Rapport de Projet

Julien OHANA

Machine Learning

Livré le 13/04/2024

Table des matières

1 Introduction

1.1 Objectif du projet

L'objectif principal de ce projet consiste à élaborer un réseau de neurones capable d'effectuer une classification précise en déterminant si un individu a éprouvé un sentiment de satisfaction à l'égard de son expérience de vol (affirmatif ou négatif), en se fondant sur une gamme variée de paramètres. Pour mener à bien ce projet, je vais débuter avec un ensemble de données d'entraînement et de test préalablement constitué et prêt à l'emploi.

1.2 Environnement de travail

Pour la réalisation de ce projet, j'ai choisi d'adopter Anaconda, une plateforme open-source reconnue pour sa capacité à simplifier la gestion et le déploiement d'environnements de développement dédiés au langage de programmation Python. Cette décision s'avère stratégique dans la mesure où elle permettra une installation fluide et harmonieuse des bibliothèques essentielles au bon déroulement du projet. En optant pour Anaconda, je m'assure également de bénéficier d'un écosystème robuste et largement éprouvé, offrant ainsi un cadre de travail stable et efficient pour la mise en œuvre du réseau de neurones.

2 Analyse et manipulation des données

2.1 Import des librairies

Tout d'abord je vais importer les différentes librairies nécessaires au projet.

Import des librairies

Entrée [197]:
```
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dense, Dropout
from sklearn.preprocessing import LabelEncoder
from keras.callbacks import EarlyStopping
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
from tensorflow.keras.optimizers import Adam,SGD
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import os
```

On retrouvera dans le lot :

- Pandas qui va nous servir pour extraire les datasets du excel et faire des modifications dessus.
- Tensorflow et Scikit-Learn pour la construction de réseau de neurone.

2.2 Import des données

Nous avons en notre possession un dataset de test et un dataset d'entraînement sous format excel. La première étape sera donc d'extraire ces données pour venir les stocker dans des variables que l'on va pourvoir manipuler dans l'environnement de travail (jupyter notebook). On affichera l'en tête d'un des 2 datasets pour avoir une idée de quoi est composé le jeu de données.

1 - Analyse et manipulation des données

Dans cette partie on va s'attarder à importer nos données et à venir les manipuler pour qu'on puisse les comprendre

Entrée [173]:
```
test_set = pd.read_csv(os.getcwd()+"/test.csv",usecols=lambda column: column != 'Unnamed: 0') # jeu de données de test
training_set = pd.read_csv(os.getcwd()+"/train.csv",usecols=lambda column: column != 'Unnamed: 0') # jeu de donnéees de train
training_set.head() # afficher l'en-tête des données pour comprendre à quoi elle ressemble
```

Out[173]:

	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/ Arrival time convenient	Ease of Online booking	...	Inflight entertainment	On-board service	Leg room service	Baggage handling	Checkin service	Inflight service
0	70172	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	4	3	...	5	4	3	4	4	5
1	5047	Male	disloyal Customer	25	Business travel	Business	235	3	2	3	...	1	1	5	3	1	4
2	110028	Female	Loyal Customer	26	Business travel	Business	1142	2	2	2	...	5	4	3	4	4	4
3	24026	Female	Loyal Customer	25	Business travel	Business	562	2	5	5	...	2	2	5	3	1	4
4	119299	Male	Loyal Customer	61	Business travel	Business	214	3	3	3	...	3	3	4	4	3	3

5 rows × 24 columns

2.3 Nettoyage des données inutiles

L'objectif est de venir supprimer les données qui ne seront pas interprétables par le modèle que nous allons mettre en place par la suite. Les données avec des valeurs à "null" ne seront pas retenues car non interprétables par notre futur modèle. Aussi on va venir supprimer la colonne id du dataset qui n'est en aucun cas corrélé avec la satisfaction des clients suite à un vol.

Nettoyage des données inutiles

On va reprérer les données qui sont à null pour les supprimer du dataset

Entrée [217]:
```
print(training_set.shape)
training_set.dropna(axis=0, inplace=True)
test_set.dropna(axis=0,inplace=True)
print(training_set.shape)
training_set.drop("id",axis=1,inplace=True)
test_set.drop("id",axis=1,inplace=True)
```

```
(103904, 24)
(103594, 24)
```

2.4 Repérage de la colonne qui nous intéresse

Ensuite en observant le dataset on comprends que la colonne des étiquettes correspond à la colonne "satisfaction". On va donc venir afficher quels sont les 2

possibilités de cette colonne.

Répérage de la colonne qui nous intéresse

La colonne qui va nous intéresser est la toute dernière du jeu de données. C'est elle qui va nous permettre de classer de façon binaire.

Entrée [218]:
```
val_unique = training_set['satisfaction'].unique()
val_formates = ' et '.join(['"{}"'.format(valeur) for valeur in val_unique])
print("Les valeurs uniques de la colonne 'satisfaction' sont : ", val_formates)
```

```
Les valeurs uniques de la colonne 'satisfaction' sont :  "neutral or dissatisfied" et "satisfied"
```

2.5 Encodage des données

Les réseaux de neurones sont conçus pour apprendre à partir de données numériques car cela offre une efficacité de traitement et une compatibilité avec les opérations mathématiques. De ce fait, on comprends donc que l'on va devoir faire des modifications sur le dataset. Dans un premier temps on va donc repérer quels sont les colonnes à modifier sous format numérique.

Encodage des données

On remarque que certaines données sont au format non numérique (object). Pour que notre modèle soit en capacité de comprendre ces valeurs on va devoir les transformer au format numérique. Ainsi on admet que chaque caractéristiques (features) du dataset initial sont importantes.

Entrée [219]:
```
training_set.dtypes # liste des champs avec leur format
```

Out[219]:
```
Gender                                object
Customer Type                         object
Age                                    int64
Type of Travel                        object
Class                                 object
Flight Distance                        int64
Inflight wifi service                  int64
Departure/Arrival time convenient      int64
Ease of Online booking                 int64
Gate location                          int64
Food and drink                         int64
Online boarding                        int64
Seat comfort                           int64
Inflight entertainment                 int64
On-board service                       int64
Leg room service                       int64
Baggage handling                       int64
Checkin service                        int64
Inflight service                       int64
Cleanliness                            int64
Departure Delay in Minutes             int64
Arrival Delay in Minutes             float64
satisfaction                          object
dtype: object
```

Par la suite on va mettre en place un encodeur qui va associer à une chaîne de caractère un format numérique. Il ne nous reste plus qu'à relever les colonnes en s'appuyant sur le résultat précédant et à appliquer l'encodeur pour chaque colonne (autant pour le jeu d'entraînement que le jeu de test).

Entrée [220]:
```
label_encoder = LabelEncoder() # on instancie un encoder qui va associer à une chaîne de caractère un format numérique
colonnes_a_encoder = ['Gender', 'Customer Type', 'Type of Travel','Class','satisfaction']
for col in colonnes_a_encoder:
    training_set[col] = label_encoder.fit_transform(training_set[col])
    test_set[col] = label_encoder.fit_transform(test_set[col])
training_set.head()
```

Out[220]:

	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/ Arrival time convenient	Ease of Online booking	Gate location	...	Inflight entertainment	On-board service	Leg room service	Baggage handling	Checkin service	Inflight service	Cl
0	1	0	13	1	2	460	3	4	3	1	...	5	4	3	4	4	5	
1	1	1	25	0	0	235	3	2	3	3	...	1	1	5	3	1	4	
2	0	0	26	0	0	1142	2	2	2	2	...	5	4	3	4	4	4	
3	0	0	25	0	0	562	2	5	5	5	...	2	2	5	3	1	4	
4	1	0	61	0	0	214	3	3	3	3	...	3	3	4	4	3	3	

5 rows × 23 columns

Dorénavant on a : neutral or dissatisfied = 0 et satisfied = 1

2.6 Distinction entre les étiquettes et les features

La dernière étape avant de passer au réseau de neurone est de dissocier les étiquettes du restant des caractéristiques. On va donc faire ça sous la forme d'un antécédent (x) et d'une image (y). On comprends donc le nom associé à nos variables.

Distinction entre les étiquettes et les features

Entrée [221]:
```
x_train, y_train = training_set.iloc[:, :-1],training_set['satisfaction']
x_test, y_test = test_set.iloc[:, :-1], test_set['satisfaction']
x_train.head()
```

Out[221]:

	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/ Arrival time convenient	Ease of Online booking	Gate location	...	Seat comfort	Inflight entertainment	On-board service	Leg room service	Baggage handling	Checkin service	In se
0	1	0	13	1	2	460	3	4	3	1	...	5	5	4	3	4	4	
1	1	1	25	0	0	235	3	2	3	3	...	1	1	1	5	3	1	
2	0	0	26	0	0	1142	2	2	2	2	...	5	5	4	3	4	4	
3	0	0	25	0	0	562	2	5	5	5	...	2	2	2	5	3	1	
4	1	0	61	0	0	214	3	3	3	3	...	5	3	3	4	4	3	

5 rows × 22 columns

3 Construction d'un réseau de neurone

3.1 Construction du modèle et phase d'entraînement

Pour construire notre réseau de neurone, on rappel que le Machine Learning est une science expérimentale ce qui signifie que lorsque l'on va vouloir mettre en place une architecture de réseau de neurone profond, il n'y a pas une seule et unique bonne réponse et qu'il va falloir faire des essais por trouver ce que l'on cherche. Pour ce modèle j'ai donc mis en place 4 couches :

- Une première couche avec 256 neurones qui utiliseront la fonction d'activation ReLu. Les fonctions d'activations vont nous permettre d'introduire de la non linéarité dans le modèle.
- Une deuxième couche avec 156 neurones qui utiliseront la fonction d'activation de la tangente hyperbolique.
- Une troisème couche avec 56 neurones qui utiliseront la fonction d'activation ReLu.
- Une quatrième et dernière couche avec 1 seul neurone qui fera usage de la fonction d'activation sigmoid.

Chaque couche est suivie d'un Dropout qui permet d'éviter le surapprentissage (overfitting) sur nos données d'entraînement.

Ensuite on va compiler notre modèle en venant lui ajouter un algorithme d'optimisation. Dans ce cas, j'ai choisi d'utiliser Adam qui combine les avantages de la descente de gradient stochastique (SGD) et de l'adaptation du taux d'apprentissage (Adagrad). Par défaut, le learning rate de l'algorithme sera égale à 0,001. Ensuite on va utiliser la fonction de perte nommée BCE (binary cross-entropy) qui est adapté dans le cas d'une classification binaire. Enfin on détermine l'usage de la métrique accuracy lors de l'entraînement.

On passe enfin à la phase de test et cette fois on donne au modèle nos données d'entraînement (x et y) et on peut lui fournir d'autres informations comme le nombre d'époque et le lot de données qu'on lui fournit par époque. À savoir qu'une époque correspond à un passage complet du lot de donnée d'entraînement à travers le modèle. Pendant une époque, le modèle voit chaque exemple d'entraînement une fois et ajuste ses poids en conséquence pour minimiser la perte.

2 - Construction d'un modèle de ML

Construction du modèle et phase d'entraînement

Dorénavant on va chercher à mettre en place notre réseau de neurone. Nous sommes dans le cas d'une problématique de classification donc on pourra utiliser une fonction de perte comme la BCE.

Entrée [222]:
```
# Réseau de neurones => le ML est une science expérimentale, on fait donc des essais à taton afin de trouver le bon modèle
# ou bien on fait usage d'une boucle while
model = Sequential([
    Dense(256, activation='relu', input_shape=(x_train.shape[1],)),
    Dropout(0.3),  # Ajout de Dropout pour éviter le surajustement/overfitting
    Dense(156, activation='tanh'),
    Dropout(0.2),
    Dense(56, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

# Algorithme d'optimisation => Adam
# fonction de perte => BCE
model.compile(optimizer='Adam', loss='binary_crossentropy', metrics=['accuracy'])

# Phase d'entrainement
history = model.fit(x_train, y_train,batch_size =1000, epochs = 50)
```

```
Epoch 1/50
104/104 [==============================] - 1s 6ms/step - loss: 0.7102 - accuracy: 0.5451
Epoch 2/50
104/104 [==============================] - 1s 6ms/step - loss: 0.6832 - accuracy: 0.5638
Epoch 3/50
104/104 [==============================] - 1s 6ms/step - loss: 0.6793 - accuracy: 0.5686
```

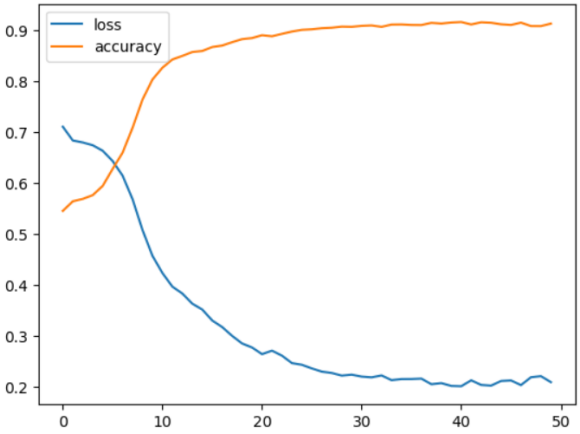
Au départ on constatera que la perte (loss) est basse mais pas assez et on se rends compte que notre précision est de 54% donc que le modèle n'est au départ des itérations pas très performant. Par la suite, on remarque que notre loss se rapproche de 0 et que notre précision (accuracy) a bien augmenté et qu'en fin d'apprentissage le modèle est capable d'avoir 90% de précision ce qui est nettement mieux.

```
Epoch 47/50
104/104 [==============================] - 1s 7ms/step - loss: 0.2033 - accuracy: 0.9144
Epoch 48/50
104/104 [==============================] - 1s 6ms/step - loss: 0.2187 - accuracy: 0.9077
Epoch 49/50
104/104 [==============================] - 1s 6ms/step - loss: 0.2208 - accuracy: 0.9076
Epoch 50/50
104/104 [==============================] - 1s 6ms/step - loss: 0.2091 - accuracy: 0.9125
```

Une courbe de l'évolution de la loss ainsi que de l'accuracy rendra la chose plus parlante pour mieux comprendre comment les 2 phénomènes sont liés.

Affichage de l'évolution de la loss et de l'accuracy au fil des epochs

```
Entrée [223]: pd.DataFrame(history.history).plot()
Out[223]: <AxesSubplot:>
```



3.2 Phase de test

Enfin après avoir entrainé notre modèle on passe à la phase de test.

Phase de test

Une fois la phase d'entraînement terminé on va vérifier ce que vaut notre modèle avec des nouvelles données

```
Entrée [224]: y_predictions=(model.predict(x_test) > 0.5).astype(int)
              810/810 [==============================] - 1s 999us/step
```

Ce qu'il faut comprendre de cette ligne de code c'est que grâce à la fonction predict() le modèle va pour chaque données de x_test donner une probabilité entre 0 et 1 pour donner une estimation selon ce qu'il a compris de si la personne est heureuse ou non. L'instruction >0.5 va agir comme un boolean pour trancher entre heureux (1) ou non (0) et ensuite on va tout logiquement venir comparer ça avec les étiquettes de test qui représente le résultat correct.

3.3 Évaluation de notre modèle à travers différentes métriques

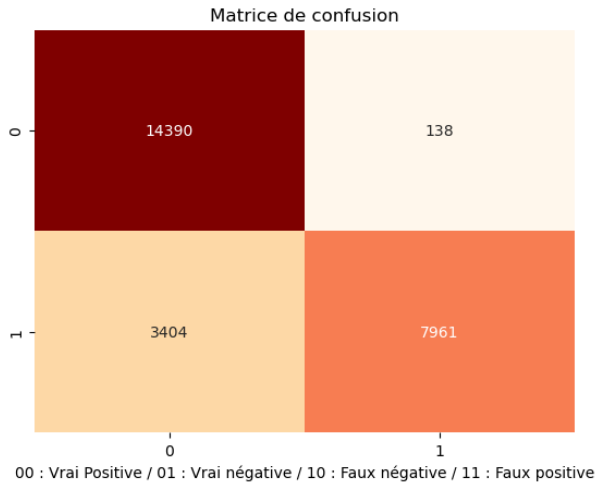
Ici on va donc mettre en place des métriques pour savoir si les prédictions du modèle sont bonnes ou non. J'ai choisis d'en utiliser 2 :

— La matrice de confusion
— la précision (l'accuracy)

Évaluation de notre modèle à travers différentes métriques

Il existe plusieurs métriques pour évaluer un modèle. Ici je choisis d'utiliser la matrice de confusion et l'accuracy 1er métrique : Matrice de confusion

Entrée [225]:
```
sns.heatmap(confusion_matrix(y_test, y_predictions), annot=True, fmt='g', cbar=False, cmap='OrRd')
plt.title("Matrice de confusion")
plt.xlabel('00 : Vrai Positive / 01 : Vrai négative / 10 : Faux négative / 11 : Faux positive')
plt.show()
```



00 : Vrai Positive / 01 : Vrai négative / 10 : Faux négative / 11 : Faux positive

2e métrique : Accuracy

Entrée [226]:
```
print('Accuracy Score: ', '{:.2f}%'.format(accuracy_score(y_test, y_predictions)*100))
```

```
Accuracy Score:  86.32%
```

Globalement on constate que notre modèle est plutôt performant et a réussi à passer les tests avec 86% de réussite avec la métrique de scoring accuracy.

4 Conclusion

En conclusion, dans ce projet de construction d'un réseau de neurones pour la classification binaire (BCE), nous avons exploré et mis en œuvre un modèle d'apprentissage automatique capable de distinguer efficacement entre deux classes. En utilisant des techniques telles que la sélection des caractéristiques et l'optimisation des hyperparamètres, nous avons pu créer un modèle robuste et précis. Ce processus de construction du réseau de neurones nous a permis de comprendre les principes fondamentaux du machine learning, ainsi que les défis et les stratégies associés à la création de modèles performants.

5 Annexe

Lien du repertoire Github : https ://github.com/JulienFX/ANNschool