SY32 – TD Apprentissage 02 : Apprentissage automatique

On souhaite généraliser le classifieur développé au précédent TD dans le cas à plusieurs dimensions. Comme précédemment, on souhaite distinguer deux types (A et B) de chat. En plus du poids, on dispose maintenant de la taille du chat. Les fichiers `cat_data_X_train.csv` et `cat_data_y_train.csv` contiennent les poids et tailles mesurés sur un ensemble de 300 chats ainsi que leur type, respectivement. Dans le premier fichier, la première colonne représente le poids et la seconde la taille. Dans le second fichier le type est représenté par : -1=Type A et +1=Type B. La Figure 1 représente la distribution des données observées.

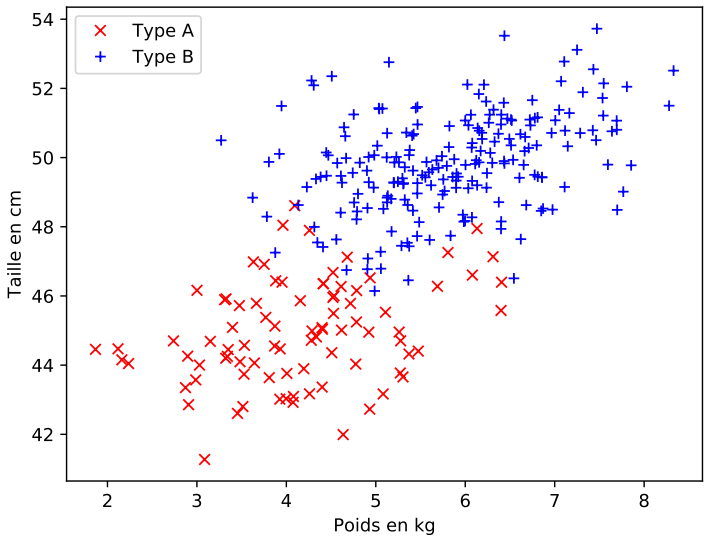

FIGURE 1 – Poids et tailles mesurés sur un ensemble de 300 chats.

Exercice 1 : Classifieur apprenant la meilleure dimension à utiliser

On définit un classifieur f paramétré par trois variable h, d et z tel que

$$ {}^{z}f_h^d(\mathbf{x}) = \begin{cases} z & \text{si } \mathbf{x}[d] \leq h \\ -z & \text{sinon} \end{cases} , $$

où $\mathbf{x} \in \mathbb{R}^D$ est un vecteur à D dimension, $h \in \mathbb{R}$, $d \in \{0, 1, \ldots, D-1\}$ et $z \in \{-1, +1\}$.

1 . En vous inspirant de la classe `CatClassifier`, implémentez dans le fichier `catClassifier.py` une nouvelle classe `CatClassifierMultiDim` permettant de trouver les paramètres $\widehat{h}$, $\widehat{d}$ et $\widehat{z}$ du classifieur. La classe comportera notamment les méthodes ci-dessous :

- `predict(self, X, h, d, z)`
- `err_emp(self, X, y, h, d, z)`
- `fit(self, X, y)`

2 . Trouver les paramètres optimaux pour les données des 300 chats de la base d'apprentissage et visualiser la frontière de décision à l'aide la fonction `plotCatClassifier` contenu dans le fichier `catClassifierPlot.py`.

```
from catClassifier import CatClassifierMultiDim
clf = CatClassifierMultiDim()
clf.fit(X_train, y_train)

from catClassifierPlot import plotCatClassifier
plotCatClassifier(clf)
```

Exercice 2 : Classifieur multi-dimensionnel Adaboost

On souhaite construire un meilleur classifieur à partir d'une approche par « boosting ». On rappelle que l'algorithme Adaboost est une approche permettant de combiner plusieurs classifieurs en modifiant itérativement le poids des exemples d'apprentissage. Pour un nombre K d'itérations, on procède de la manière suivante :

1. Initialiser $k = 0$ et les poids des $p_k^{(i)} = \frac{1}{n}$ pour tous les exemples d'apprentissage (les $n = 300$ chats de l'ensemble d'apprentissage) $\mathbf{x_i}$, $i \in \{1, \ldots, n\}$.
2. Tant que $k < K$,
   (a) Trouver le classifieur f_k qui minimise l'erreur de classification pondérée ϵ_k :

$$\epsilon_k = \sum_{i=1}^{n} p_k^{(i)} L\left(f_k(\mathbf{x}_i), y_i\right) \quad \text{où} \quad L\left(f_k(\mathbf{x}_i), y_i\right) = \begin{cases} 0 & \text{si } f_k(\mathbf{x_i}) = y_i \\ 1 & \text{sinon} \end{cases}.$$

   (b) On définit le poids du classifieur f_k comme $\alpha_k = \frac{1}{2} \ln \frac{1-\epsilon_k}{\epsilon_k}$.
   (c) On met à jour les poids des données d'apprentissage pour l'itération suivante :

$$p_{k+1}^{(i)} = \frac{p_k^{(i)} e^{-\alpha_k y_i f_k(\mathbf{x}_i)}}{2\sqrt{\epsilon_k(1-\epsilon_k)}}$$

   (d) Incrémenter k.
3. Le classifieur Adaboost final est donnée par :

$$H(\mathbf{x}) = \begin{cases} +1 & \text{si } \sum_{k=0}^{K} \alpha_k f_k(\mathbf{x}) > 0 \\ -1 & \text{sinon} \end{cases}$$

3 . Dans la classe `CatClassifierMultiDim`, ajouter un paramètre liste de poids p pour que la fonction `err_emp(self, X, y, h, d, z, p)` retourne l'erreur de classification pondérée pour chaque exemplaire testé.

4 . Dans la classe `CatClassifierMultiDim`, ajouter un paramètre liste de poids p pour que la fonction `fit(self, X, y, p)` trouve les paramètres $\widehat{h}$, $\widehat{d}$ et $\widehat{z}$ minimisant l'erreur de classification pondérée pour chaque exemplaire utilisé.

5 . Implémenter une classe `CatClassifierBoost` contenant une méthode `fit` et `predict` permettant de construire un *boosting* de `CatClassifierMultiDim`.

6 . Visualiser les frontières de décision pour $K \in \{3, 10, 20, 30, 40\}$.

7 . Calculer le taux d'erreurs empirique sur les données d'apprentissage pour les différentes valeurs de K.

8 . Calculer par validation croisée de nouveaux estimateurs du taux d'erreurs pour les différentes valeurs de K.

9 . Utiliser la validation croisée pour trouver un K « optimal » (K étant le nombre d'estimateurs pour le classifieur Adaboost).

10 . Prédire les types de chat de l'ensemble de test `cat_data_X_test.csv` dans un fichier `y_test.txt` et utiliser le site `http://robotics.gi.utc/SY32/TD02/` pour évaluer la performance du classifieur (accessible uniquement depuis le réseau interne à l'UTC).

```
X_test = np.loadtxt('cat_data_X_test.csv')
y_test = clf.predict(X_test)
np.savetxt('y_test.txt', y_test, fmt='%d')
```

Liste de fonctions utiles :

- `numpy.exp`
- `numpy.log`
- `numpy.sqrt`
- `numpy.mean`
- `numpy.loadtxt`
- `numpy.savetxt`
- `numpy.count_nonzero`