Projet Science des données

Introduction :

L'objectif général de ce projet est de nous mettre en situation réelle de projet de science des données de type classification. Lors de ce projet, nous avons travaillés sur le jeu de données Weather regroupant les observations de centres météorologiques.
À partir de ces observations, notre objectif sera de prédire s'il pleuvra le lendemain en formant un modèle de classification binaire sur le jeu de données.

Pour cela, nous commencerons d'abord par importer les données puis vérifier si l'importation est correcte en gardant à l'esprit qu'il s'agit d'un objet de type dataframe.

Nous effectuerons ensuite l'examen des données, c'est à dire identifier le type des données puis la qualité des données. Il s'agit principalement de données manquantes, de données qualitatives (de type texte) ou encore des données aberrantes.

Puis viens l'étape de la préparation des données qui consiste en la suppression des variables comportant trop de valeurs manquantes si celles-ci sont trop nombreuses, ou de remplacer les valeurs manquantes par la médiane de la variable pour les variables numériques ou par la valeur suivante pour les variables de type texte. Nous devrons aussi supprimer les valeurs aberrantes observer sur les histogrammes et enfin recalibrer les variables d'entrées afin que notre algorithme d'apprentissage fonctionne au mieux.

Par la suite, nous nous intéresserons aux relations qui existent entre les variables d'entrées et la variable de sortie (ici : RainTomorrow) afin d'identifier lesquelles étaient les plus pertinentes à utiliser pour la classification.

Enfin, on extraira un jeu de test et d'apprentissage de notre jeu de données afin d'entraîner un modèle et d'ensuite l'évaluer. Pour entraîner le modèle, nous utiliserons directement la classe LogisticRegression disponible dans Scikit-Learn. Puis, pour l'évaluer, nous comparerons les classes prédites par le modèle avec les classes fournies par le jeu de données. Pour cela, on utilisera plusieurs métriques fourni par Scikit-Learn.

Pour conclure, nous chercherons à améliorer l'évaluation du modèle en utilisant la méthode de validation croisée.

I) Importation des données

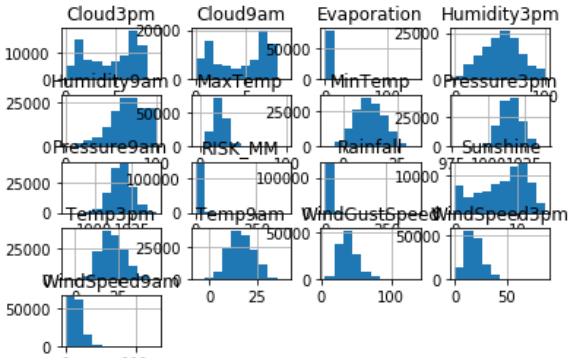
J'ai d'abord commencer par importer le fichier « weather.csv » en mémoire en utilisant la fonction read_csv. Ce qui me permettra par la suite de pouvoir manipuler les données sans modifier le jeu de données original.

Puis à l'aide de la fonctionnalité info, me renvoyant les informations suivantes, j'ai pu vérifier que l'importation était correcte. En effet, on peut y voir le nombre de colonnes, le nombre de lignes ainsi que le type de chaque colonnes. Information facilement vérifiable sur le jeu de données.

```
Nombres de lignes et type de l'objet
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 142193 entries, 0 to 142192
Data columns (total 24 columns):
Date             142193 non-null object
Location         142193 non-null object
MinTemp          141556 non-null float64
MaxTemp          141871 non-null float64
Rainfall         140787 non-null float64
Evaporation      81350 non-null float64
Sunshine         74377 non-null float64
WindGustDir      132863 non-null object
WindGustSpeed    132923 non-null float64
WindDir9am       132180 non-null object
WindDir3pm       138415 non-null object
WindSpeed9am     140845 non-null float64
WindSpeed3pm     139563 non-null float64
Humidity9am      140419 non-null float64
Humidity3pm      138583 non-null float64
Pressure9am      128179 non-null float64
Pressure3pm      128212 non-null float64
```

II) Examen des données

Grâce a la méthode info(), nous connaissons déjà le type de chaque variables ainsi que la taille du jeu de données.
Lors de l'examen des données, nous devons aussi identifier les valeurs aberrantes et manquantes.
Afin d'identifier le maximum de valeurs aberrantes et manquantes, j'ai utiliser la commande .hist() me permettant de visualiser un histogramme de chaque variables.



Malheureusement, ceux-ci sont très peu lisibles, et je n'ai seulement pu qu'isoler les valeurs aberrantes concernant la température maximum. Ici notre critère de sélection était : Si Tmax > 50, alors cette valeur est aberrante.

III) Préparation des données

Nous avons d'abord commencé par quantifier le nombre de valeurs manquantes pour chaque variables en utilisant la méthode isna() et isna.sum(). La méthode isna(), nous renvoie un tableur alors que la méthode isna.sum() nous renvoie directement le nombre total de valeurs manquantes.

```
[142193 rows x 24 columns]
Somme des valeurs manquantes par colonnes
Date                0
Location            0
MinTemp           637
MaxTemp           322
Rainfall         1406
Evaporation     60843
Sunshine        67816
WindGustDir      9330
WindGustSpeed    9270
WindDir9am      10013
WindDir3pm       3778
WindSpeed9am     1348
WindSpeed3pm     2630
Humidity9am      1774
Humidity3pm      3610
Pressure9am     14014
Pressure3pm     13981
Cloud9am        53657
Cloud3pm        57094
Temp9am           904
Temp3pm          2726
RainToday        1406
RISK_MM             0
RainTomorrow        0
dtype: int64
```

On peut s'apercevoir que pour les variables Evaporation, Sunshine, Cloud9am et Cloud3pmnous avons énormément de valeurs manquantes.
Nous avons donc fait le choix de supprimer ses 4 variables, les données étant trop peu nombreuses.
Pour cela, nous utilisons la fonction del data['nom colonne'].

Nous nous sommes ensuite occupés des valeurs aberrantes, ainsi nous utilisons une boucle for pour parcourir toutes les données disponibles dans la colonne MaxTemp et supprimons toute celles supérieur à 50. Nous avons décidé de d'abord les supprimés afin que ces fausses valeurs ne rentre pas en compte dans le calcul de la médiane.

Enfin nous nous occupons des valeurs manquantes :

- Pour les variables numériques ne possédant pas trop de valeurs manquantes, nous avons décidé de remplacer celles-ci par la valeur médiane en utilisant la méthode fillna(data.median(), inplace=True).

- Pour les variables de type texte, nous avons décidé de remplir les valeurs manquantes par la prochaine valeur correcte disponible. Pour cela, nous utilisons la méthode fillna(method= 'bfill', inplace=True)

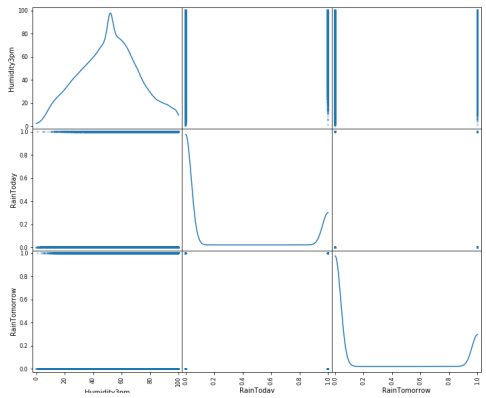
Ensuite, pour la classification il était nécessaire de transformer les variables textuelles en variables numériques. De ce fait, nous avons utilisé LabelEncoder() qui nous permet directement de transformer un texte en une valeur numérique spécifique.
n dernier lieu, nous avons aussi séparé les variables d'entrées de la variable de sortie en les stockant dans deux variables X et y.
Enfin, pour que notre algorithme d'apprentissage soit optimal, nous avons utilisé la classe StandardScaler sur toutes les variables d'entrées (maintenant complètement numérique). Cette classe permet de centré toute les variables sur 0 et de réduire l'écart type de chaque variables. Nous ne l'utilisons pas sur la variable de sortie car celle-ci est binaire.

IV) Recherche de corrélations

Dans cette partie, nous voulions mieux comprendre les relations entre la variable de sortie et la variable d'entrée. Pour cela, nous avons calculés les différents coefficients de corrélation entre la variable de sortie et toutes les variables d'entrées. Or on sait que la corrélation entre deux variables statistiques mesure leur degré de dépendance, et prend une valeur comprise entre -1 et 1. Si la corrélation en norme est proche de 0, cela signifie que les variables sont (très) indépendantes, si elle est proche de 1 cela marque une forte dépendance des variables. Grâce à cela, nous pourrons donc savoir quelles sont les variables d'entrées les plus pertinentes pour la classification.
Calcul des coefficients :

```
Calcul du coefficient de corrélation de la colonne  Date  avec
RainTomorrow
-0.010913468341621615
Calcul du coefficient de corrélation de la colonne  Location
avec RainTomorrow
-0.003608407901900166
Calcul du coefficient de corrélation de la colonne  MinTemp  avec
RainTomorrow
0.08374291984604822
Calcul du coefficient de corrélation de la colonne  MaxTemp  avec
RainTomorrow
-0.15903914817781634
Calcul du coefficient de corrélation de la colonne  Rainfall
avec RainTomorrow
0.2351328426180901
Calcul du coefficient de corrélation de la colonne  WindGustDir
avec RainTomorrow
0.05185760458294973
Calcul du coefficient de corrélation de la colonne  WindGustSpeed
avec RainTomorrow
0.22475166546441525
Calcul du coefficient de corrélation de la colonne  WindDir9am
avec RainTomorrow
0.03924574286868182
Calcul du coefficient de corrélation de la colonne  WindDir3pm
avec RainTomorrow
0.030829821287014024
Calcul du coefficient de corrélation de la colonne  WindSpeed9am
avec RainTomorrow
0.09045492323479096
Calcul du coefficient de corrélation de la colonne  WindSpeed3pm
avec RainTomorrow
0.08693022147832623
Calcul du coefficient de corrélation de la colonne  Humidity9am
avec RainTomorrow
0.25532981556481943
Calcul du coefficient de corrélation de la colonne  Humidity3pm
avec RainTomorrow
0.4397275295165434
```

Nous identifions deux variables que nous jugeons pertinentes : RainToday ,Humidity3pm
En effet, leurs coefficients de corrélations est respectivement de 0,33 et de 0,44. Ce qui montre qu'ils sont très liés à notre variable de sortie. Par la suite, Humidity3pm et RainToday seront donc nos seules variables d'entrées.
Enfin, nous utilisons la fonction scatter_matrix, pour visualiser les nuages de points correspondants.



Malheureusement, les graphiques obtenues sont difficilement exploitable. En effet, aucune loi ne semble ressortir de ceux-ci.

V) Extraction des jeux d'apprentissage et de test

Créer ces deux jeux consiste à choisir au hasard des éléments dans le jeu de données et à les placer dans deux ensembles distincts. Pour réaliser cela, nous avons utilisé la fonction train_test_split() qui nécessite que nous transformations nos variables d'entrées et de sortie en tableaux Numpy grâce à la commande .values. Cette fonction nous renvoie quatre tableaux de numpy correspond aux jeux d'apprentissage et de test. 20 % des données étant répartit dans le jeu de test et 80 % dans le jeu d'apprentissage.

VI) Entraînement d'un modèle

Nous pouvons désormais utiliser notre jeu d'apprentissage pour entraîner notre modèle destiner à la classification. L'algorithme que nous utiliserons sera celui fourni par la classe LogisticRegression().
Le principe de cet algorithme est que la sortie ne peut prendre que deux valeurs 0 et 1 (c'est pourquoi nous n'avons pas utiliser StandardScaler sur la sortie).
Afin de mieux comprendre l'algorithme, nous reprenons le principe de la régression linéaire pour l'appliquer à un problème de classification.

VII) Évaluation du modèle

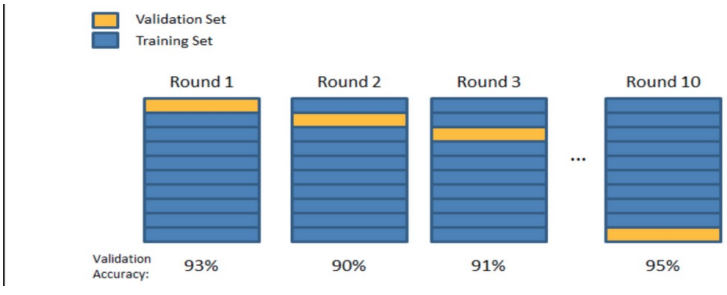
Maintenant que nous avons entraîner le modèle nous pouvons prédire les classes sur le jeu de test.
On utilise LogisticRegression().predict() pour effectuer cette prédiction.
Dans un premier temps, nous avons simplement effectuer une boucle afin d'afficher cote à cote les premiers termes de la prédiction et du jeu de test. Ceux-ci nous permets de comparer visuellement nos résultats.
Lorsque nous affichons les 20 premiers termes nous nous rendons comptes qu'il n'y aucunes erreurs.
Mais pour plus de précision nous utilisons 5 outils mises à notre disposition par Scikit-Learn (accuracy_score(),confusion_matrix(),precision_score(), recall_score(), f1_score()). En effet, ses cinqs outils nous permettent d'évaluer notre modèle en nous donnant un score correspondant à leurs évaluations propres . Voici ce que nous obtenons :

```
accuracy_score: 0.8252031090634122
confusion_matrix : [[20856  1082]
 [ 3888  2607]]
precision_score : 0.7066955814583898
f1_score : 0.5119795758051847
recall_score : 0.40138568129330254
```

Nous pouvons nous apercevoir que les résultats des cinq évaluations sont différentes. Cela semble normal car chaque modèle possède ses propres critères.

VIII) Amélioration de l'évaluation

Afin d'améliorer l'évaluation de notre modèle, nous avons voulu utiliser la méthode de validation croisée. Elle consiste à découper aléatoirement le jeu d'entraînement en plusieurs sous-ensembles distincts puis à entraîner et à évaluer le modèle en passes successives. A chaque passe, un bloc est réservé pour l'évaluation et les blocs restants sont utilisés pour l'entraînement.



En utilisant la classe Kfold et la fonction cross_val_score nous obtenons les résultats de cette évaluation croisée. En effet, nous obtenons :

```
  FutureWarning)
[0.83844423 0.8267567  0.83054305 0.81422341 0.82301632 0.82125774
 0.83096511 0.82787001 0.82210186 0.83511536]
```

Chaque nombre correspond à un accuracy_score pour un bloc entrainement. Ici nous pouvons voir que tous les blocs obtiennent des scores très proches les uns des autres mais aussi très proche de notre valeur sans validation croisée. Ce qui nous paraît peu cohérent car la méthode d'évaluation croisée permet de réduire la variance associée à un seul essai de séparation.

Conclusion :

Lors de ce projet, nous avons du nous plonger dans une situation proche de la réalité. En effet, les étapes que nous avons du respecter correspond à beaucoup de projets de type classification :
1) Importer les données puis vérifier que celles-ci correspondent au jeu de données de bases
2)Examiner les données pour identifier les données manquantes, aberrantes ou encore textuelle (celles-ci nécessitant d'être transformer en données numériques pour la classification)
3)Préparation des données pour éliminer toutes les données manquantes, aberrantes puis standardiser toutes les valeurs pour que notre algorithme fonctionne parfaitement.
4)Recherche de corrélations pour éliminer les variables qui n'ont que très peu de liens avec la variable de sortie RainTomorrow. Ici, nous avons seulement garder deux variables : RainToday et Humidity3pm.
5)Extraction du jeu d'apprentissage et de test pour créer les listes nécessaire à l'entraînement et à l'utilisation de notre algorithme.
6)Entraînement du modèle avec la classe LogisticRegression().
7)Evaluation du modèle à partir des données de test et des données de prédictions grâce aux nombreux métriques disponibles.
8)Amélioration de l'évaluation grâce à l'évaluation croisée et la méthode Kfold et la fonction cross_val_score.