Piscine Machine Learning | Jour 2

Méthode des K plus proches voisins

Nom de repo: ml_d02

Droits: florian.bacho@epitech.eu, voravo_d, girard_z

Langage: Python

Introduction

Aujourd'hui nous allons voir un premier exemple d'apprentissage supervisé: la méthode des K plus proches voisins. Cette méthode, très simple à comprendre et à implémenter, va vous permettre notamment de résoudre des problèmes de classification.

« La classification est la catégorisation algorithmique d'objets. Elle consiste à attribuer une classe ou catégorie à chaque objet à classer. »

En utilisant un dataset contenant un certain nombre d'exemples avec des attributs (les entrées) et une classe/catégorie (les sorties), vous allez faire un programme capable de décider quelle est la classe d'un nouvel exemple. La méthode étant très intuitive, nous allons voir tout ça directement dans des exercices.

Exercice 00 :

Fichier: ex00 .py

En utilisant le DataLoader de la journée 1, charger le dataset « ex00.ds », et afficher sur un repère 2D l'ensemble des exemples sous forme de points, avec en x leur premier attribut et en y leur second. Les points devront être affichés en rouges si la catégorie est 0, et en bleus si elle est 1.

Exercice 01 :

Fichier: ex01.py

Afficher à nouveau le dataset à la manière de l'exercice 00 en rajoutant le point suivant en vert (représentant une classe/catégorie inconnue):

x = 0.4, y = 0.4

A votre avis, le point appartient-il à la catégorie 0 ou à la catégorie 1 ? Pourquoi ? (répondez en commentaire dans le .py)

Exercice 02 :

Fichier : ex02.py

Une manière de répondre à la question précédente était de dire que le nouveau point appartenait à la catégorie 1 car il était tout simplement plus proche des points rouges que des points bleus. C'est exactement le principe de l'algorithme des K plus proches voisins. Pour attribuer une classe à un nouvel exemple, nous calculons la distance entre ce point et tous les points de notre dataset d'entraînement, en retenant les 1,2, …, K plus proches.

Comment définir la distance entre 2 exemples ? Il existe de nombreuses fonctions de distance différentes, mais nous allons utiliser la distance Euclidienne:

Soit deux points p = $(p_1, p_2, ..., p_n)$ et q = $(q_1, q_2, ..., q_n)$, on note alors leur distance d(p, q) où :

d(p, q) = $\sqrt{((q_1 - p_1)^2 + (q_2 - p_2)^2 + ... + (q_n - p_n)^2)}$ = $\sqrt{(\sum_{i=1}^{n} (q_i - p_i)^2)}$

Créer une fonction *computeDistance* qui prend en paramètres 2 exemples et retourne leur distance euclidienne.

Exercice 03 :

Fichier: ex03.py

Pourquoi faire varier K ?

Revenons au « K » de la méthode. Il définit combien de voisins les plus proches nous devons chercher. Si K = 1, nous attribuons à notre nouvel exemple la classe du voisin le plus proche.

Si K > 1, nous comptons parmi ces K plus proches voisins le nombre d'exemples appartenant à chaque catégorie, et nous attribuons à notre nouvel exemple la classe la plus présente.

Quel est donc l'intérêt d'utiliser différents K ? Pourquoi ne pas par exemple toujours sélectionner un K = 1 ?

A la manière de l'exercice 2, afficher les points du dataset « ex03.ds » en ajoutant à nouveau en vert le point x = 0.4, y = 0.4.

Quelle serait la catégorie assignée au point vert pour K = 1 ? Pour K = 3 ? (Ne calculez pas de distances, faîtes le visuellement et répondez en commentaires dans le .py).

Exercice 04 :

Fichier: ex04.py

Choisir K

L'exercice 3 nous a fait voir que quand K est petit (ici = 1), le résultat pour un exemple peut facilement/fortement varier lors d'un petit changement dans notre dataset. En effet, nous avons seulement ajouté 1 exemple dans le dataset ex03.ds et la catégorie attribuée est devenue 1 (bleu) alors que le point était toujours entouré d'une grande majorité de 0 (rouge).

Il est donc important dans la méthode des K plus proches voisins de choisir un K qui nous donne des résultats stables. Il n'existe pas de règle universelle pour choisir K, il faut faire des tests pour comprendre/voir ce qui fonctionne le mieux. Retenez cependant que:

1) Lorsque K est trop petit, la méthode est instable et que des petites variations dans le dataset peuvent engendrer de grandes variations dans la classification.

2) Lorsque K est trop grand, la méthode choisira simplement la classe la plus probable/ la plus présente dans le dataset.

Créer une fonction *getNearestNeighbors* qui retourne la liste des K voisins les plus proches. Cette fonction prendra en paramètre le nouvel exemple, le dataset d'entraînement, et K.

Exercice 05 :

Fichier: ex05.py

Faire une fonction *getMostPresentCategory* qui retourne la catégorie la plus présente dans la liste retournée par *getNearestNeighbors*.

Comment gérer les égalités?

Il est en effet possible d'obtenir dans votre liste un nombre égal de plusieurs différentes catégories.

Vous êtes libres de décider comment gérer un tel cas. Cependant voici quelques idées:

1) Choisir des K qui ne sont pas des multiples du nombre de catégories possibles, afin de d'éviter/limiter ce genre de situation. (Ex : Pour 2 catégories possibles, choisir K impaire).

2) Retourner la classe du voisin le plus proche.

3) Retourner aléatoirement une des classes les plus présentes.

Exercice 06 :

Fichier: ex06.py

Implémenter la méthode des K plus proches voisins avec comme dataset d'entraînement trainWine.ds » en combinant les fonctions réalisées précédemment. Il s'agit d'un dataset contenant les caractéristiques chimiques de vins (entrées) produits dans une même région d'Italie mais dérivés de différents cépages (sorties).

Le dataset est déjà normalisé donc vous n'avez pas à le faire vous-même. Mais si vous n'avez pas eu le temps de voir cette notion hier nous vous conseillons de le faire maintenant, et également de lire la conclusion de cette journée.

Caractéristiques (pour les curieux): Alcohol, malic acid, ash, alcalinity of ash, magnesium, total phenols, flavanoids, nonflavanoid phenols, proanthocyanins, color intensity, hue, OD280/OD315 of diluted wines, proline.

Tester les prédictions de votre algorithme avec le dataset de test « testWine.ds » avec différents K, afin de voir lequel fonctionne le mieux. (Un affichage graphique serait plus intéressant que des outputs sur un terminal).

Conclusion :

Si vous avez réussi tous les exercices de la journée, félicitations! Vous venez d'implémenter votre premier algorithme d'apprentissage supervisé. Vous venez de créer un programme capable de classifier différents types de vins sans nécessiter de connaissances en vin, ou de formules et conditions explicites (ex : « if alcohol > … then … »).

Nous avons parlé hier de l'importance de la normalisation de données, et qu'en son absence les algorithmes pouvaient mal fonctionner. Nous pouvons donner un début d'explication avec l'exercice 06. Imaginez que les caractéristiques des vins n'avaient pas été normalisées. Nous aurions calculé la distance euclidienne avec certaines valeurs qui oscillent entre 0 et 0.5, et d'autres qui vont par exemple de 400 à 1200. La valeur de la distance aurait donc été très majoritairement décidée par les caractéristiques ayant de grandes valeurs, et très peu/pas du tout par les caractéristiques ayant de très petites valeurs. Étant donné que l'algorithme est entièrement basé sur cette notion de distance, nous aurions finalement « ignoré » un bon nombre des caractéristiques disponibles, et le résultat final aurait sûrement été beaucoup moins performant (intuitivement, il est clair qu'il est plus dur de classifier un objet avec 2/3 caractéristiques qu'avec 12/13).

Chaque dataset fourni pendant le reste de cette piscine sera déjà normalisé, cependant n'oubliez pas cette notion si vous décidez de vous lancer dans un projet de machine learning de votre côté.

Pour résumer l'algorithme des K plus proches voisins:

Avantages :

Simple à comprendre/visualiser

Simple à implémenter

Performant

Inconvénients :

Très coûteux en temps lorsqu'on a beaucoup de données/caractéristiques

Sensible aux variations / données aberrantes

Paramètres à régler/tester :

Fonction de distance

K