TP2 : Les k plus proches voisins

Classification

Objectifs

Dans ce TP, nous allons travailler sur la base des iris de Fisher. Il s'agit de prédire l'espèce d'iris en fonction de différentes caractéristiques végétales. Nous allons utiliser l'algorithme des k-plus proches voisins afin de réaliser notre classification.

Analyse des données

Commençons par récupérer la base de données des iris. Elle est disponible ici. Nous allons la charger dans un dataframe `pandas` à l'aide de la fonction `read_csv`.

Avec la bibliothèque `pandas`, il existe plusieurs fonctions permettant une première analyse simple des données:

- L'attribut `shape` permet de connaitre les dimensions du dataframe.
- La fonction `info` permet d'avoir un résumé rapide des données.
- La fonction `describe` permet d'avoir des statistiques sur différentes tendances sur les données.
- La fonction `head` permet d'afficher les premières lignes du dataframe.
- La fonction `value_counts` permet de compter le nombre d'occurrences de chaque valeur.

Utiliser ces fonctions pour répondre à ces questions :

- Combien de classes ?
- Combien de caractéristiques descriptives ? De quels types ?
- Combien d'exemples ?
- Combien d'exemples de chaque classe ?
- Comment sont organisés les exemples ?

Séparation des données en bases d'apprentissage et de test

La bibliothèque sklearn fournit la fonction `train_test_split` qui permet de séparer la base. Pour cela, nous allons utiliser `from sklearn.model_selection import train_test_split`. Cette fonction a l'avantage de randomiser l'ensemble avant de faire le split, ce qui est très important avec cette base des iris.

Apprentissage et test

Il nous faut maintenant apprendre notre modèle. Commencez par créer un knn en python (`KNeighborsClassifier()`). Ensuite il faut l'entrainer sur la base d'apprentissage avec la fonction `fit`.

On peut mesurer la précision de notre modèle avec la fonction score. Quel score obtenez vous en apprentissage ? Et en test ?

Afficher ensuite la matrice de confusion. Qu'observez vous ?

Jouer avec le paramètre k

Etudiez l'influence du paramètre k.

Regression

Objectif

L'algorithme des k-plus proches voisins est aussi utilisé pour des problèmes de regression. Dans cette seconde partie, nous allons étudier la base de données *auto-mpg* disponible ici. Il s'agit de prédire la consommation (miles per gallon - mpg) en fonction du nombre de cylindres, du déplacement, de la puissance, du poids, de l'accélération, de l'année et de l'origine. La dernière colonne correspond au nom de la voiture, il est unique et ne sera donc pas utile pour notre apprentissage, il nous faut donc ignorer cette colonne.

A vous de jouer

Comme pour la première partie, récupérez la base, analysez les données, et effectuez un apprentissage avec knn.
Là encore étudiez l'influence du paramètre k.
Vous devez utiliser les bibliothèques suivantes :

- `from sklearn.neighbors import KNeighborsRegressor`
- `from sklearn.model_selection import train_test_split`

- `from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score`

Afin d'évaluer les performances de nos modèles, nous allons introduire deux nouvelles mesures pour les problèmes de regression : **Mean Absolute Error (MAE)** et **Mean Squared Error (MSE)**.

MAE est la moyenne des valeurs absolues des erreurs de prédiction, elle est donnée par la formule suivante :

$$MAE = \frac{1}{n}\sum_{i=0}^{n-1}|y_i - \hat{y_i}|$$

avec $\hat{y}$ la valeur prédite pour le ième exemple, et y_i la vraie valeur pour le ième exemple. MSE est une mesure d'erreurs classiques, on la retrouve très souvent. Elle correspond à la moyenne des carrés des erreurs de prédiction. Elle est définie par la formule suivante :

$$MSE = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y_i})^2$$

avec $\hat{y}$ la valeur prédite pour le ième exemple, et y_i la vraie valeur pour le ième exemple.

Quelle que soit l'erreur utilisée, il est important que dans le cadre d'une régression la mesure de performance ne tienne pas compte de la direction de l'erreur, car dans le cas contraire, une erreur "sur-prédite" et une "sous-prédite" s'annuleraient. MAE et MSE tiennent comptent de ça en prenant la valeur absolue et le carré des différences. MSE donnent plus de poids aux erreurs sur les points abérrants (grosses erreurs).

Comme vu en cours, pour cet algorithme il est important de normaliser les données. Utilisez `from sklearn.preprocessing import StandardScaler` et en particulier la fonction `fit_transform` qui permet de normaliser les données.

Résumé

- Knn est un algorithme simple à mettre en place et facile à comprendre.
- Il permet de traiter à la fois des problèmes de classification et de regression.
- Il est important d'avoir des données normalisées pour cet algorithme.