SY32 – TD Apprentissage 01 : Apprentissage automatique 1D

Installation

Pour l'installation de `python` et des bibliothèques utilisées en TD, suivre ou s'inspirer des instructions données dans la page web de l'UV : `https://vision.uv.utc.fr/doku.php?id=setup-python`.

Exercice 1 : Classification de chats par leur poids

On souhaite distinguer deux types (A et B) de chat en utilisant le poids comme facteur discriminant. Les fichiers `cat_data_X.csv` et `cat_data_y.csv` contiennent les poids mesurés sur un ensemble de 300 chats ainsi que leur type, respectivement. Le premier fichier représente le poids et le second le type : -1=Type A et +1=Type B. La Figure 1 représente la distribution des données observées.

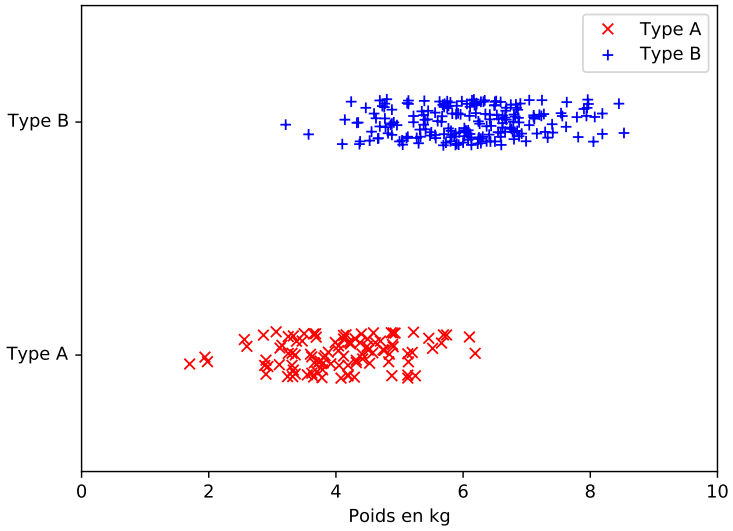

FIGURE 1 – Poids mesurés sur un ensemble de 300 chats.

On considère un classifieur f paramétré par une variable h tel que

$$f_h(x) = \begin{cases} -1 & \text{si } x \leq h \\ +1 & \text{sinon} \end{cases},$$

où x représente un poids.

1 . Charger les données contenues dans les fichiers `cat_data_*.csv` en utilisant la fonction `loadtxt` de la librairie `numpy`.

```
import numpy as np
X_train = np.loadtxt('cat_data_X.csv', ndmin=2)
y_train = np.loadtxt('cat_data_y.csv')
```

2 . Afficher quelques statistiques sur les données dont nous disposons :

- Quantité et proportion des deux types de chats traités (infos dans `y_train`).
- Poids mini, médian, et maxi entre tous les chats traités (infos dans `X_train`).
- Poids mini, médian, et maxi pour les deux types de chats (exploiter `X_train` et les annotations `y_train`).

3 . Le fichier `catClassifier.py` définit une classe Python `CatClassifier` pour le classifieur de chat. Écrire la fonction `predict` qui prend en entrée un tableau de poids `X`, une valeur de h et prédit pour chacun des poids le type de chat correspondant sous la forme d'un tableau `y`.

4 . Prédire le type des chats `X_train` avec $h = 5$.

```
from catClassifier import CatClassifier
clf = CatClassifier()
print(clf.predict(X_train, 5))
```

5 . Écrire la fonction `err_emp` de `CatClassifier` qui prend en entrée un ensemble de données `(X,y)` et une valeur de h puis calcule le taux d'erreur empirique du classifieur f_h sur l'ensemble `(X,y)`.

6 . Calculer le taux d'erreur empirique sur `(X_train,y_train)` avec $h = 5$.

```
print(clf.err_emp(X_train, y_train, 5))
```

7 . Écrire la fonction `fit` de `CatClassifier` qui prend en entrée un ensemble de données `(X,y)` et calcule une valeur optimale $\widehat{h}$ qui minimise le taux d'erreur sur `(X,y)`.

8 . Donner une valeur de h qui minimise le taux d'erreur sur `(X_train,y_train)`.

```
clf.fit(X_train, y_train)
print(clf.h_hat)
```

9 . Calculer le taux d'erreur empirique de $f_{\widehat{h}}$ sur l'ensemble `(X_train,y_train)`.

```
print(clf.err_emp(X_train, y_train))
```

10 . Comparer le taux d'erreur calculé à la question précédente avec le taux d'erreur réel consultable à l'adresse suivante, pour d'autres données similaires : `http://robotics.gi.utc/SY32/TD01/` (accessible uniquement depuis le réseau interne à l'UTC).

11 . Implémenter une validation croisée à K échantillons et calculer un nouvel estimateur du taux d'erreur associé à la valeur de $\widehat{h}$ calculée à la question 7. Comparer le résultat pour différentes valeurs de K au taux d'erreur réel et conclure.

12 . La librairie `scikit-learn` propose différents échantillonnages pour la validation croisée :

- K-Fold
- Shuffle Split
- Stratified K-Fold
- Stratified Shuffle Split

La description des différents échantillonnages est disponible dans la documentation : `https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators` Discuter de l'intérêt de chacun de ces échantillonnages.

13 . Reproduire les résultats de la question 10 en utilisant les échantillonnages proposés dans `scikit-learn`.

Liste de fonctions utiles :

- `numpy.array_split`
- `numpy.squeeze`
- `numpy.flatten`
- `numpy.expand_dims`
- `numpy.newaxis`
- `numpy.loadtxt`
- `numpy.mean`
- `numpy.count_nonzero`
- `numpy.argmin`
- `numpy.shape`
- `numpy.size`
- `numpy.sum`
- `numpy.where`
- opérateur `~`