TP #3

Initiation à l'apprentissage automatique sous Python

L'apprentissage automatique, *machine learning* en anglais, est une discipline issue de l'intelligence artificielle qui repose sur l'utilisation des mathématiques, et en particulier des statistiques, afin de permettre aux ordinateurs de résoudre des tâches sans que celles-ci soient explicitement programmées. D'un point de vue fonctionnel, l'apprentissage automatique est un ensemble de techniques qui ont pour but d'automatiser les prises de décision en créant des systèmes en capacité de découvrir, d'identifier et de classifier des motifs récurrents à partir d'un jeu de données fourni.

Principe de l'apprentissage supervisé

L'apprentissage supervisé s'appuie sur un jeu de données étiquetées, c'est-à-dire distinguées par un identifiant, pour s'entraîner à produire des résultats conformes. Le principe de l'apprentissage supervisé consiste à surveiller l'apprentissage de la machine en lui présentant des exemples de ce qu'elle doit effectuer.

Les utilisations de l'apprentissage supervisé sont nombreuses. Il peut s'agir, par exemple, de reconnaissance vocale ou d'un traîtement destiné à la catégorisation de données fournies.

Notions de régression et de classification

Il existe deux grands objectifs à l'apprentissage automatique. Il s'agit de la régression et de la classification.

La régression correspond à un algorithme dont l'exécution aboutit à un résultat sous la forme d'une valeur continue. Par exemple, un prix, une température ou toute autre grandeur physique.

La classification relève d'un algorithme dont l'exécution produit un résultat sous la forme d'une valeur discrète qui permet de trier un ensemble de données selon plusieurs catégories. Par exemple, une valeur boléenne, une valeur entière correspondant à un type de donnée mais aussi un identifiant permettant de distinguer des données.

Certains algorithmes peuvent permettre d'effectuer une régression ou une classification.

Présentation du jeu de données

Le jeu de données fourni est consigné dans un fichier au format `.csv`. Il est nommé `diabetes.csv` et concerne des données médicales issues d'analyses permettant de diagnostiquer une pathologie liée au diabète. Les différentes champs du fichier sont ordonnés de la façon suivante :

— `Pregnancies`
— `Glucose`
— `BloodPressure`
— `SkinThickness`
— `Insulin`
— `BMI`
— `DiabetesPedigreeFunction`
— `Age`
— `Outcome`

Le libellé des différents champs correspond à la première ligne du fichier `diabetes.csv`.

Les valeurs numériques des différents champs correspondent à des colonnes et elles sont séparées par une virgule. Chaque ligne du fichier de données représente un enregistrement.

L'étiquette `Outcome` indique la situation du patient au regard du diabète. La valeur `1` signifie que le patient est atteint du diabète alors que la valeur `0` marque le fait qu'il ne l'est pas.

Code de la fonction d'import et d'affichage du jeu de données

Le programme ci-dessous permet d'importer les données issues du fichier `diabetes.csv` :

```
def import_csv(filename):
  with open(filename, 'r') as f:
    data = []
    headers = f.readline().split(',')
    for line in f:
      words = line.split(',')
      data.append([float(w) for w in words])
  ndata = np.array(data)
  return headers, ndata
```

La fonction `import_csv()` renvoie deux variables `headers` et `ndata` correspondant respectivement à l'entête du fichier sous la forme d'une liste de chaînes de caractères ainsi qu'à un tableau Numpy contenant les données converties en type `float`.

Le résultat de l'exécution de la fonction `import_csv()` permet ainsi de récupérer le libellé de chacun des champs et l'ensemble des valeurs associées qui peuvent ensuite être affichés.

Code de la fonction d'analyse simple des données

Le code de la fonction `describe_data()` suivant s'appuie sur l'importation de données obtenue par le programme de la section précédente :

```
def describe_data(data, header, labeled=True):
  stop = 0
  if labeled:
    stop = data.shape[1] - 1
  else:
    stop = data.shape[1]
  for i in range(stop):
    print(header[i] + " --> ", end="")
    print(f"Average : {np.mean(data[:, i]):3.2f}", end=", ")
    print(f"Std Dev : {np.std(data[:, i]):3.2f}", end=", ")
    print(f"Min : {np.min(data[:, i]):3.2f}", end=", ")
    print(f"Max : {np.max(data[:, i]):3.2f}")
```

Les paramètres de la fonction sont les variables `headers` et `ndata`. La valeur de l'argument `labeled` est forcé à la valeur `True` afin de marquer le fait que les enregistrements sont étiquetés.

Code de la fonction de normalisation des données

Les données récupérées par la fonction `describe_data()` ne sont pas normalisées. Il est cependant nécessaires qu'elles le soient avant d'être traitées par un algorithme d'apprentissage automatique

Le code de la fonction `normalize` présenté ci-dessous permet de normaliser les données à des valeurs réelles étalées entre 0 et 1 :

```
def normalize(data, labeled=True):
  if labeled:
    e_data = data[:, :-1] / data[:, :-1].max(axis=0)
    labels = np.reshape(data[:, -1], (e_data.shape[0], 1))
    return np.concatenate((e_data, labels), axis=1)
  else:
    return data / data.max(axis=0)

header, data = import_csv('diabetes.csv')
data = normalize(data)
describe_data(data, header)
```

Code d'utilisation des fonctions

Le programme ci-après concerne l'enchaînement des instructions permettant d'exploiter les données contenues dans le fichier `diabetes.csv` :

```
header, data = import_csv('diabetes.csv')
data = normalize(data)
describe_data(data, header)
```

Implémentation de la méthode des k plus proches voisins avec Numpy

La méthode des k plus proches voisins, *k-nearest neighbors (KNN)* en anglais, est avant tout un algorithme d'apprentissage supervisé utilisé pour la classification. Elle s'applique ainsi à des données comportant des étiquettes discrètes. Il s'applique néanmoins aussi à la régression basée sur des données avec des étiquettes continues.

La figure 1 schématise le principe de l'algorithme KNN.

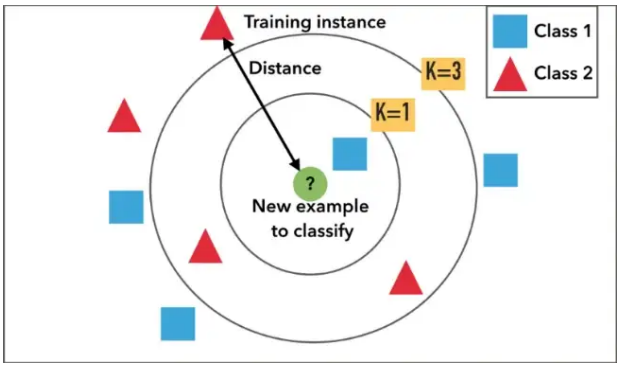

Fig. 1 – *Principe de l'algorithme KNN*

La démarche globale est la suivante :

1. Calcul de la distance Euclidienne
2. Identification du plus proche voisin
3. Identification des k plus proches voisins
4. Établissement des prédictions

Travail à réaliser

1. Implémenter la fonction de prototype `euclidean_distance(p1, p2)` où `p1` et `p2` sont des vecteurs de données numériques de dimension quelconque. Cette fonction renvoie la distance euclidienne entre les deux points.
2. Écrire une fonction de prototype `knn(data, x, k, d)` où `data` est le tableau Numpy de données, `x` est l'élément x qu'on cherche à classifier, `k` est l'entier k de l'algorithme KNN et d'une fonction calculant la distance entre deux points. Cette fonction implémente l'algorithme KNN et renvoie la classe de x.
3. ...