**Examen : Les bases de Machine Learning**

**Exercice 1 :**

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

**Réponse :**

**Régression** : prédire des valeurs continues.

**Classification** : attribuer des catégories ou étiquettes discrètes.

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

**Réponse :**

Le modèle de régression linéaire peut être exprimé comme suit :

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

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 :**

**X1**

**X2 Y**

**X3**

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**

**Y**

**X2**

**X3**

**Couche sortie**

**Couche d’entrée**

**Couche caché**

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

**Repose :**

1. Donner une définition à un problème linéairement séparable ?

**Réponse :**

Un problème linéairement séparable est un problème où les données appartenant à différentes classes peuvent être séparées par une frontière de décision linéaire, c'est-à-dire une ligne droite ou un plan dans l'espace des caractéristiques

1. Pourquoi les arbres de décisions se sont considérés des modèles boite blanche au contraire des réseaux de neurones ?

**Réponse :**

Les arbres de décisions sont des modèles boîte blanche car leur processus de prise de décision est facilement interprétable, tandis que les réseaux de neurones sont des modèles boîte noire car leur processus de prise de décision est complexe et difficile à comprendre.

1. Pourquoi naive bayes est considéré comme étatant un classifieur naif?

**Réponse :**

Naive Bayes est considéré comme "naif" car il fait l'hypothèse que toutes les caractéristiques sont indépendantes les unes des autres, ce qui peut être une simplification excessive dans la réalité.

1. Pour un problème de classification, dans quels cas préférer travailler avec :

**Réponse :**

* **Naive bayer** : Préférez Naive Bayes lorsque les caractéristiques sont indépendantes ou que l'hypothèse de naïveté peut être raisonnablement appliquée.
* **Arbres de décision :** Préférez les Arbres de décision lorsque vous avez besoin d'interopérabilité et de compréhension des règles de décision**.**
* **Réseau de neurones :** Préférez les Réseaux de neurones pour des problèmes complexes et non linéairement séparables.
* **Régression logistiques :** Préférez la Régression logistique pour des problèmes de classification binaire simples.

1. Pour un problème de classification, dans quels cas utiliser les métriques suivantes:

**Réponse :**

* **Confusion matrix :** Utilisez la confusion matrix pour obtenir une vue globale des performances du modèle de classification.
* **Accuracy :** Utilisez l'accuracy lorsque vous voulez mesurer la proportion totale de prédictions correctes du modèle par rapport à l'ensemble des prédictions.
* **Precision :** Utilisez la precision lorsque vous voulez vous concentrer sur la qualité des prédictions positives du modèle
* **Recall :** Le rappel mesure la proportion de vrais positifs parmi tous les exemples réels positifs présents dans les données

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

Votre code source :

import pandas as pd

df= pd.read\_csv("data1.csv”)

df.head()

df.info()

1. Pourquoi il est nécessaire de standardiser les données ?

Reponse :

La standardisation des données met toutes les caractéristiques à la même échelle, améliorant ainsi la performance des algorithmes d'apprentissage automatique et évitant les problèmes liés aux écarts d'échelle entre les variables.

Votre code source pour standardiser les données:

import pandas as pd

# Lire le fichier CSV

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

# Séparer les caractéristiques (X) de la variable cible (y)

X = df.drop(columns=["y"])

# Calculer la moyenne de chaque colonne

means = X.mean()

# Calculer l'écart type de chaque colonne

stds = X.std()

# Standardisation manuelle des caractéristiques (X)

X\_standardized = (X - means) / stds

# Remplacer les données originales par les données standardisées

df[X.columns] = X\_standardized

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:

import pandas as pd

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

# Lire le fichier CSV

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

# Séparer les caractéristiques (X) de la variable cible (y)

X = df.drop(columns=["y"])

y = df["y"]

# Diviser le dataset en training dataset (80%) et test dataset (20%)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

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

**Réponse :**

Séparer les données en apprentissage et en test permet d'évaluer la capacité du modèle à généraliser sur de nouvelles données et à éviter le surapprentissage.

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

Votre code source:

# Afficher le nombre d'enregistrements pour le training dataset et le test dataset

print("Nombre d'enregistrements dans le training dataset:", X\_train.shape[0])

print("Nombre d'enregistrements dans le test dataset:", X\_test.shape[0])

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

**Réponse :**

Il sagit d’un probleme de classification car le variable y est une variable discréte .

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

**Def sigmoid(Z) :**

return 1 / (1 + np.exp(-Z))

**class NeuralNetwork:**

def \_\_init\_\_(self):

#hyperparameters

self.learning\_rate=0.001

self.n\_iterations=5000

def fit(self,X,y):

n\_samples, n\_features = X.shape

self.weights = np.random.randn(n\_features)

self.bias = np.random.randn()

# Entraînement du modèle en utilisant la descente de gradient

for \_ in range(self.n\_iterations):

# Calculer la prédiction du modèle (fonction d'activation sigmoid)

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

A = sigmoid(Z)

# Calculer les gradients

dZ = A - y

dW = np.dot(X.T, dZ) / n\_samples

db = np.sum(dZ) / n\_samples

# Mettre à jour les poids et le biais

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

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

return self.weights, self.bias

def predict\_proba(self,X):

# Calculer la probabilité de chaque échantillon d'appartenir à la classe positive (1)

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

A = sigmoid(Z)

return A

def predict(self,X):

proba = self.predict\_proba(X)

predictions = (proba >= 0.5).astype(int) # seuil de 0.5 pour la classification

return predictions

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

**Caractéristiques d'entrée (X1, X2, ..., Xn)**

**W**

**(Poids w1, w2, ... )**

**│ ..., wn)**

**Z**

**(Combinaison linéaire)**

**Fonction d'activation (Sigmoid)**

**Probabilité de classification (0 ≤ P ≤ 1)**

**Sortie de classification**

**(0 ou 1)**

1. Créer le modèle à partir de la classe créée

Votre code source :

import numpy as np

X = df.drop(columns=["y"])

y = df["y"]

# Instancier un objet du type NeuralNetwork

model = NeuralNetwork()

# Entraîner le modèle en utilisant la méthode fit

model.fit(X, y)

# Exemples de nouvelles données pour effectuer des prédictions

nouvelles\_donnees = np.array([[13, 14, 15],

[16, 17, 18]])

# Effectuer des prédictions sur les nouvelles données

predictions = model.predict(nouvelles\_donnees)

# Afficher les prédictions

print(predictions)

1. Faire l’apprentissage

Votre code source :

import numpy as np

X\_train X\_train

y\_train y\_train

# Instancier un objet de la classe NeuralNetwork

model = NeuralNetwork()

# Entraîner le modèle en utilisant la méthode fit avec vos données d'entraînement

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

predictions = model.predict(X\_test)

# Afficher les prédictions

print(predictions)

1. Afficher les paramètres du modèle obtenus après apprentissage

Votre code source pour afficher les paramètres:

model = NeuralNetwork()

# Entraîner le modèle en utilisant la méthode fit avec vos données d'entraînement

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

# Afficher les paramètres du modèle obtenus après l'apprentissage

print("Poids (Weights) du modèle :")

print(model.weights)

print("\nBiais (Bias) du modèle :")

print(model.bias)

**Les valeurs des paramètres**

|  |  |  |
| --- | --- | --- |
| Itération 100 | Itération 1000 | Fin apprentissage |
|  |  |  |

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

Votre code source :

model = NeuralNetwork()

# Entraîner le modèle en utilisant la méthode fit avec vos données d'entraînement

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

# Prédire l'output sur le dataset de test

predictions = model.predict(X\_test)

# Afficher les prédictions

print(predictions)

**Output :**

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

**Réponse :**

L'évaluation de la performance du modèle est indispensable pour mesurer son efficacité, comparer différentes approches, ajuster les hyperparamètres, détecter le surajustement, prendre des décisions éclairées et améliorer continuellement le modèle.

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

**Réponse :**

Pour évaluer un modèle efficacement : divisez les données en ensembles d'entraînement et de test, utilisez la validation croisée, choisissez des mesures de performance appropriées, analysez les courbes d'apprentissage et de validation, et comparez avec des modèles de référence.

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

Votre code source :

#confusion matrix

#accuracy

#precision

#f1score

#recall

**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

**Exercice 3 : Neural Network à l’aide keras-tensoflow**

Dans cet exercice, nous allons reprendre le même problème de l’exercice 2, donc reprendre les mêmes données pour l’entrainement et pour le test.

1. En se basant sur les librairies nécessaires de keras-tensorflow, écrire le code source qu’il faut pour :
2. Créer le modèle

Votre code source :

1. Entrainer le modèle

Votre code source :

1. Évaluer le modèle

Votre code source :

1. Comparer les résultats obtenus avec l’exercice 2

**Exercice 4 : Les arbres de décision de sklearn**

Dans cet exercice, nous allons reprendre le même problème de l’exercice 2, donc reprendre les mêmes données pour l’entrainement et pour le test.

1. En se basant sur les librairies nécessaires sklearn, écrire le code source qu’il faut pour :
2. Créer le modèle

Votre code source :

1. Entrainer le modèle

Votre code source :

1. Évaluer le modèle

Votre code source :

1. Comparer les résultats obtenus avec l’exercice 2 et 3