Université de Carthage	Module : Data Mining
INSAT	Sections : GL
Département Mathématiques et Informatique	Niveau : 4ème année
Année Universitaire : 2023 - 2024	Enseignante : Sana Hamdi

TP N°3 : Classification supervisée :

Apprentissage et évaluation

Objectif :

Ce TP vise à :

- La prise en main de la bibliothèque scikit-learn de Python, dédiée à l'apprentissage automatique
- Sensibilisation à l'évaluation des modèles appris en classification supervisée.

I- Jeux de données :

1. Iris :

Iris est un ensemble (jeu) de données introduit en 1936 par Ronald Aylmer Fisher comme un exemple d'analyse discriminante. Cet ensemble contient 150 exemples de critères observés sur espèces différentes d'iris de Gaspésie (Setosa, Versicolor, Verginica). Chaque exemple est composé de quatre attributs (longueur et largeur des sépales en cm, longueur et largeur des pétales en cm) et d'une classe (l'espèce).

Iris est disponible dans scikit-learn dans le package datasets. Comme tous les datasets offerts, il est constitué de deux dictionnaires :

.data qui stocke un tableau de dimensions *n.m* ou *n* est le nombre d'instances, et *m* le nombre d'attributs. On dispose ainsi, pour chaque instance, de la valeur de chacun des attributs - en l'occurrence des réels chez Iris.

.target qui stocke les classes (étiquettes) de chaque instance (dans le cas supervisé).

Les instructions Python suivantes permettent de charger le jeu de données Iris, et d'afficher la partie data (description des données en termes d'attributs) et la partie target (classe, cible, étiquette).

```
from sklearn import datasets
```

```
irisData = datasets.load_iris()
print (irisData.data)
print (irisData.target)
```

2. A vous !

a. Comprendre et programmer les quelques lignes précédentes : comment sont réparties les données dans les tableaux ? Combien y a-t-il de données dans chaque classe ? Quels sont les attributs et la classe du 32ème élément de l'échantillon ?

b. Comprendre, commenter et programmer le code source suivant :

```
import matplotlib
import pylab as pl
def plot_2D(data, target, target_names):
  colors = cycle('rgbcmykw') # cycle de couleurs
  target_ids = range(len(target_names))
  pl.figure()
  for i, c, label in zip(target_ids, colors, target_names):
   pl.scatter(data[target == i, 0], data[target == i, 1], c=c, label=label)
  pl.legend()
  pl.show()
```

S'en inspirer pour produire l'affichage de la répartition des données selon d'autres couples d'attributs.

c. Est-ce qu'il existe un espace 2D (constitué de deux attributs) dans lequel une droite (parmi d'autres) permettrait de séparer les exemples d'une classe des exemples des deux autres classes ? Si oui, fournir une équation de cette droite.

3. Autres jeux de données !

D'autres jeux de données sont directement disponibles dans scikit-learn (notamment *boston, diabetes, digits, linnerud, sample images, 20newsgroups*) via une fonction de chargement spécifique (load name) ou un processus de rapatriement web facile à mettre en œuvre (fetch).
Il existe aussi des générateurs d'échantillons, ainsi que la possibilité de récupérer, au bon format d'entrée, les jeux de données compilés sur le site mldata.org.

II- Un premier apprentissage de classiffieur

1. La variable clf :

Sous scikit-learn, dans toute méthode de classification (plus exactement, tout estimateur) représentée par la variable `clf`, il existe deux fonctions essentielles :

a. La fonction `clf.fit(tab_data, tab_target)` qui apprend un modèle à partir des données (ce modèle est stocké en interne dans un enregistrement de la variable `clf`),
b. La fonction `clf.predict(tab_data)` qui renvoie un tableau qui stocke, pour chaque nouvelle donnée en entrée, la classe prédite par le modèle précédemment appris via l'estimateur `clf`..

2. A vous!

L'algorithme Naïve Bayes Multinomial permet d'apprendre un modèle de classification en se basant sur la règle de Bayes, avec hypothèse d'indépendance des attributs; l'aspect multinomial indique que cet algorithme fonctionne sur des données décrites par plusieurs attributs à valeurs discrètes.

Cet algorithme est implanté sous la plupart des logiciels d'apprentissage automatique, notamment sous scikit-learn dans le package naive bayes.

a. Comprendre et programmer l'exemple ci-dessous : quel est le résultat ?

```
>>> from sklearn import naive_bayes
>>> nb = naive_bayes.MultinomialNB(fit_prior=True)# un algo d'apprentissage
>>> irisData = datasets.load_iris()
>>> nb.fit(irisData.data[:-1], irisData.target[:-1])
>>> p31 = nb.predict(irisData.data[31])
>>> print p31
>>> plast = nb.predict(irisData.data[-1])
>>> print plast
>>> p = nb.predict(irisData.data[:])
>>> print p
```

b. De la même façon : que réalise le programme ci-dessous ? Les résultats vous semblent-ils cohérents ? Comment interpréter les résultats ? Que proposez-vous pour équilibrer l'ensemble d'apprentissage et l'ensemble de test ?

```
>>> from sklearn import naive_bayes
>>> nb = naive_bayes.MultinomialNB(fit_prior=True)
>>> nb.fit(irisData.data[:99], irisData.target[:99])
>>> nb.predict(irisData.data[100:149])
```

III- Evaluer les performances d'un classiffieur

L'objectif ici est de lancer des apprentissages sur le jeu de données Iris, et d'évaluer la performance de ces apprentissages en calculant pour chacun :

- L'erreur d'apprentissage
- L'estimation de l'erreur réelle par séparation en 2 parties de l'échantillon d'apprentissage
- L'estimation de l'erreur réelle par validation croisée (cross validation).
-

1. Performances sur l'ensemble d'apprentissage

Il suit, pour évaluer la performance d'un classiffieur sur l'ensemble des données qui a servi à sa construction, de lancer la prédiction sur les mêmes données qui ont été fournies à la méthode fit ; puis de comparer le résultat de prédit au target de ces données. Pour cela, plusieurs façons très simples existent :

1. Soit P le tableau qui contient les prédictions, et Y le tableau qui contient les cibles (Y = irisData.target). Le code suivant permet de compter les erreurs et ainsi de calculer l'erreur d'apprentissage :

```
ea = 0
for i in range(len(irisData.data)):
if (P[i] != Y[i]):
ea = ea+1
     print ea/len(irisData.data)
```

2. Les opérateurs sur les tableaux et matrices devraient vous permettre d'effectuer ce comptage en une seule instruction !

Indice : pensez à exploiter le tableau P-Y : que représentent des valeurs non nulles ?
Comment compter (en une seule instruction) le nombre de valeurs non-nulles[1] ?

3. La méthode `clf.score(X,y)` permet de calculer le taux de bonne classification du modèle appris par la méthode `clf`, sur un ensemble de données stocke dans X et dont les classes sont stockées dans y. Le taux de bonne classification a, appelé en anglais accuracy, est tel que **a = 1 - e** si **e** est le taux d'erreur.

[1] package Python numpy : cherchez la doc de la fonction count nonzero

Implantez les trois méthodes décrites ci-dessus pour calculer l'erreur apparente. Pour chacune, indiquez votre code. Pour chacune, quelle erreur apparente obtenez-vous ? (Priez pour que ce soit la même ! Sinon, vous vous êtes trompés quelque part...). Sur quelle(s) données (s) y a-t-il une erreur de prédiction ?

2. Performances en généralisation

2.1. Estimer l'erreur réelle par division de l'échantillon d'apprentissage

Difficile de connaître l'erreur sur les données à venir, puisqu'on ne les connait pas encore... O suppose l'existence d'une distribution P(X; y) sur notre échantillon, même si on ne connait pas cette distribution.

Il est alors possible d'imaginer que l'on peut séparer notre échantillon S en deux : une partie S1 pour apprendre le modèle, et une autre partie S2 pour tester le modèle appris. On essaye d'avoir $|S_1| = 2\,|S_2| = 2/3\,|S|$. Attention : on doit avoir S_1 et S_2 disjointes, chacune est supposée suivre la même loi que P !

Comme on connait les étiquettes des exemples de S_2, on peut comparer les prédictions du modèle appris sur S_1, aux cibles (target) réelles de S_2.

a. Créez la fonction **`split(S)`** pour séparer en deux un échantillon S de couples (X, y) supposés uniformément repartis, selon le principe évoqué précédemment. On suppose que S est une variable de type datasets. Cette fonction renvoie un tableau à quatre éléments :

   `[dataS1, targetS1, dataS2, targetS2]`

   où `dataS1` et `dataS2` sont des matrices ayant le même nombre de colonnes que S.`data`, et où `targetS1` et `targetS2` sont des vecteurs d'étiquettes.

Pour cela, on peut utiliser les fonctions du package `random` de Python, donc la documentation est disponible en ligne: http://docs.python.org/library/random.html .

Testez cette fonction sur Iris.

b. Créez la fonction **`test(S, clf)`** qui sépare S en deux parties, puis apprend sur la première partie un modèle avec l'estimateur `clf`, et teste ce modèle sur l'autre partie, en comptant les erreurs faites sur l'échantillon : la proportion d'erreurs est renvoyée par la fonction. Testez cette fonction sur iris avec naive Bayes. L'erreur estimée est-elle plus petite que l'erreur apparente précédemment calculée ? Obtient-on toujours la même estimation pour l'erreur réelle ?

c. Réalisez t fois le test de la fonction test sur le même jeu de données et avec le même algorithme d'apprentissage, puis moyennez les erreurs estimées. Quelle erreur moyenne obtenez-vous pour t = 10; 50; 100; 200; 500; 1000 ? Puis, pour chaque t, répétez 20 fois l'expérience : pour chaque t, est-ce que l'erreur moyenne est stable ou non ? Pouvez-vous interpréter ce résultat ?

d. Est-ce que l'erreur estimée (dans sa version stable), dans le cas d'un échantillon de test qui ne prend que le 10ème de l'échantillon initial, est la même qu'avec la proportion d'1/3 ?

e. Il existe déjà, une fonction qui réalise le découpage d'un échantillon en deux parties disjointes. Il s'agit de la fonction `train_test_split`. Elle prend en entrée un échantillon de données (descriptions et cibles), et le sépare, selon une proportion en paramètre, en un ensemble d'apprentissage et un ensemble test (par défaut : `test_size=0.25`). Elle renvoie ainsi quatre quantités, dans l'ordre : une matrice de descriptions des données pour l'apprentissage (D_train), une matrice de descriptions des données pour le test (D_test), le vecteur des classes (cibles) des données d'apprentissage (C train), et le vecteur des classes (cibles) des données de test (C test). La documentation en ligne est : http://scikit-learn.org/stable/modules/classes.html#module-sklearn.cross_validation

f. Testez cette fonction avec plusieurs valeurs pour test size (dont 0.33) pour estimer l'erreur réelle de naïve Bayes sur les données Iris. Quelles erreurs obtenez-vous ?

2.2 Estimer l'erreur réelle par validation croisée

Parmi les techniques utilisées pour estimer l'erreur réelle d'un classiffieur, la validation croisée sur n parties (folds) est une des plus courantes, quasi optimale d'après de récentes recherches. Son principe est le suivant :

- Soit un échantillon *S*.
- Soit un algorithme d'apprentissage *A*.
- Séparer S en n parties disjointes de tailles similaires *Si, i = 1..n*.
- Pour *i* allant de 1 à n : entrainer A sur toutes les *Sj* sauf Si, pour obtenir un classiffieur *hi* (méthode `fit`). Tester *hi* sur *Si* : calculer ainsi *Ei*, l'erreur faite par *hi* sur la partie *Ei* (méthode `predict`).
- Calculer E = $E = \sum_i \frac{E_i}{n}$

E est ainsi une bonne estimation de l'erreur réelle d'un classiffieur appris avec A sur S.

Testez la fonction cross val score du package `sklearn.cross_validation`, pour estimer l'erreur réelle par validation croisée 10 folds.
Quelle erreur obtenez-vous pour Iris avec naïve Bayes ? Donnez cette erreur avec 2 folds ? 3 folds ? 5 ou 8 folds ?

IV- Compte Rendu

a. Donnez le code
b. Produisez un tableau résumant les différentes erreurs obtenues (calculées et estimées) sur le jeu de données iris, et commentez les écarts.
c. Faites de même avec une autre méthode de classiffication, que nous avons vu au cours: sklearn.tree.DecisionTreeClassifier, sans modifier les paramètres par défaut. Est-ce que les erreurs obtenues par la méthode naïve Bayes et par la méthode arbre de décision, vous permettent d'indiquer si une méthode de classification est meilleure qu'une autre ?