TP 06

Exercice 1 : Un petit échauffement rapide...

Vous disposez d'un jeu de données caractérisant des fruits. Vous disposez :
— `etiquette_fruit` : son label (numéro associé au nom)
— `nom_fruit` : son nom
— `poids`
— `largeur`
— `hauteur`
— `score-couleur` : un score calculé à partir des couleurs observées.

Q1. Récupérez le jeu données `data_mm_fruits.csv` et chargez-le dans un *dataframe*.

Q2. Séparez les données en ensemble d'entraînement et de tests. N'utilisez que le poids, la largeur et la hauteur.

Q3. Affichez les données selon le poids, la taille et la hauteur, par nom, afin de voir si des familles émergent. Pour cela, le module `mpl_toolkits.mplot3d import Axes3D` peut s'avérer utile.

Q4. On peut voir différentes familles se dessiner. Un modèle de prédiction par similarité semble être une bonne piste. Entraînez un modèle *K-Nearest Neighbors* avec d'abord un voisinage de 1, puis avec un voisinage de 5. Vérifiez l'augmentation de la précision du modèle qui doit théoriquement suivre l'élargissement du voisinage.

Q5. Voici une nouvelle observation : `[20,4.3,5.5]`. Utilisez votre modèle pour prédire le type de fruits correspondant.

Exercice 2 : Prise en main des données

Nous allons travailler à la prédiction des chances de survie pour les passagers d'un célèbre navire, le *Titanic*, en fonction de leurs caractéristiques. Le jeu de données est structuré de la manière suivante :
— `PassengerId` (`int`) : numéro d'identification du passager
— `Survived` (`bool`) : 0 si le passager n'a pas survécu, 1 sinon
— `Pclass` (`int`) : numéro de la classe du passager (1, 2 ou 3)
— `Name` (`str`) : nom d'enregistrement du passager
— `Sex` (`str`) : genre du passager (*male* / *female*)
— `Age` (`float`) : age du passager
— `SibSp` (`int`) : nombre de frères /sœurs / conjoint(e) à bord du navire
— `Parch` (`int`) : nombre de parents / enfants à bord du navire
— `Ticket` (`int`) : numéro du ticket
— `Fare` (`float`) : prix du ticket payé à l'embarquement
— `Embarked` (`char`) : une lettre correspondant au type d'embarquement

Nous allons effectué de l'apprentissage supervisé : la colonne qui nous intéresse particulièrement est la colonne `Survived`. C'est cette colonne qu'on va chercher à prédire. Malheureusement, le jeu de données n'est pas forcément utilisable en l'état et quelques ajustements vont être nécessaires.

Q1. Récupérez le jeu de données du fichier `data_mm_titanic.csv`. Chargez-le en mémoire et affichez les 5 premières lignes.

Q2. Commençons par nettoyer les données. D'une manière générale, s'il y a trop de valeurs manquantes, on ne peut rien faire. Il toujours est possible de supprimer les colonnes en question. Cependant, pour certains types de variables, il est possible de substituer une valeur plausible aux valeurs nulles. Regardons ce qu'on peut faire ici.

Q2.1. Déterminez quelles sont les colonnes contenant des valeurs nulles, et combien sont-elles (au total et par colonne) ?

Q2.2. La colonne contenant le plus de valeurs nulles n'est pas utilisable. Supprimez-la de votre jeu de données.

Q2.3. Une colonne contient uniquement 2 valeurs nulles. Comme cette colonne est de type textuelle, on va remplacer les valeurs nulles par la valeur qu'on retrouve le plus souvent dans la colonne. Établissez d'abord la liste des valeurs uniques qu'on retrouve dans la colonne ainsi que le nombre d'occurences associé de chacune. L'utilisation de `mode()` semble particulièrement appropriée.

Q2.4. Reste la dernière colonne contenant des valeurs manquantes à traiter. Quelle valeur pourrait-on substituer aux valeurs manquantes ? Dessinez la distribution empirique des âges et faites apparaître la médiane.

Q2.5. Les données s'étalent de manière significative autour de la valeur médiane : la solution n'est pas très satisfaisante. On peut trouver une meilleure solution (peut-être non optimale mais facile d'accès). Quelle variable est plus corrélée avec l'âge : la classe dans le navire (`Pclass`) ou le prix du billet (`Fare`) ?

Q2.6. Tracez des boîtes à moustaches des âges par catégorie. Si les boîtes sont différentes les unes des autres, il conviendra de remplacer les âges manquants par différentes valeurs.

Q2.7. Remplacez chacune des valeurs manquantes par la valeur médiane des âges des passagers la même classe. La méthode `.apply(...)` permet d'appliquer une méthode à définir par vous mêmes aux éléments d'une colonne, et paraît utile dans ce cas.

Q3. Certaines méthodes préfèrent les valeurs numériques que les chaînes de caractères pour opérer. Prenons le cas du genre (colonne `Sex`). Utilisez la méthode `get_dummies(...)` afin de créer une nouvelle colonne (nommée `male`), équivalente, mais contenant des booléens.

Q4. Une dernière transformation peut être nécessaire : normalisons certaines données.

Q4.1. Affichez la description détaillée du jeu de données. Certains algorithmes d'apprentissage sont sensibles aux différences d'échelles de valeurs (pour les variables non catégorielles). Quelles sont les deux variables qui ont une échelle de valeurs sensiblement différentes des autres ?

Q4.2. Afin de mieux appréhender les tarifs (colonne `Fare`), représentez sur la même figure un nuage de points du numéro d'identification des passagers en fonction du tarif, un histogramme des tarifs et une boîte à moustaches sur les tarifs également.

Q4.3. Étudiez la fonction `pandas.qcut(...)`. Grâce à elle, rajoutez une colonne `farebin`, correspondant au quartile auquel la valeur appartient.

Q4.4. Faites de même pour les âges dans une colonne `agebin`, correspondant à l'un des 6 intervalles auquel chaque valeur appartient.

Q5. Faites-en une copie et supprimez de votre jeu de données toutes les variables qui ne sont pas numériques.

Exercice 3 : Deux modèles de classification

Vous disposez maintenant d'un jeu de données nettoyé. Les valeurs nulles ont été soit supprimées, soit remplacées par une valeur appropriées. Certaines valeurs ont été aussi normalisées au regard des autres.

Q1. Avant de nous lancer sur la prédiction des chances de survie des passagers, il serait bon d'avoir une idée de ces chances selon différentes caractéristiques.

Q1.1. Affichez sur une figure le nombre de survivants et le nombre de victimes.

Q1.2. De la même manière, affichez sur la même figure, le nombre de survivants et de victimes selon leur genre. Y-a-t-il eu plus de victimes chez les hommes ou chez les femmes ?

Q1.3. Enfin, affichez sur la même figure le nombre de survivants et le nombre de victimes selon leur classe sur le navire. Y-a-t-il eu plus de survivants en première classe ou en troisième classe ?

Q2. Découpez votre jeu de données afin d'en conserver 30% pour l'évaluation de vos modèles. Combien obtenez vous d'observations pour l'entraînement et pour l'évaluation ?

Q3. Entraînez votre modèlé de régression logistique et affichez son évaluation.

Q4. Essayons un autre modèle de classification pour cette même tâche. Créez un arbre de décision sur le jeu de données d'entraînement, avec un profondeur maximale de 3. Affichez-le et évaluez la performance de ce modèle.

Q5. Essayons un peu de *bagging* pour voir si ça améliore le modèle. Créez sur les mêmes données d'entraînement un modèle par *Random Forests* de 100 arbres, chacun d'une profondeur maximale de 3. Évaluez votre modèle selon les mêmes métriques que précédemment. A-t-on amélioré notre capacité de prédiction ?

Exercice 4 : Reconnaissance de caractères

Un jeu de données très célèbre est l'ensemble `MNIST`. Il est constitué d'un ensemble de 70000 images 28x28 pixels en noir et blanc annotées du chiffre correspondant (entre 0 et 9). L'objectif de ce jeu de données était de permettre à un ordinateur d'apprendre à reconnaître des nombres manuscrits automatiquement (pour lire des chèques par exemple). Ce dataset utilise des données réelles qui ont déjà été pré-traitées pour être plus facilement utilisables par un algorithme.

Afin d'utiliser ce jeu de données, le préambule suivant est nécessaire :

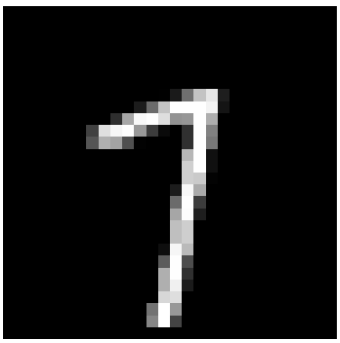
```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1)
```

On accède aux données via un `.data` et au résultat de classification via un `.target`.

Q1. Premières manipulations.

Q1.1. Chargez le *dataset* en mémoire et affichez sa taille.

Q1.2. Récupérez les données liées au 42ème caractère, reconstituez l'image et affichez-la grâce à la méthode `plt.imshow(..., cmap=plt.cm.gray, interpolation='nearest')`. L'option d'interpolation sert juste à lisser l'affichage alors que l'option `cmap` spécifie un affichage en niveaux de gris. Vous devriez obtenir l'image suivante :



Q1.3. De la même manière, affichez la classification réelle de ce caractère.

Q2. Le jeu de données est trop volumineux...

Q2.1. La taille du jeu de données étant assez conséquent, il est nécessaire de faire un échantillonnage et ne travailler que un sous ensemble des données. Extrayez 5000 données sur lesquelles travailler.

Q2.2. Maintenant, il convient de séparer nos données en un jeu d'entraînement (80% du volume) et d'un jeu de test.

Q3. Entrainons un premier modèle.

Q3.1. Entrainez un modèle basé sur les *K-Nearest Neighbors* avec `k=6`.

Q3.2. Réalisez une prédiction pour le 3ème élément de l'ensemble de test et confrontez la prédiction et la réalité.

Q3.3. Évaluez la précision sur l'ensemble de test.

Q4. Malgré les bonnes performances, on peut se poser la question de la pertinence du choix arbitraire de la taille du voisinage.

Q4.1. Cherchons à optimiser notre hyper-paramètre `k`, afin de minimiser notre erreur. Pour trouver la valeur optimale, on va simplement tester le modèle pour tous les k de 2 à 15, mesurer l'erreur test et afficher la performance en fonction de k :

Q5. Entrainez le modèle bien calibré. Affichez en une figure 12 images choisies aléatoirement et les valeurs prédites correspondantes.

Q6. De manière similaire, affichez en une figure de 12 images aléatoires des cas pour lesquels la prédiction était fausse. On affichera l'image reconstitutée et la prédiction.