Apprentissage automatique
Méthode des k plus proches voisins

Chap. 22,01

1 Introduction

L'*apprentissage automatique* (*machine learning*, *analyse prédictive*, *apprentissage statistique*) a pour objectif de faire émerger de la « connaissance » à partir de l'étude de données. C'est un champ de recherche qui se situe à l'intersection des statistiques, de l'intelligence artificielle et de l'informatique. Ces applications sont vastes : recommandations sur la musique que vous devriez écouter en fonction de vos goûts, sur les films que vous devriez regarder mais aussi la reconnaissance de visage dans une photo, d'une tumeur cancéreuse, ... et dans le domaine des sciences, la découverte de nouvelles planètes, de nouvelles particules, l'analyse de séquences d'A.D.N., ...

2 Intérêt de l'apprentissage automatique : cas d'un filtre antispam

Les premiers filtre anti-spam nécessitaient la création de listes « noires » dans lesquelles on énumérait les adresses mails ou les mots clés qui caractérisaient généralement les messages à déclarer comme « spam ». L'application analysait ensuite chaque mail reçu en vérifiant s'il relevait ou pas des règles établies.

Les règles décisionnelles étaient essentiellement basées sur des raisonnements du type « si... sinon... » et nécessitaient l'intervention d'*humains ayant une très bonne compréhension du phénomène étudié*.

Les filtres anti-spam actuels prennent leurs décisions *après qu'on leur a fait analyser un grand nombre de messages et après qu'ils en ont déduit automatiquement les règles nécessaires à la détection de messages frauduleux*.

En résumé,

- Concevoir toutes les règles que doit suivre un programme nécessite une compréhension approfondie de la manière dont une décision devrait être prise par un expert humain qui réaliserait la tâche ;
- Il doit être possible de d'adapter ces règles au traitement informatique[1] ;
- Généralement, la logique de la prise de décision est spécifique à un programme (ou domaine). La modifier oblige à modifier en profondeur ce programme et il est généralement difficile d'étendre cette logique à d'autres domaines.

L'idée d'apprentissage automatique ne date pas d'hier, puisque le terme de « machine learning » a été utilisé pour la première fois par l'informaticien américain Arthur Samuel en 1959. Les algorithmes d'apprentissage automatique ont connu un fort regain d'intérêt au début des années 2000 notamment grâce à la quantité de données disponibles sur internet.

1. L'une des difficultés de la reconnaissance automatique des visages dans les images est le fait que les humains et les ordinateurs n'ont pas la même perception des images. Il est donc difficile de définir un ensemble de règles qui permettraient à un ordinateur de différencier un visage d'un bouquet de fleurs.

3 Deux familles d'apprentissage automatique

Deux familles d'algorithmes d'apprentissage automatique existent :

L'apprentissage supervisé. On fait analyser par l'algorithme un grand nombre de couples *entrée/sortie désirée*. Cet algorithme trouve alors une manière de produire la sortie désirée à partir d'une certaine entrée qu'il n'a jamais rencontrée auparavant.

Par exemple, pour le cas du filtre anti-spam, on fait étudier, par l'algorithme, un grand nombre de messages (*entrée*) en précisant à chaque fois s'il s'agit d'un message à conserver ou d'un spam (*sortie*). À l'issue de cette phase d'analyse, l'algorithme est capable de prédire si un message est un ou non un spam.

On parle « d'apprentissage supervisé » car tout se passe comme si un « professeur » supervisait l'apprentissage en expliquant le lien entre les entrées et les sorties désirées.
La création du jeu de données initial[2] est *souvent un processus manuel laborieux*, mais ces algorithmes sont bien compris, ont des performances faciles à évaluer et sont, au final, les plus efficaces.

On retrouve des algorithmes supervisés dans l'identification d'un code postal par la poste (depuis très longtemps), dans la détection de tumeurs malignes en imagerie médicale, dans la détection d'activités frauduleuses dans les transactions bancaires faites avec des cartes de crédit, ...

Algorithmes non supervisés. On fait analyser par l'algorithme un grand nombre d'*entrées* mais *aucune sortie connue n'est fournie*.

De nombreuses applications utilisent ces algorithmes mais ils restent plus difficiles à comprendre et à évaluer.

On retrouve des algorithmes non supervisés dans l'identification des thèmes les plus traités dans les blogs ou les forums, dans la recherche des connexions possibles entre profils différents, dans la détection d'accès anormaux à des sites Web, ...

4 K plus proches voisins

4.1 Idée de base

Imaginons que l'on souhaite déterminer pour qui une personne va voter aux prochaines élections présidentielles. Si on ne sait rien de son orientation politique, on peut demander à ses voisins pour qui, eux, vont voter. On peut considérer que puisque toutes ces personnes habitent dans le même quartier, il existe une connexion entre leurs votes et donc qu'une prédiction est possible.

On imagine maintenant que non seulement on interroge les voisins mais aussi qu'on considère en plus le métier qu'exerce la personne, la superficie de son appartement, le nombre d'enfants qu'il a, l'endroit où il passe ses vacances, ... On augmente alors le nombre d'informations et on améliore ainsi sûrement la prédiction initiale. C'est l'idée qui sous-tend la classification par « plus proches voisins ».

4.2 Le modèle

Le modèle des plus proches voisins est un des modèles prédictifs les plus simples. *Il ne nécessite pas de phase d'apprentissage à proprement parler, il faut juste stocker le jeu de données d'apprentissage.* À partir d'un **ensemble E de données labellisées**, il permet de **classer** (*déterminer le label*) **une nouvelle donné**e (donnée n'appartenant pas à E).

2. Ce n'est pas toujours possible d'une créer un !

Énoncé général.

Soit un ensemble E contenant n données (échantillons) labellisées : $E=\{(y_i, \vec{x_i})\}$ avec i compris entre 1 et n, où y_i correspond à la **classe** (le label) de la donnée i et où le vecteur $\vec{x_i}$ de dimension p ($\vec{x_i}=(x_{1i}, x_{2i}, ..., x_{pi})$) représente les **variables prédictrices** de la donnée i. Soit une donnée u qui n'appartient pas à E et qui *ne possède pas de label* (u est uniquement caractérisée par un vecteur $\vec{x_u}$ de dimension p). *Soit d une fonction qui renvoie la distance entre la donnée u et une donnée quelconque appartenant à E. Soit un entier k inférieur ou égal à n.* Voici le principe de l'algorithme de k plus proches voisins :

- On calcule les distances entre la donnée u et chaque donnée appartenant à E à l'aide de la fonction d ;
- On retient les k données du jeu de données E les plus proches de u ;
- On attribue à u la classe qui est la plus fréquente parmi les k données les plus proches.

Traduction de l'énoncé dans l'exemple de l'orientation politique.

On possède un ensemble E de n échantillons comportants les p caractéristiques : quartier, superficie appartement, métier, nombre enfants, ... auxquels sont attachés les labels : républicain, socialiste, ... *On cherche à classifier un échantillon u n'appartenant pas à E.*
On transforme toutes les caractéristiques en grandeurs numériques. Chaque échantillon est alors représentable par un point dans un espace à p dimensions. On établit finalement la liste des k plus proches voisins de l'échantillon u dans cet espace à p dimensions. La classe de u est la classe la plus fréquente dans cette liste.

1) L'algorithme des plus proches voisins est-il un algorithme supervisé ou non supervisé ?

4.3 Iris de Fischer (https://fr.wikipedia.org/wiki/Iris_de_Fisher)

On suppose qu'un botaniste souhaite pouvoir distinguer diverses espèces d'iris. Il a réalisé un certain nombre de mesures associées à chaque iris : la longueur et la largeur des pétales, les longueur et la largeur des sépales. Toutes ces données sont exprimées en centimètres.

Pour classifier chacun des iris, le botaniste dispose d'autres mesures précédemment classifiées par un expert : setosa, versicolor ou virginica. Les iris que le botaniste a prélevé appartiennent forcément à l'une de ces trois espèces (classes).

- Télécharger le fichier `iris.csv`.
- Le corrigé de ce TP se trouve à cette adresse : https://repl.it/@dlatreyte/iris01

2) Importer les modules `pandas` avec l'alias `pd` et `Matplotlib.pyplot` avec l'alias `plt`.

3) Depuis la fonction `main`, charger le fichier `iris.csv` et observer son contenu.

```
iris = pd.read_csv("./iris.csv", sep=";", header=0)
print(iris)
```

et ses caractéristiques :

```
print(iris.info())
```

4) Combien d'échantillons comporte ce documents ? Combien de caractéristiques possède chaque échantillon ?

5) On se limite, pour débuter à deux caractéristiques que l'on transforment en listes Python. Toujours dans la fonction `main`, entrer les lignes de code suivantes :

```
x = iris.loc[:, 'Longueur pétale (cm)'].tolist()
y = iris.loc[:, 'Largeur pétale (cm)'].tolist()
classe = iris.loc[:, 'Classe'].tolist()
```

6) Définir la fonction `creation_echan` dont la spécification est :

```
def creation_echan(x, y, classe):
    """
    À partir des listes des coordonnées et des étiquettes des échantillons,
    création d'une liste de dictionnaires d'échantillons.

    Paramètres
    ----------
    x : List[float]
        Liste des abscisses des points de l'échantillon de test.
    y : List[float]
        Liste des ordonnées des points de l'échantillon de test.
    classe : List[str]
        Liste des étiquettes de l'échantillon de test.

    Retour
    ------
    echantillons : List[Dict]
        Liste des dictionnaires représentant les échantillons.
        Chacun des échantillons est un dictionnaire possédant les clés
        x, y, indice, classe.
    """
```

7) Depuis la fonction `main`, appeler la fonction `creation_echan` :

```
echan_modeles = creation_echan(x, y, classe)
```

8) Le botaniste mesure les caractéristiques de l'iris qu'il a prélevé : largeur pétale = 0,75 cm ; longueur pétale = 2,5 cm. On définit donc le dictionnaire de l'échantillon test, dans la fonction `main` :

```
echan_test = {'x' : 2.5, 'y' : 0.75}
```

9) Définir la fonction `affichage` suivante :

```
def affichage(liste_echan, echan_test, titre, nom_fichier):
    """
    Construction des graphiques

    Paramètres
    ----------
```

```
    liste_echan : List[Dict]
        Liste de dictionnaires. Chaque dictionnaire représente un échantillon
        et possède les clés x, y, classe, indice.
    echan_test : Dict
        Dictionnaire qui représente un échantillon. Possède les clés x et y.
    titre : str
        Titre du graphique
    nom_fichier : str
        Nom du fichier image

    Retour
    ------
    None
    """
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    ax.set_title(titre)
    ax.set_xlabel("Longueur pétale (cm)")
    ax.set_ylabel("Largeur pétale (cm)")

    for echan in liste_echan:  # echan est un dictionnaire
        x = echan['x']
        y = echan['y']

        if echan['classe'] == 'Iris-setosa':
            couleur = 'g'
        elif echan['classe'] == 'Iris-versicolor':
            couleur = 'r'
        else:
            couleur = 'b'
        ax.scatter(x, y, color=couleur)

    ax.scatter(echan_test['x'], echan_test['y'], color='black')

    ax.grid(True)
    #plt.show()  # Sauf sur repl.it
    plt.savefig(nom_fichier)  # Sauvegarde
```

et l'appeler depuis la fonction `main`.

10) Définir la fonction `calculs_distances` dont la spécification est :

```
def calculs_distances(liste_echan, echan_test):
    """
    Calcule la distance entre chaque point de l'échantillon modèle et
    l'échantillon étudié. Ajoute cette distance à chaque dictionnaire de la
    liste echan_modeles.
    Retourne la liste echan_modeles.

    Paramètres
    ----------
```

```
    liste_echan : List[Dict]
        Liste de dictionnaires. Chaque dictionnaire représente un échantillon
        et possède les clés x, y, classe, indice.

    echan_test : Dict
        Dictionnaire qui représente un échantillon. Possède les clés x et y.

    Retour
    ------
    liste_echan : List[Dict]
        Liste de dictionnaires. Chaque dictionnaire représente un échantillon
        et possède les clés x, y, classe, indice, distance.
    """
```

11) Appeler la fonction `calculs_distance` depuis la fonction `main` :

```
echan_modeles_avec_distances = calculs_distances(echan_modeles, echan_test)
```

12) Réfléchir à la définition d'une fonction `k_plus_proches_voisins` dont la spécification serait :

```
def k_plus_proches_voisins(echan_modeles_avec_distances, k):
    """
    Construit la liste des k échantillons plus proches voisins.

    Paramètres
    ----------
    echan_modeles_avec_distances : List[Dict]
        Liste de dictionnaires. Chaque dictionnaire représente un échantillon
        et possède les clés x, y, classe, indice, distance.
    k : int
        Nombre de plus proches voisins à prendre en compte.

    Retour
    ------
    plus_proches_voisins : List[Dict]
        Liste de longueur k, composée des dictionnaires représentant les k
        échantillons les plus proches de l'échantillon testé.
    """
```

L'écriture de cette fonction est délicate, c'est la raison pour laquelle je vous conseille de récupérer le code dans le corrigé, en ligne.

13) Appeler la fonction `k_plus_proches_voisins` depuis la fonction `main` :

```
k = 10
voisins = k_plus_proches_voisins(echan_modeles_avec_distances, k)
```

et les afficher à l'aide de la fonction `affichage` :

```
affichage(voisins, echan_test, "{} plus proches voisins".format(k),
          "Voisins_proches.svg")
```

14) Définir la fonction `determination_classe` dont la spécification est la suivante :

```
def determination_classe(liste_voisins_plus_proches):
    """
    Détermine la classe probable de l'échantillon testé. Cette classe est
    celle qui apparait le plus souvent dans la liste des plus proches
    voisins.

    Paramètre
    ---------
    liste_voisins_plus_proches : List[Dict]
        Liste des dictionnaires représentant les voisins les plus proches.

    Retour
    ------
    classe_test : str
        Classe probable de l'échantillon testé.
    """
```

15) Appeler la fonction `determination_classe` depuis la fonction `main` :

```
classe = determination_classe(voisins)
print("Classe probable de l'échantillon testé : {}".format(classe))
```

5 Iris de Fischer (avec plus de paramètres)

Reprendre le code précédent de façon à ce qu'il prenne en compte davantage de paramètres du fichier. Une version à trois dimensions est accessible à cette adresse : https://repl.it/@dlatreyte/iris02.

Remarque. Pour un affichage dans un espace à trois dimensions, débuter le fichier par

```
from mpl_toolkits.mplot3d import Axes3D
```

et modifier la fonction d'affichage de la sorte :

```
def affichage(liste_echan, iris, titre, nom_fichier):
    """
    Construction des graphiques

    Paramètres
    ----------
    liste_echan : List[Dict]
        Liste de dictionnaires. Chaque dictionnaire représente un échantillon
        et possède les clés x, y, classe, indice.
    iris : Dict
        Dictionnaire qui représente un échantillon. Possède les clés x et y.
    titre : str
        Titre du graphique
    nom_fichier : str
        Nom du fichier image
```

```
    Retour
    ------
    None
    """
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1, projection='3d')
    ax.set_title(titre)
    ax.set_xlabel("Longueur pétale (cm)")
    ax.set_ylabel("Largeur pétale (cm)")
    ax.set_zlabel("Longueur sépale (cm)")

    for echan in liste_echan:  # echan est un dictionnaire
        x = echan['x']
        y = echan['y']
        z = echan['z']

        if echan['classe'] == 'Iris-setosa':
            couleur = 'g'
        elif echan['classe'] == 'Iris-versicolor':
            couleur = 'r'
        else:
            couleur = 'b'
        ax.scatter(x, y, z, color=couleur)

    ax.scatter(iris['x'], iris['y'], iris['z'], color='black')

    ax.grid(True)
    #plt.show()  # Sauf sur repl.it
    plt.savefig(nom_fichier)  # Sauvegarde
```