TP 8 - Introduction à l'Intelligence Artificielle

I Banque de données sur les réseaux sociaux

On s'intéresse dans ce premier exemple à un ensemble de données contenant des profils d'utilisateurs d'un réseau social, qui, en interagissant avec une publicité que un produit, ont acheté ce produit ou non.

Le dataset étudié enregistré au format csv comporte les informations suivantes des utilisateurs : Leur numéro d'identifiant, leur sexe ("F" ou "M"), leur âge, leur salaire annuel et l'indication d'achat du produit (0 pour non ou 1 pour oui). Les fichier étant au format csv, on rappelle que chaque ligne est une chaîne de caractères, dans laquelle chaque donnée (obervations ou étiquettes) est séparée par une virgule. La première ligne contient les intitulés de chaque variable, à savoir ici : "Id,Sexe,Age,Salaire,Achat".

I.1 Préparation des données

Afin d'extraire les données sous forme de tableaux numpy, nous allons utiliser une bibliothèque très pratique pour manipuler les fichiers csv ; il s'agit de pandas. Nous allons l'importer de la façon suivante : `import pandas as pd`.

Pour ouvrir le dichier csv, il suffit alors de la stocker dans un on objet de type *Dataframe* de saisir par exemple : `dataset = pd.read_csv("NomFichiercsv.csv")`. Dans ce cas le fichier NomFichiercsv.csv doit être dans le même dossier que le script python.

On pourra afficher dans la console le contenu du dataset, en saisissant `print(dataset)` ou `dataset` tout simplement. On doit obtenir :

```
>>>dataset

            Id Sexe  Age  Salaire  Achat
0     15624510    M   19     19.0      0
1     15810944    M   35     20.0      0
2     15668575    F   26     43.0      0
3     15603246    F   27     57.0      0
4     15804002    M   19     76.0      0
..         ...  ...  ...      ...    ...
395   15691863    F   46     41.0      1
396   15706071    M   51     23.0      1
397   15654296    F   50     20.0      1
398   15755018    M   36     33.0      0
399   15594041    F   49     36.0      1

[400 rows x 5 columns]
```

Pour extraire ces données sous forme se tableaux numpy, nous allons utiliser la méthode `.to_numpy`. Ainsi, pour créer une matrice de taille (N,d) où N et d sont respectivement le nombre d'échantillons et le nombre de variables à étudier :

`X = dataset[["NomVariable1", "NomVariable2"]].to_numpy(dtype=float)` créera une matrice de taille (N,d). Une particularité remarquable est que les chaînes de caractères écrites avec des nombre sont automatiquement converties en entiers ou flottants. Comme ici, nous aurons besoin de flottants, on le précise avec `dtype=float`.

Dans notre cas, la colonne "Sexe" sera convertie, pour chaque ligne avec la chaine de caractères "F" ou "M". Il est possible de faire une boucle avec test pour substituer un "F" par 0 et un "M" par 1.

(généralisation possible si ce n'est pas binaire). Sinon, une alternative est d'utiliser les deux lignes de codes suivantes :

```
y = pd.Categorical(dataset["Sexe"]).astype('category').codes
y = y.reshape((y.shape[0], 1)) #Pour mettre au format matrice de taille (400,1)
```

Dans une première étude nous souhaitons prédire le comportement des utilisateurs de ce réseau social pour l'achat su produit (une voiture ici), en fonction de leur salaire annuel et de leur âge.

On doit dont extraire du dataset une matrice des observations (ou entrées) de taille $(N, 2)$, notée X_t et un vecteur de N étiquettes (sorties) y_t.

Q1. A partir des indications fournies précédemment, extraire du fichier *Reseaux_sociaux_auto.csv* une matrice numpy `Xt` de données d'entrées avec les deux variables "Salaire" et "Age". Faire de même avec le vecteur de sortie `yt`, issu de la donnée "Achat".

Q2. Déterminer si l'on est en présence d'un problème de classification ou de régression.

Nous allons maintenant représenter graphiquement les données sous forme de nuage de points dans un plan (Salaire,Age). Si l'utilisateur a effectivement acheté le produit, il sera représenté en bleu, sinon en rouge. Cela nous permettra de valider si les deux variables sont bien discriminantes vis-à-vis de l'achat du produit étudié ici. On importera la bibliothèque pyplot du module matplotlib : `import matplotlib.pyplot as plt`, et on utilisera la fonction `scatter`. pour les couleurs on utilisera aussi `from matplotlib.colors import *` On pourra alors utiliser les lignes de code suivante, précisément écrite pour notre exemple d'étude :

```
fig, ax = plt.subplots()
scatter=ax.scatter(Xt[:,0], Xt[:,1], c=yt, cmap = ListedColormap(['r','b']))
```

Q3. Analyser ces quelques lignes de code et les compléter code de façon à renseigner correctement les légendes sur les axes. Le salaire figurera en abscisses et sera converti en kilo-euros.

Avant de mettre en oeuvre l'algorithme de régression logistique, il peut s'avérer nécessaire, en machine learning de normaliser les différentes données. Cela est particulièrement le cas dans les problèmes de classification... Ainsi, les données d'entrées seront standardisées selon une loi centrée réduite. On rappelle qu'il est possible d'utiliser de fonctions de la bibliothèque `numpy` ; à savoir : `np.mean(x)` calculera la moyenne de la matrice ou du vecteur `x`, et `np.std(x)` calculera l'écart type de la matrice ou du vecteur `x`.

Q4. Stocker dans deux listes `mu` et `sigma` les moyennes et écarts types des données sur le salaire et l'âge. Établir alors une nouvelle matrice Xt dans laquelle on retrouve les valeurs centrées réduites pour les deux variables.

Nous sommes désormais prêts à utiliser les données pour pouvoir les classifier.

I.2 Classification avec Régression Logistique

Afin de pouvoir effectuer l'apprentissage sur les données étudiées ici, nous allons devoir programmer trois fonctions intermédiaires :

— Fonction modèle `h` prenant en arguments la matrice `Xt` et le vecteur de paramètres `W` et qui retourne la valeur : $S(Xt.W)$, où S est la fonction Sigmoïdale.

— Fonction coût `J` prenant en arguments la matrice des entrées, le vecteur des sorties et le vecteur des paramètres et calculant le coût (ou l'erreur) de la fonction modèle

— Fonction Gradient, prenant les mêmes arguments que la fonction précédente et qui sera utilisée dans l'algorithme de descente de Gradient.

Q5. Coder ces trois fonctions, ainsi qu'une fonction globale `Regression_Logistique` prenant en arguments la matrice X_t des observations, le vecteur y_t des étiquettes, le nombre d'itérations maximales, le taux d'apprentissage, et qui retourne le vecteur W des paramètres optimisant le coût, ainsi que l'historique de ce coût à chaque passage de boucle sous forme de liste. On prendra garde à ne pas oublier dans la fonction `Regression_Logistique` l'ajout d'une colonne de 1 dans la matrice X_t pour prendre en compte le biais dans le vecteur W.

Q6. Tester l'apprentissage sur le dataset, puis tracer l'évolution du coût en fonction du nombre d'itérations. Affiner si nécessaire le taux d'apprentissage et le nombre d'itérations pour que le coût converge vers une valeur asymptotique.

Nous allons maintenant analyser les résultats et visualiser graphiquement les résultats de l'apprentissage automatique. Pour cela, une méthode consiste à effectuer un maillage du plan avec une résolution de P points. Les axes du plan seront délimités grâce aux valeurs minimales des variables **normalisées** :

```
x1lim = (min(Xt[:,0])-0.5,max(Xt[:,0])+0.5)
x2lim = (min(Xt[:,1])-0.5,max(Xt[:,1])+0.5)
```

Le maillage est ensuite généré avec la fonction `meshgrid` du module `numpy`.

```
# meshgrid
x0 = np.linspace(x0lim[0], x0lim[1], resolution)
x1 = np.linspace(x1lim[0], x1lim[1], resolution)
X0, X1 = np.meshgrid(x0, x1)
# assembler les 2 variables
XX = np.vstack((X0.ravel(), X1.ravel())).T
# Prédictions
Z = prediction(XX, W)
Z = Z.reshape((resolution, resolution))
```

On notera que l'argument `resolution` doit être défini en amont dans le script ; de même que la fonction `prediction` qui permet de de prédire dans quelle classe se trouve un point du maillage.

Q7. Définir la fonction `prediction` prenant en argument une matrice X, dont le nombre de colonnes correspond au nombre de variables de l'étude, et W le vecteur des paramètres optimisés.

Enfin, sur la figure initiale, on ajoute les deux demis plans, avec la zone de séparation prédite par la régression logistique :

```
ax.pcolormesh(X0, X1, Z, alpha=0.2) #alpha : transparence...
ax.contour(X0, X1, Z, colors='k') # Frontière
```

Q8. Tracer et observer le résultat de l'apprentissage automatique avec une méthode de régression logistique

Pour évaluer la performance d'un algorithme de classification, on peut regarder la matrice de confusion des prédictions. Dans notre cas de classification binaire, il s'agit d'une matrice qui se présente de la façon

	Classe réelle 0	Classe réelle 1
Classe prédite 0	VRAIS NÉGATIFS (VN)	FAUX NÉGATIFS (FN)
Classe prédite 1	FAUX POSITIFS (FP)	VRAIS POSITIFS (VP)

On peut alors définir plusieurs paramètres complémentaires pour évaluer le modèle, à savoir :

— Sensibilité = Rappel = Taux de vrais positifs = $\dfrac{VP}{VP+VN}$

— Spécificité = Taux de vrais négatifs = $\dfrac{VN}{VN+FP}$

— Précision = $\dfrac{VP}{VP+FP}$

— Justesse = $\dfrac{VP+VN}{VP+FP+VN+FN}$

— Score-F = Moyenne harmonique (Précision, Sensibilité) = $\dfrac{2\times Precision\times Sensibilite}{Sensibilite+Precision}$

Q9. Déterminer la matrice de confusion pour l'algorithme de regression logistique étudié ici et en déduire numériquement la sensibilité (ou rappel), précision, justesse et Score-F

I.3 Utilisation de la bibliothèque scikit-learn

Nous avons vu comment programmer "à la main" un algorithme de régression logistique pour faire de la classification binaire. Cependant, pour des problèmes plus complexes, on peut être amené à utiliser des bibliothèques qui utilisent les mêmes principes que ceux codés précédemment.

Le module `scikit-learn` va nous permettre d'entraîner notre modèle de régression logistique. La régression logistique est implémentée dans la classe `LogisticRegression` du module `linear_model` de scikit-learn. Il existe une multitude de classes dans différents modules de textttscikit-learn. Néanmoins, les méthodes associées sont souvent similaires. On retiendra la méthode `.fit` poour entrainer le modèle sur un jeu de données, ou encore la méthode `.predict` pour prédire des résultats avec le modèle une fois entraîné...

On instancie un objet de la classe de modèle qui nous intéresse ; ici `LogisticRegression` : Nous utilisons une régression logistique « classique », sans pénalisation/régularisation. Il nous faut donc fixer l'argument penalty à none. Il ne reste plus qu'à entraîner cet objet sur les données d'entraînement avec la méthode `.fit`. Il n'est pas utile de normaliser les données.

```
1 from sklearn import linear_model
2 Modele_RL = linear_model.LogisticRegression(penalty='none')
3 Modele_RL.fit(Xt, yt)
```

Q10. Imaginer et concevoir un programme permettant, à partir de la matrice Xt et du vecteur yt de tracer la droite de décision permettant d'illustrer le résultat de la classification, comme cela a été

introduit dans la question 7. On pourra réutiliser le maillage élaboré précédemment pour faire les prédictions du modèle entraîné.

`scikit-learn` nous permet aussi de quantifier la qualité du modèle grâce à différents scores, que l'on trouve dans le module `metrics` : `from sklearn import metrics`. Notamment la fonction `plot_confusion_matrix` permet de tracer de la matrice de confusion sous forme graphique. Cette fonction prend trois arguments : l'objet entraîné, la matrice des entrées et le vecteurs des sorties.

Q11. En utilisant la description précédente, tracer la matrice de confusion pour notre classification binaire. Comparer le résultat avec ce qui a été obtenu précédemment en codant l'apprentissage "à la main".

II Modèle de prédiction de la température annuelle moyenne en France

Dans ce deuxième exemple, on s'intéresse à l'évolution de la moyenne de la température annuelle en France depuis l'année 1900, première année pour laquelle des relevés réguliers ont été réalisés.

II.1 Préparation des données

Le fichier *Temp_annuelle_France.xlsx* comprend deux colonnes dans lesquelles on retrouve l'évolution de la température moyenne annuelle en fonction de l'année.

Q12. Ouvrir le fichier avec la bibliothèque `pandas`. On utilisera la fonction `pd.read_excel` pour ouvrir un fichier de type xlsx. Afficher les données ainsi stockées dans ce nouveau *Dataframe* et créer deux vecteurs `Xt` et `yt` constituant les jeux de données d'entrée et sortie.

Une régression linéaire multiple est envisagée, avec un modèle polynomial. Plusieurs degrés seront comparés (de 0 à 6) afin de valider l'évolution de la température, selon plusieurs modèles.

II.2 Régression avec Régression Linéaire Multiple

Afin de pouvoir effectuer l'apprentissage sur les données étudiées ici, nous allons devoir programmer trois fonctions intermédiaires :

— Fonction modèle `h` prenant en arguments une matrice `X` et le vecteur de paramètres `W` et qui retourne la valeur : $h(Xt.W)$, où h est la fonction polynomiale.

— Fonction coût `J` prenant en arguments la matrice des entrées, le vecteur des sorties et le vecteur des paramètres et calculant le coût (ou l'erreur) de la fonction modèle.

— Fonction Gradient, prenant les mêmes arguments que la fonction précédente et qui sera utilisée dans l'algorithme de descente de Gradient.

Q13. Coder ces trois fonctions, ainsi qu'une fonction globale `Regression_Lineaire_Multiple` prenant en arguments la matrice X_t des observations, le vecteur y_t des étiquettes, le degré d du modèle du modèle polynomial, le nombre d'itérations maximales, le taux d'apprentissage, et qui retourne le vecteur W des paramètres optimisant le coût, ainsi que l'historique de ce coût à chaque passage de boucle sous forme de liste. On prendra garde à ne pas oublier dans la fonction `Regression_Lineaire_Multiple` l'ajout d'une colonne de 1 dans la matrice X_t pour prendre en compte le biais dans le vecteur W, ainsi

qu'un nombre de colonnes correspondant au degré du modèle polynomial choisi (par exemple, on aura une colonne supplémentaire contenant les carrés des entrées pour un degré 2, etc.).

Q14. Tracer l'évolution du coût (qui est, on le rappelle pour une régression linéaire, une erreur quadratique) en fonction du nombre d'itérations. On pourra estimer que la convergence est atteinte lorsque ce coût tend vers une asymptote horizontale.

Q15. Tracer, en la superposant aux points de relevés de température précédent, l'évolution de la température en fonction des années 1900 à 2050. Déterminer quel degré du modèle polynomial semble le plus adapté.

II.3 Utilisation de la bibliothèque scikit-learn

Comme pour la classification binaire, il est possible d'utiliser des fonctions toutes faites, au lieu de re-coder l'ensemble des fonctions. En effet, les modèles linéaires sont implémentés dans le module `linear_model`. La régression linéaire elle-même est implémentée dans la classe `linear_model. LinearRegression`. Comme pour la classification binaire (Régression logistique), on va entraîner un modèle d'apprentissage en suivant la même logique dans `scikit-learn`.

Dans le cas plus général de régression linéaire polynomiale, il est nécessaire de préparer les données en générant des données d'entrées compatible avec une regression polynomiale grâce à `PolynomialFeatures`, que l'on utilisera ainsi :

```
1 from sklearn.preprocessing import PolynomialFeatures
2 transformation_polynomiale = PolynomialFeatures(degree = degre)
3 X = transformation_polynomiale.fit_transform(Xt)
```

Dans les lignes de code ci-dessus, `degre` est un entier indiquant le degré du polynôme, tandis que `X` sera la résultat de la transformation générant une matrice incluant un nombre de colonne correspondant au degré. `Xt` est le vecteur des entrées.

Comme pour la classification binaire avec régression logistique, dans le cas de la régression polynomiale, on instancie un objet de la classe de modèle qui nous intéresse ; ici `LinearRegression()`. De même, on entraînera cet objet sur les données d'entraînement (`(X,yt)` et non `(Xt,yt)`) avec la méthode `.fit`. Enfin, pour prédire des résultats, une fois le modèle entraîné, on utilisera la méthode `.predict`. On remarque alors que quel que soit le type d'apprentissage utilisé, mes méthodes pour entraîner le modèle puis faire des prédictions restent les mêmes...

Q16. A partir de la description précédente, imaginer et concevoir des lignes de code permettant d'entraîner le modèle, puis de prédire l'évolution de la température en fonction des années de 1900 à 2050. On pourra superposer plusieurs courbes selon le degré du polynôme choisi (0 à 6).

Enfin, il lest possible d'évaluer les performances du modèle d'apprentissage grâce au module `metrics`, qui s'importe comme il l'a été fait dans l'exercice précédent. On utilisera la fonction `mean_squared_error` qui prend en arguments `yt`, les étiquettes à prédire, et `yp`, les sorties prédites par le modèle. On aura alors l'erreur quadratique entre les données prédites et à prédire par le modèle entraîné.

Q17. A partir du dataset étudié ici, déterminer l'erreur quadratique pour plusieurs degrés de polynômes choisis et les comparer aux coûts constatés précédemment.