18:", prediction\_t\_18)

Prévision pour t=17: -1.0971867

Prévision pour t=18: -1.2197093

**Annexe -program1**

**Annexe -program1–fit pour LinearRegression**

def fit(X,y):

n\_samples,n\_features=X.shape

learning\_rate=0.00001

n\_iterations=1000

bias=0

W=np.zeros(n\_features)#ligne 6

for \_ in range(n\_iterations):

y\_predicted=np.dot(X,weights)+bias#ligne 8

d\_W=(2/n\_samples)\*np.dot(X.T,(y\_predicted-y)) #ligne 9

d\_bias=(2/n\_samples)\*np.sum(y\_predicted-y) #ligne 10

W-=learning\_rate \*d\_W #ligne 11

bias-= learning\_rate \*d\_bias #ligne 12

return bias,W