Classification supervisée - TP1

Rémy Degenne

19 Janvier 2022

1 Mise en place

Création d'un dossier pour le travail:

- Créer un dossier pour le cours de SD3, et un sous-dossier pour le TP1.
- Ouvrir un terminal
- Se rendre au dossier créé avec la commande `cd`

Création d'un notebook:

- Dans le terminal, taper `jupyter notebook`
- Une page a dû s'ouvrir dans un navigateur
- Sur cette page, cliquer sur "new" et choisir "python" (ou "python 2"). Une nouvelle page s'ouvre. C'est le notebook dans lequel on va travailler.
- Donner un nom au notebook en cliquant sur "Untitled" en haut de la page.

2 Documentation

En cas de doute sur la manière d'utiliser une fonction, l'idéal est d'aller lire le manuel.

Documentation de scikit-learn: `https://scikit-learn.org/0.24/index.html`

Documentation de matplotlib: `https://matplotlib.org/stable/tutorials/introductory/usage.html`

Documentation de numpy: `https://numpy.org/doc/1.18/`

3 Imports

Dans la première cellule, on va importer une liste de "packages" dont on aura besoin:

```
# Imports
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets
```

Explication de la première ligne: les fonctions définies dans la librairie `numpy` sont maintenant disponibles et on donne le nom court `np` à cette librairie. Par example la fonction `numpy.sum` peut maintenant être utilisée en tapant `np.sum`.

Explication de la troisième ligne: on importe juste `ListedColormap` de `matplotlib.colors`. On peut l'appeler en tapant `ListedColormap`.

4 Dataset: Iris

On va commencer par rassembler nos données. On utilisera dans ce TP le dataset (ensemble de données) Iris. Dans une cellule vide en dessous des imports, écrire:

```
iris = datasets.load_iris()
```

On peut maintenant écrire `print(iris)` pour voir ce que contient l'objet `iris` que l'on vient de définir. C'est un dictionnaire. On peut obtenir la liste des champs de ce dictionnaire en tapant `print(iris.keys())`. Ceux qui nous intéressent sont `data`, `target` et `target_names`. `data` contient les données et `target` contient leur classe.

- Combien de données sont présentes dans `iris` ?
- Combien d'attributs ont ces données ?
- Combien de classes (ou étiquettes) différentes ?

Pour ce TP, on ne va utiliser que les deux premiers attributs des données. Dans une cellule libre à la fin du notebook, écrire:

```
X = iris.data[:, :2] # X est un tableau qui contient toutes les lignes de iris.data, et
    les colonnes 0 et 1.
Y = iris.target # y contient les etiquettes
```

`X` contient maintenant nos données, qui ont chacune deux attributs. On se restreint à deux attributs pour pouvoir faire des graphiques plus facilement. Mais idéalement, on voudra ensuite utiliser tous les attributs disponibles.

Le code suivant permet d'afficher toutes nos données:

```
# Creer des "color maps", qui permettent de dire quelle couleur on veut donner au graphique
cmap_light = ListedColormap(['orange', 'cyan', 'cornflowerblue'])
cmap_bold = ['darkorange', 'c', 'darkblue']

# On cree une "figure", qui est le cadre dans lequel on mettra le graphique
plt.figure(figsize=(8, 6))

# L'objet col contient 'k' 'r' ou 'b' en fonction de la classe du point, qui sont des
    codes couleurs
col = np.where(iris.target_names[Y]=='setosa','r',
    np.where(iris.target_names[Y]=='versicolor','b','k'))

# On affiche un graphique "scatter", c'est a dire avec un point pour chaque donnee
```

```
plt.scatter(x=X[:, 0], y=X[:, 1], c=col)
plt.show()
```

On a obtenu une représentation graphique de nos données, où les coordonnées des points sont les attributs et la couleur représente l'étiquette.

5 Classeur: k plus proches voisins

Le but du tp est d'expérimenter avec le classeur des k plus proches voisins (k-NN) et de l'évaluer.

Le principe du classeur k-NN est le suivant:

- Construction: stocker tous les exemples.
- Evaluation: étant donné une nouvelle donnée x,
  - Calculer la distance d_i de x à x_i pour tout i de 1 à N (nombre de données)
  - Trouver les k données x_i les plus proches de x
  - Retourner l'étiquette majoritaire parmi les k plus proches voisins

5.1 Création du classeur

La librairie scikit-learn propose une implémentation de k-NN: la fonction `neighbors.KNeighborsClassifier` .

On peut définir un classeur:

```
k = 15
clf = neighbors.KNeighborsClassifier(k, weights='uniform', algorithm='brute')
```

Pour constuire le classeur, on utilise sa fonction `fit`:

```
clf.fit(X, Y) # X contient les donnees, Y contient les etiquettes
```

Pour classer de nouvelles données, on utilise la fonction `predict`, qui prend un tableau de données en entrée. Pour une unique donnée `x`, on écrit

```
clf.predict([x])
```

Pour évaluer la qualité de la classification obtenue, on peut compter le nombre d'erreurs.

```
errors = np.zeros_like(Y) # un tableau rempli de zeros de la meme taille qu'Y
for i, (x, y) in enumerate(zip(X, Y)): # i est l'indice, x la donnee numero i et y
    l'etiquette numero i
    if clf.predict([x]) != y: # si l'etiquette obtenue est differente de y
        errors_test[i] = 1
print(np.sum(errors_test)/len(errors_test)) # affiche la proportion d'erreurs
```

- Faire varier k et calculer la proportion d'erreurs pour chaque valeur.

- L'argument `weights` de `neighbors.KNeighborsClassifier` permet de donner un poids différent aux données en fonction de leur distance à la nouvelle donnée pour laquelle on veut faire une prédiction. Essayez `weights='distance'` pour utiliser un poids qui décroit avec la distance.

5.2 Evaluer un classeur

Nous ne sommes pas intéressés par la performance d'un classeur sur les données qui ont servi à le construire. On veut un classeur qui *généralise*, c'est à dire un classeur qui permet de bien classer des données qu'il n'a encore jamais vues.

Pour construire puis évaluer un classeur, on sépare les données en deux groupes: les données d'apprentissage et les données de test. Les données d'apprentissage sont utilisées pour construire le classeur et les données de test sont utilisées pour l'évaluer.

On va mélanger les données puis les découper en deux ensembles:

```
def unison_shuffled_copies(a, b): # melange deux tableaux a et b de la meme maniere: si le
    premier element de a est envoye en position n, le premier element de b est aussi en
    position n
    assert len(a) == len(b)
    p = np.random.permutation(len(a))
    return a[p], b[p]

np.random.seed(1)
X_rand, Y_rand = unison_shuffled_copies(iris.data, iris.target) # on melange les donnees

# On extrait des donnees d'apprentissage (train) et de test
X_train = X_rand[:120, :2]
Y_train = Y_rand[:120]
X_test = X_rand[120:, :2]
Y_test = Y_rand[120:]
```

On a maintenant des donnees `X_train` avec des étiquettes `Y_train`, qui serviront à l'apprentissage, et des données `X_test` avec étiquettes `Y_test` qui serviront à l'évaluation.

- Construire un classeur à partir des données d'apprentissage.
- Compter le nombre d'erreurs de ce classeur sur les données d'apprentissage d'une part, et sur les données de test d'autre part.
- Répéter l'opération pour plusieurs valeurs de k, puis créer un graphe avec deux courbes, une pour le nombre d'erreurs sur (`X_train`, `Y_train`) et l'autre pour le nombre d'erreurs sur (`X_test`, `Y_test`).

On pourra utiliser la fonction `plot` de matplotlib, comme dans le code suivant:

```
ks = np.arange(1,120,2) # un tableau qui contient des valeurs de 1 a 119, par increments
    de 2
length = len(ks)
errors_train = np.zeros(length)
errors_test = np.zeros(length)
```

```
# TODO: ici, calculer les proportions d'erreur et les stocker dans les deux tableaux


plt.plot(ks, errors_train, label="train")
plt.plot(ks, errors_test, label="test")
plt.legend()
plt.show()
```

- Pour quelle valeur de k on a le moins d'erreurs sur les données d'apprentissage ? et sur les données de test ?
- Expliquer la différence.
- Que se passe-t-il quand $k = 120$?

Pour aller plus loin:

- Allez voir cette page et essayez la visualisation proposée: `https://scikit-learn.org/0.24/auto_examples/neighbors/plot_classification.html#sphx-glr-auto-examples-neighbors-plot-classif`
- Utilisez tous les attributs au lieu d'en utiliser deux.
- Implémentez k-NN vous même.