ML : K-nearest Neighbors

TP2 Apprentissage automatique

Les k plus proches voisins

Jérôme Buisine
jerome.buisine@univ-littoral.fr

25 septembre 2023

Durée : 3h

L'objectif de ce TP est d'exploiter l'algorithme des k plus proches voisins pour concevoir des modèles pour des problèmes à la fois de classification et de régression. De plus, le TP permettra également une meilleure prise en main de la librairie `scikit-learn`.

1 Ressources

Voici un ensemble de ressources qui peuvent vous être utiles durant ce TP :

— 1. Documentation officielle de l'outil `Git` ;
— 2. Gitlab : interface web pour la gestion de projets versionnés sous `Git` ;
— 3. pyenv : permet la gestion d'environnement Python.
— 4. matplotlib : librairie Python qui permet un affichage rapide de données.
— 5. pandas : librairie Python qui permet de lire des données et les traiter.
— 6. jupyter : un outil de travail permettant une interaction rapide et visuelle avec une console Python.
— 7. scikit-learn : librairie proposant des outils pour l'apprentissage automatique.

2 Configuration de l'environnement

Créez un dossier à la racine de votre projet nommé `tp2-knn`. Dans ce dossier et depuis votre terminal, lancez les commandes suivantes :

```
# spécifie l'environnement virtuel Python à Jupyter
ipython kernel install --user --name=ml-venv
# lance l'application Jupyter
jupyter-lab
```

Remarque : pour chaque section suivante, il vous sera demandé de créer un notebook depuis Jupyter. Faites bien attention à la sélection de l'environnement Python lors de sa création. De plus, il vous sera demandé de bien documenter votre code. Vous pouvez à cet effet utiliser des cellules de type `Markdown` plutôt que de type `code`.

3 KNN pour la classification

Créez un nouveau notebook nommé « knn_classification.ipynb ». Dans ce notebook nous allons aborder l'utilisation d'un modèle KNN pour un problème de classification.

Pour cette première partie de TP, nous allons utiliser la base de données diabetes. On cherchera à prédire si une personne est considérée comme diabétique (colonne `Outcome`) ou non en fonction de plusieurs descripteurs.

3.1 Analyse des données

La bibliothèque pandas, permet généralement de réalisez une première analyse simple des données. À partir du *dataframe* chargé, on peut connaître :

- L'attribut `shape` donne les dimensions du *dataframe*.
- La fonction `info` propose un résumé rapide des données.
- La fonction `describe` permet d'avoir des statistiques sur différentes tendances sur les données.
- La fonction `value_counts` qui compte le nombre d'occurrences de chaque valeur.

Tâche 1 : À partir de ces fonctions, répondez aux questions suivantes :

→ Combien de classes sont présentes dans la base de données ?

→ Combien de caractéristiques descriptives de ces classes et de quels types ?

→ Combien d'exemples dans la base de données ? Et par classe ?

3.2 Préparation des données

Comme vu dans le TP précédent, lorsque l'on souhaite apprendre sur des données, on propose au minimum de découper la base de données en deux ensemble : un d'apprentissage et un de test.

Tâche 2 : Préparez les données de manière à ce que l'on puisse prédire les classes `Outcome`. Puis, en exploitant la librairie `scikit-learn`, proposez un découpage de la base de données à hauteur de $\frac{2}{3}$ apprentissage et $\frac{1}{3}$ test.

3.3 Apprentissage et validation du modèle

La librairie `scikit-learn` propose le modèle `KNeighborsClassifier` qui pourra apprendre à partir de plusieurs descripteurs. Le standard de cette librairie reste le même :

- une méthode `fit` pour apprendre à partir de données.
- une méthode `predict` pour obtenir les prédictions de notre modèle sur des données.
- une méthode `score` qui prédit et fournit directement le score de performance.

Tâche 3 : Réalisez l'apprentissage du modèle avec le paramètre `n_neighbors` fixé à 1. Vérifiez ensuite les scores de performances (précision) sur les deux ensembles apprentissage et test.

Avec un `random_state=42` pour le découpage des données, vous devriez obtenir :

```
Model (k=1): [train: 1.00, test: 0.68]
```

Un surapprentissage peut de suite être constaté. Pour analyser ce que le modèle a pu apprendre, la matrice de confusion des prédictions peut-être utilisée.

Tâche 4 : Analysez la matrice de confusion proposée par les prédictions du modèle sur la base de test. Que pouvez-vous observer ?

Tâche 5 : Testez plusieurs valeurs de paramètre `n_neighbors` et sélectionnez celle qui vous semble vous procurer le modèle le plus performant. Avec par exemple `n_neighbors` $\in [1, 20]$.

4 KNN pour la régression

Créez un nouveau notebook nommé « knn_regression.ipynb ». Dans ce notebook nous allons aborder l'utilisation d'un modèle KNN pour un problème de régression.

4.1 Analyse des données et préparation des données

Cette seconde partie de TP va exploiter la base de données housing à partir de laquelle on va chercher à prédire le prix de maisons en fonction de plusieurs descripteurs.

Tâche 6 : De la même manière que dans la première partie du TP, analysez et préparez les données.

4.2 Apprentissage et validation

On cherche à obtenir le modèle qui prédit au mieux la valeur de vente d'une maison identifiée par la colonne `MEDV`.

Tâche 7 : Réalisez l'apprentissage de votre modèle et identifiez une valeur du paramètre `n_neighbors` proposant une bonne généralisation. Pour cela, vous pouvez vous appuyez sur différentes métriques, dont certaines utilisées dans le précédent TP :

- Le coefficient of determination
- L'erreur moyenne absolue définie par : $\frac{1}{n}\sum_{i=1}^{n}|\hat{y} - y|$
- L'erreur quadratique moyenne définie par : $\frac{1}{n}\sum_{i=1}^{n}(\hat{y} - y)^2$

  avec $\hat{y}$ la valeur prédite et y la valeur attendue.

Remarque : l'erreur quadratique moyenne aura tendance à considérer plus fortement les erreurs graves car élevées au carré.

4.3 L'importance de la normalisation

Comme vu en cours, la sélection des voisins peut varier en fonction de la mise à l'échelle ou non de certains descripteurs. L'algorithme est donc sensible à la normalisation des données.

La librairie « `scikit-learn` » propose des outils rapide de mise à l'échelle des données au sein de son module « `preprocessing` ».

Tâche 8 : Utilisez le « `StandardScaler` » proposé par « `scikit-learn` » pour normaliser les données d'entrée de la base de données. Vérifiez ensuite les performances du modèle.

Bonus

Proposez une nouvelle version de votre modèle KNN de classification précédent. Dans cette nouvelle version vous allez exploiter la même procédure de prétraitement des données et vérifier si les résultats sont améliorés ou non.

5 Résumé

→ KNN est un algorithme simple à mettre en place et facile à comprendre.

→ Il peut traiter plusieurs descripteurs en entrée pour émettre une notion de distance et identifier ses voisins.

→ Il peut traiter à la fois des problèmes de classification et de regression.

→ Il est important d'avoir des données normalisées pour cet algorithme, amenant de meilleure performance.

→ Dans le cadre d'une classification binaire, la matrice de confusion permet d'identifier visuellement les erreurs.

6 Remise des travaux

N'oubliez pas de réaliser un commit de vos travaux. Taguez également votre projet avec le tag « `tp2` » et soumettez-le sur le serveur Gitlab. Il fera office de rendu.