Andréa WAUTERS Narjisse LASRI

Projet Science des données

Introduction :

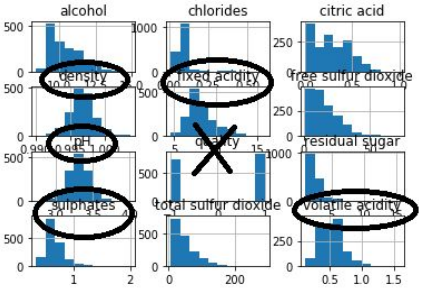
Nous disposons pour ce projet, d'un jeu de données indiquant certaines propriétés chimiques de vins que l'on souhaite utiliser pour prédire la qualité de ces vins. Nous traitons ce sujet à l'aide du langage de programmation python. Ces données numériques sont déjà annotées avec les attributs suivants : "fixed acidity", "volatile acidity", "citric acid","residual sugar", "chlorides", "free sulfur dioxide", "total sulfur dioxide", "density", "pH", "sulphates", "alcohol", "quality". La répartition des vins est faite selon deux critères de qualité : "-1" et "1", nous sommes donc face à un problème de classification binaire. Nous retrouvons ces informations sous python à l'aide de la bibliothèque Pandas qui nous permet de classer nos données dans un tableau (voir ci-dessous un extrait).

```
Les classes sont :
         fixed acidity  volatile acidity  ...  alcohol  quality
quality                                   ...
-1                  76               117  ...       42        1
 1                  82               107  ...       58        1
```

1. La préparation des données

La préparation des données joue un rôle essentiel dans la suite de leur traitement, c'est elle qui détermine le degré d'exigence sur les données et c'est d'elle que résulte le nombre de données acceptées, leur précision et leur véracité. En voici les différentes parties :

- L'identification et la correction, le cas échéant, des données manquantes ou aberrantes. Elle peut se décomposer comme il suit :
    1) Repérer dans un premier temps les valeurs aberrantes. Pour cela nous faisons une analyse indépendante de chaque colonne du tableau de données pour repérer les valeurs qui :
    - Ne sont pas numériques : nulles ou autre.
    - Sont supérieures à 14 pour l'attribut "pH".
    - Enfin, suite à l'utilisation de la fonction *red_wines.hist()* on se rend compte que les courbes correspondant aux attributs : density, pH, sulphates, fixed acidity et volatile acidity ont une allure gaussienne. On peut donc appliquer le principe de statistique des lois gaussiennes suivant : les valeurs doivent être comprises dans l'intervalle : [moyenne - 3*écart_type ; moyenne + 3*écart_type]. Les valeurs en dehors de cet interval sont considérées inexactes. On utilise pour cela les fonctions *.mean()* et *.std()* pour récupérer moyenne et écart type.



    - On se rend compte que la colonne "quality" ne comporte pas de valeurs différentes de 1 ou -1. Il n'y a donc pas de valeurs aberrantes à repérer.

2) Puis il faut remplacer ces valeurs par null en utilisant dans la bibliothèque Numpy : *np.nan*. Dans notre code, une fonction permet de réaliser ces deux premières étapes.
3) Nous identifions ensuite la proportion de lignes qui contiennent des valeurs nulles. Cette étape nous donne une critère de décisions pour le choix des deux prochaines étapes :
4) On peut décider de supprimer les lignes contenant ces valeurs nulles à l'aide de la fonction *dropna* (supprime les lignes qui contiennent des données non numériques).
5) Ou alors de calculer les moyennes des colonnes et les utiliser pour remplacer les valeurs nulles. On utilise pour cela la fonction *fillna* qui remplace toutes les données non numériques par la valeur entrée en paramètre. Néanmoins cette méthode n'est pas applicable à la colonne d'attribut "quality". Nous avons donc décidé de supprimer les lignes sur lesquelles "quality" n'a pas de valeur numérique.

- Pour trouver la corrélation entre les attributs, nous utilisons d'abord la méthode de corrélation "Pearson" qui ne peut être utilisée que pour des attributs de relation linéaire. On considère que si 2 colonnes sont corrélées à avec un coefficient de corrélation de 0.5 ou plus, l'une des colonnes peut être supprimée.
    - Nous créons donc une fonction qui crée une matrice de corrélation sur les données, puis qui la parcourt pour chercher les valeurs supérieures à 0.5 et supprime les colonnes corrélées.
    - Les attributs linéairement corrélés sont : ['citric acid et fixed acidity', 'total sulfur dioxide et free sulfur dioxide', 'density et fixed acidity'].
    - De ce fait nous pouvons supprimer les colonnes : ['citric acid', 'total sulfur dioxide', 'density'].

  Pour trouver les attributs corrélés mais sans relation linéaire. Nous utilisons la méthode "Spearman". La corrélation de Spearman est étudiée lorsque deux variables statistiques semblent corrélées sans que la relation entre les deux variables soit de type affine. Nous ne remarquons aucuns attributs corrélés sans relation linéaire.

- La détermination de la proportion de données dans chaque classe se fait à l'aide de la fonction : *value_counts()* qui nous renvoie le nombre de vins de bonne (quality = 1) et mauvaise qualité (quality = -1) dont voici les résultats :
    - Avec suppression des lignes : BQ = 809 et MQ = 688

```
Proportion de vin de bonne qualité : 53.470919324577864
Proportion de vin de mauvaise qualité : 46.529080675422136
```

    - Avec remplacement des valeurs nulles par la moyenne : BQ = 855 et MQ = 744

```
Proportion de vin de bonne qualite : 54.041416165664664
Proportion de vin de mauvaise qualité : 45.958583834335336
```

  On peut noter que le rapport entre ces deux résultats est de l'ordre de 1-2 %.

- Pour centrer et réduire les données, nous soustrayons d'abord aux données la moyenne de chaques attributs que nous divisons ensuite par l'écart-type des attributs associés.
  $$T = \frac{X - m}{\sigma}$$
  Nous n'effectuons pas cette démarche sur l'attribut "quality" car il correspond aux classes et ne doit pas être modifié.

2. Evaluation des classifieurs

Pour évaluer les classifieurs, nous utilisons deux méthodes :

- La première méthode est la validation croisée. Nous choisissons k =10 et nous effectuons donc au total 10 itérations pour chaque classifieur.
- La deuxième méthode est de scinder une unique fois les données en : une partie de données d'apprentissage et une partie de données de test .

Nous choisissons ensuite trois métriques d'évaluation : le rappel (recall), le score F1 (F1 score) et le taux de reconnaissance (accuracy). Nous n'utilisons pas la précision (precision) car elle est en fonction du rappel et du score F1, de ce fait elle n'apporte pas de nouvelle information.

Pour les deux méthodes, nous testons les classifieurs suivants : régression logistique, machines à vecteurs supports, analyse discriminante linéaire, analyse discriminante quadratique, k-plus proches voisins, arbres de décision, le perceptron créé par notre binôme et le perceptron de sklearn.

3. Analyse des résultats

1) La validations croisée

Nous testons d'abord les classifieurs avec la validations croisée. Les scores des classifieurs oscillent entre 0.64 et 0.78. Le classifieur le plus performant est donc l'arbre de décision avec le score maximal d'environ 0,78.

score total avec validation croisée

	recall	f1	accuracy
LogisticRegression	0.721836	0.746989	0.736805
svm.SVC	0.736728	0.767049	0.758183
LinearDiscriminantAnalysis	0.704599	0.738610	0.731472
QuadraticDiscriminantAnalysis	0.685957	0.727926	0.723477
KNeighborsClassifier	0.776188	0.756926	0.731436
tree.DecisionTreeClassifier	0.772531	0.776362	0.760197
Notre perceptron	0.659074	0.641455	0.611132
SKlearn	0.680910	0.674999	0.647852

Nous annulons ensuite la normalisation des données, un message d'erreur "warning" s'affiche. Les scores obtenus en sortie sont du même ordre de grandeur qu'avec la normalisation. Le fait de ne pas normaliser peut donc entraîner une impossibilité des erreurs lors de la compilation.

Nous normalisons à nouveau les données et nous annulons la suppressions des colonnes corrélées. On remarque que les scores ont toujours la même performance. Enlever les attributs corrélés permet une optimisation sans perte de performance.

score total avec validation croisée

	recall	f1	accuracy
LogisticRegression	NaN	NaN	NaN
svm.SVC	NaN	NaN	NaN
LinearDiscriminantAnalysis	NaN	NaN	NaN
QuadraticDiscriminantAnalysis	NaN	NaN	NaN
KNeighborsClassifier	NaN	NaN	NaN
tree.DecisionTreeClassifier	NaN	NaN	NaN
Notre perceptron	0.0	0.0	0.465291
SKlearn	NaN	NaN	NaN

Nous supprimons à nouveau les colonnes corrélées et nous n'utilisons pas les fonctions de nettoyage des données, un message d'erreur s'affiche et il est impossible d'évaluer les classifieurs.

2) Division en données d'apprentissage et de test

Nous testons ensuite les classifieurs avec la division en données d'apprentissage (67%) et de test (33%) en une fois. Il est difficile d'avoir des résultats précis avec cette méthode car les scores changent à chaque nouvel appel de cette division. En moyenne, les scores sont du même ordre de grandeur et le classifieur "k-plus proches voisins" obtient les meilleurs résultats.

score total avec division des données

	recall	f1	accuracy
LogisticRegression	0.729323	0.736243	0.719192
svm.SVC	0.728625	0.752399	0.739394
LinearDiscriminantAnalysis	0.725564	0.749515	0.739394
QuadraticDiscriminantAnalysis	0.718631	0.747036	0.741414
KNeighborsClassifier	0.797619	0.755639	0.737374
tree.DecisionTreeClassifier	0.735075	0.732342	0.709091
Notre perceptron	0.754513	0.664547	0.573737
SKlearn	0.651341	0.608229	0.557576

Nous changeons ensuite la répartition des données avec 95% de données d'apprentissage et 5% de données de test. Les résultats varient beaucoup à chaque nouveau test des classifieurs. On peut en conclure que tester les classifieurs avec seulement 5% de données ne permet pas d'avoir un test fiable des classifieurs.

Le fait de ne pas normaliser amène ,pour ce test, un message d'erreur mais cela n'agit pas sur la performance des classifieurs. Lorsque les colonnes corrélées ne sont pas supprimées, les performances restent les mêmes. Le non nettoyage des données empêchent l'utilisation des classifieurs. Il est donc important de normaliser et nettoyer les données correctement avant d'utiliser les classifieurs et il est conseillé de supprimer les colonnes corrélées.

4. Implémentation de l'algorithme du perceptron

Après avoir testé plusieurs classifieurs, nous configurons le nôtre dans cette dernière partie de projet : le perceptron. Au travers de l'implémentation de cet algorithme d'apprentissage supervisé nous détaillerons son fonctionnement :

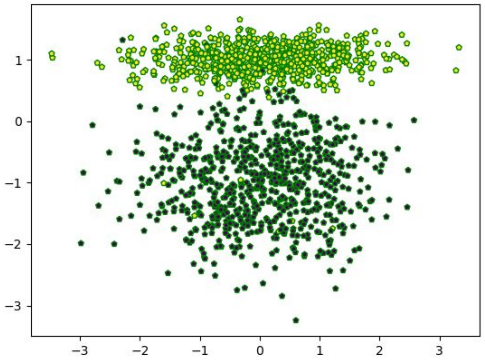
- L'initialisation de la classe permet de garder en mémoire les paramètres du test, importants pour la suite dont : n -> le pas d'ajustement de la courbe de séparation, e -> la marge d'erreur acceptable et le nombre maximum d'itérations pour établir la courbe de séparation.
- Cette courbe de séparation est définie à l'aide de ses coefficients directeurs initialisés à 1 dans la méthode "fit" (bibliothèque numpy fonction *np.ones*). (nombre d'attributs + 1 = nombre de coefficients). Elle permet de classifier les données selon leur position : au dessus ou en dessous.
- La méthode fit permet grâce à un échantillon de données d'adapter les coefficients directeurs de la courbe de cette manière :
    - On procède dans un premier temps au mélange des données grâce à la fonction : *np.random.shuffle(X).*

- Puis une première boucle permet d'ajuster la courbe jusqu'à ce que le maximum d'itération soit atteint. Et de réinitialiser l'erreur (évoquée dans le point suivant) à 0.
- A l'intérieur de cette première boucle, une deuxième boucle permet de parcourir l'intégralité des données et leur classe associée. Dans celle-ci on compare l'égalité entre la position de la donnée courante par rapport à la courbe et donc la classe qui y est associée et entre sa classe réel. Si l'égalité n'est pas vérifiée dans ce cas on incrémente l'erreur de la distance entre la courbe et la donnée courante (= somme des coeff dir * donnée avec *np.dot*). Si l'erreur est inférieure à e (définie précédemment), alors on peut arrêter l'ajustement de la courbe (utilisation d'un *return*). Dans le cas inverse on incrémente nos coefficients directeurs de n * la donnée * sa classe réel (+1 ou -1 car classification binaire).

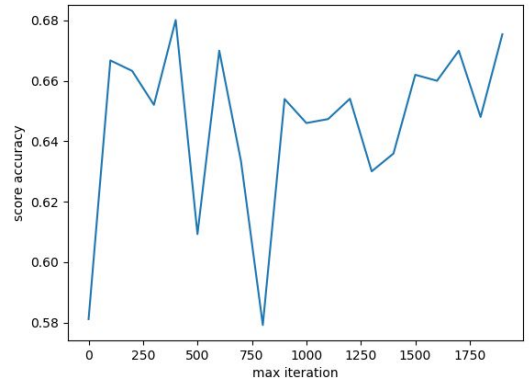
- La méthode predict permet de prédire pour une ou plusieurs données la classe associée. Pour cela une boucle parcourt l'intégralité des données et détermine sa classe : 1 si la donnée courante est au dessus de la courbe préalablement définie avec fit ou -1 si elle est en dessous. Toutes ces prédictions sont insérées dans un tableau retourné par cette méthode.

Avant d'utiliser notre classifieurs sur nos données de vins, nous décidons de le tester sur une séquence de données générées aléatoirement avec uniquement deux attributs (plus simple à visualiser). Notre but est de créer une liste de données simple et suffisante : 1500 en tout. Pour cela :

- De la bibliothèque sklearn on utilise la fonction : make_classification qui nous retourne un tableau de données et un tableau des classification binaire associées (celui-ci étant défini avec les valeurs 0 et 1 on remplace avec *np.where* les 0 par -1).
- Puis ensuite, nous testons le perceptron avec la validation croisée en utilisant k=10 (10 itérations). Nous testons également le Perceptron de Sklearn pour disposer d'un modèle de comparaison.
  Nous obtenons des scores pour les deux classifieurs, variant de 0.96 à 0.98 pour recall, F1 score et accuracy. Nous pouvons en déduire que Le Perceptron que nous avons implémenté a de bonnes performances et nous le validons.



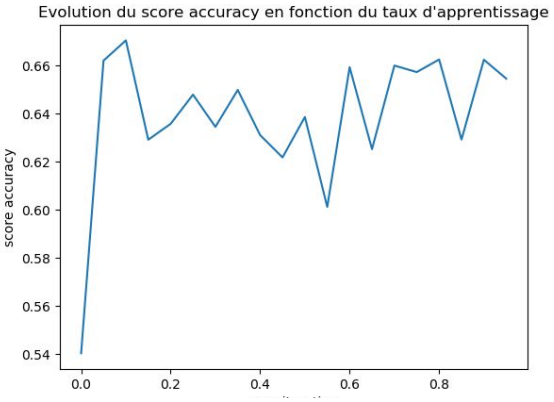
Nous pouvons désormais, utiliser le classifieur Perceptron que nous avons créé sur les données du projet.
Pour tester le Perceptron sur les données du projet, nous cherchons d'abord les meilleurs paramètres à lui fournir :



- Nous fixons l'erreur à 0,1 car elle varie d'un pas moyen supérieur ou égal à 0,1
- Nous cherchons ensuite le meilleur maximum d'itérations. Pour cela, nous fixons d'abord le taux d'apprentissage à 0,15 par défaut et l'erreur à 0,1. Nous faisons varier le max d'itérations de 0 à 2000.

Un max pour le maximum d'itérations se trouve à 400. Nous décidons de fixer notre maximum d'itérations à 400 pour les prochains tests sur les données.

- Nous faisons ensuite varier le taux d'apprentissage de 0 à 1 en gardant le max d'itérations égal à 400 et l'erreur égale à 0,1. On remarque un max pour taux d'apprentissage se trouvant à 0.1 avec un score accuracy de 0,670577. La meilleure valeur du taux d'apprentissage est donc de 0.1.



Pour finir comparons le Perceptron avec les autres classifieurs. Avec la validation croisée et la division simple, ses score sont légèrement moins élevés que les autres types de classifieurs. En comparaison les scores du Perceptron de Sklearn sont également moins élevés. Cela n'est donc pas dû à notre implémentation mais au modèle du Perceptron qui est moins efficace pour mesurer la qualité des vins.

score total avec validation croisée

	recall	f1	accuracy
LogisticRegression	0.721836	0.746989	0.736805
svm.SVC	0.736728	0.767049	0.758183
LinearDiscriminantAnalysis	0.704599	0.738610	0.731472
QuadraticDiscriminantAnalysis	0.685957	0.727926	0.723477
KNeighborsClassifier	0.776188	0.756926	0.731436
tree.DecisionTreeClassifier	0.772531	0.776362	0.760197
Notre perceptron	0.659074	0.641455	0.611132
SKlearn	0.680910	0.674999	0.647852

score total avec division des données

	recall	f1	accuracy
LogisticRegression	0.729323	0.736243	0.719192
svm.SVC	0.728625	0.752399	0.739394
LinearDiscriminantAnalysis	0.725564	0.749515	0.739394
QuadraticDiscriminantAnalysis	0.718631	0.747036	0.741414
KNeighborsClassifier	0.797619	0.755639	0.737374
tree.DecisionTreeClassifier	0.735075	0.732342	0.709091
Notre perceptron	0.754513	0.664547	0.573737
SKlearn	0.651341	0.608229	0.557576

Conclusion :

Dans ce projet, nous avons mis en place un processus complet de classification supervisée : nettoyage des données, suppressions des colonnes corrélées, normalisation, test et évaluation de classifieurs. Cela nous a permis de d'analyser et observer la meilleure méthode et le classifieur le plus performant dans le cadre de la prédiction de la qualité des vins : le classifieur 'arbre de décision' avec une validation croisée. Nous avons démontré que la normalisation et le nettoyage des données étaient essentiels pour mener à bien ce type de projet. La suppression des colonnes corrélées permet quant à elle une meilleure optimisation.

Nous avons également implémenté une méthode de classification supervisée : l'algorithme du perceptron pour ensuite l'appliquer aux données du projet. Nous avons démontré que le choix paramètres des classifieurs avaient un rôle prépondérant dans l'obtention d'un score optimal, en établissant un méthode de repérage des meilleurs paramètres pour le Perceptron.