A2DI - TP n°2

Ce second TP se focalise sur des révisions de stats et l'évaluation de classifieurs.

Exercice n°1 : Statistiques sur la taille des hommes et des femmes

Dans cet exercice, nous allons manipuler un jeu de données dont chaque élément se compose de deux informations sur un être humain participant à l'étude :

— sa taille (en cm),

— et son sexe (homme ou femme).

Questions :

1. Lancez **Spyder** et créez un script python pour cet exercice. Chargez ce dataset à l'aide des commandes python suivantes :
```
taille_h = np.loadtxt("taille_h.txt")
taille_f = np.loadtxt("taille_f.txt")
```
Les fichiers seront fournis par l'encadrant.

2. Soit X_1 la v.a. réelle positive représentant la taille d'un humain et X_2 la v.a. catégorique représentant son genre ♂ou ♀.
Calculez et tracez la distribution empirique conditionnelle de la taille des hommes $\hat{p}_{X_1|X_2=♂}$, puis celle des femmes $\hat{p}_{X_1|X_2=♀}$.
La largeur d'un *bin* est fixée à 1cm pour ces 2 histogrammes.
Les variables X_1 et X_2 sont elles indépendantes ?

3. Selon les données de l'ONU, la population humaine est répartie par genre selon la distribution suivante :
— $p_{X_2}(♂) = 0.505$,
— $p_{X_2}(♀) = 0.495$.
Déterminez et tracer la loi marginale empirique $\hat{p}_{X_1}$ correspondant à la probabilité de la taille tous genres confondus.

4. Calculez, à partir de la distribution $\hat{p}_{X_1}$, la taille la plus fréquente (mode), la taille moyenne (espérance) et la taille médiane de la population humaine.

5. A quel genre appartient-on plus probablement lorsque l'on mesure entre 1,8m et 1,85m ? Et entre 1,6m et 1,65m ?

6. Calculez, à partir des données, l'écart-type de la taille chez l'homme et chez la femme.

7. Idem pour le moment centré d'ordre 3 (skew).

8. Faisons l'hypothèse que $\hat{p}_{X_1|X_2=♂}$ suit une loi Gaussienne de moyenne μ et d'écart type σ. Créez une fonction en python qui retourne la NLL de nos données sous un modèle Gaussien. Appelez cette fonction pour calculer la NLL pour μ variant de 140 à 220cm au pas de 1cm et pour σ variant de 10 à 40 au pas de 0.2. Vous rangerez ces valeurs dans un `numpy array` 2D.

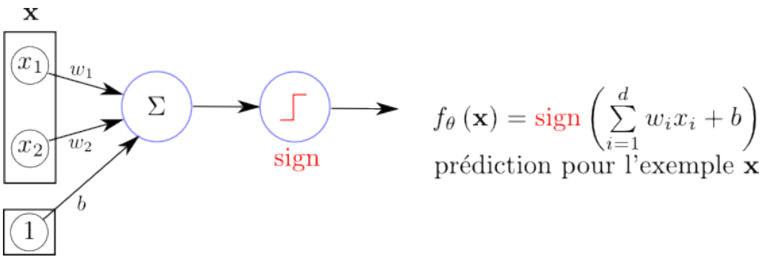
9. Affichez votre `numpy array` correspondant à la NLL avec la fonction `imshow` du module `matplotlib.pyplot`. Déterminez numériquement les valeurs de μ et σ maximisant la vraisemblance. Ces valeurs sont-elles proches de la moyenne et de l'écart type empirique ?

10. Superposez la distribution Gaussienne ainsi estimée à la distribution empirique. Le *fit* semble-t-il bon ?

Exercice n°2 : Vers l'erreur de généralisation d'un perceptron

Cet exercice suit directement l'exercice n°2 du TP n°1. On rappelle que le perceptron est un algorithme paramétrique cherchant à trouver dans l'espace d'attributs une séparation linéaire entre les classes (à supposer qu'une telle séparation existe).

La fonction de prédiction $f_{\boldsymbol{\theta}} : \mathbb{X} \rightarrow \mathcal{C}$ du perceptron s'obtient selon le schéma suivant :



La fonction sign est définie comme suit :

$$\begin{aligned} \mathrm{sign} : \mathbb{R} \quad &\longrightarrow \quad \{-1;+1\}\,, \\ x \quad &\longrightarrow \quad \begin{cases} +1 & \text{si } x \geq 0 \\ -1 & \text{si } x < 0 \end{cases}. \end{aligned} \tag{1}$$

Dans le TP précédent, nous avions implémenté un perceptron fonctionnement parfaitement car notre dataset était parfaitement calibré pour cet algorithme. Aujourd'hui, nous allons rendre la tâche du perceptron plus difficile en utilisant seulement 10% des exemples pour l'apprentissage contre 90% pour le test.

Le risque en faisant cela est que le perceptron converge vers une frontière de décision insuffisamment proche de la bonne réponse. En revanche, garder beaucoup d'exemples pour le test, nous confortera sur sa capacité réelle à généraliser, c'est à dire à donner une bonne réponse quand il recevra un nouvel exemple non vu pendant l'apprentissage.

1. Commencez par importer du TP précédent les fonctions `ptrain` et `ptest`.
2. Créez une fonction `datagen` qui génére un jeu de données répondant aux même contraintes que dans le TP précédent à l'exception du ratio entre nombre d'exemples d'apprentissage et nombre d'exemples de test. Cette fonction prend uniquement en entrée n la taille du jeu complet. Elle retourne :
   - `X_train` de type `numpy array` 2D contenant les exemples d'apprentissage,
   - `X_test` de type `numpy array` 2D contenant les exemples de test,
   - `c_train` de type `numpy array` 1D contenant les classes des exemples d'apprentissage,
   - `c_test` de type `numpy array` 1D contenant les classes des exemples de test.
3. Créez une fonction `get_test_err` qui calcule l'erreur de test à partir d'un jeu de données retourné par `datagen`. Cette nouvelle fonction prend également l'entier n en entrée.
4. Appelez la fonction `get_test_err` avec $n = 445$ plusieurs fois, de sorte à obtenir un échantillon représentatif de l'erreur de test. Représentez la distribution empirique (histogramme) de cet échantillon avec une largeur de *bin* de 0.5.

   La loi des grands nombres garantit que si l'échantillon est suffisamment grand, vous obervez une très bonne approximation de la distribution réelle de l'erreur. A vous de déterminer un critère d'arrêt pour la taille de l'échantillon.
5. Calculez l'espérance de l'erreur de test. Cette espérance est l'erreur de généralisation du perceptron dans cette situation.