Apprentissage statistique
Classification

Stéphane Marchand-Maillet
Department of Computer Science

10 mai 2023

Résumé

Le but de ce programme de travail est de comprendre et mettre en pratique certains mécanismes de la classification automatique des données par apprentissage statistique. Le langage de support pour la programmation des algorithmes est `python`. On apprend d'abord à accéder aux données et leurs labels, puis on explore les algorithmes de classifications "`k` plus proches voisins", "perceptron" et le "neurone logistique"

La classification de données consiste à apprendre et reconnaître automatiquement la catégorie (classe) d'un objet à partir de données exemples.

Example 1 (Classification d'image)**.** Étant données des images des chiffres "0" et "1", on veut automatiquement attribuer un label indiquant si l'image en question est un "1". Les caractéristiques (donnée) d'une image seront les valeurs des pixels de l'image et le label sera `0` si l'image montre un "0" et le label sera `1` si l'image montre un "1".

Example 2 (Classification de fruits)**.** Étant donnés des prunes et des cerises, on veut automatiquement attribuer un label à chaque fruit indiquant si c'est une cerise (classe "cerise", label `1`) ou une prune (classe "prune", label `0`). La caractéristique d'un fruit sera son diamètre.

Remark 1 (Rapport de travail). Pour chacune des tâches de ce travail, il est demandé de rédiger un rapport détaillant et documentant les solutions trouvées aux questions posées. Les résultats des expériences seront décrits et commentés afin de tirer des conclusions argumentées de ces expériences.

Remark 2 (Difficulté). Les tâches sont marquées par difficulté estimée : ① : facile, ② : moyen, ③ : difficile

1 Classification de données

Un modèle de classification est un modèle de décision dépendant de paramètres. Il permet de prédire la classe (`0` ou `1` pour la classification binaire) d'un objet inconnu en fonction des caractéristiques mesurées $\mathbf{x}$ de cet objet.

Example 3 (Modèle de classification de fruits)**.** Pour classifier des fruits (objets) selon leur diamètre (caractéristique), on définit une classe positive (classe `1`), par exemple "cerise", et les autres fruits seront de la classe négative (classe `0`). Pour différencier la classe "cerise" (positive) de la classe "prune" (négative) on doit choisir un seuil (paramètre) sur le diamètre, en deça duquel le fruit sera de la classe positive, sinon négative (décision).

Pour classifier les données, on distingue 3 phases. Les deux premières phases que sont l'apprentissage et le test visent à produire le modèle de classification (configurer les paramètres). Ces phases se font à l'aide de données connues (caractéristiques et labels connus). La finalité du modèle est son utilisation avec des données inconnues (caractéristiques connues, labels inconnus).

1.1 Apprentissage

Le but de la phase d'apprentissage (ou entraînement) est de configurer (à travers un algorithme d'apprentissage) les paramètres pour que les prédictions du modèle soient correctes sur l'ensemble d'entraînement.

Remark 3. Dans ce travail on verra 3 algorithmes d'apprentissage (kNN, perceptron, neurone logistique).

La qualité de l'apprentissage se mesure par une différence (erreur) pour chaque objet de l'ensemble d'entraînement, entre la classe prédite et la classe connue pour cet objet. La somme des erreurs forme l'erreur d'entraînement.

Remark 4. Dans le cas d'une classification binaire linéaire (2 classes séparées par une frontière linéaire, comme étudiée ici avec le perceptron ou le neurone logistique) on espère une erreur d'entraînement nulle pour des données séparables (par une frontière linéaire) et une erreur minimale pour des données non-séparables.

On voit donc que l'ensemble des données d'entraînement sera l'ensemble des caractéristiques des objets exemples pour l'entraînement avec leur label de classe (`0` ou `1`) que l'on fournit au modèle pour s'entraîner.

1.2 Test

Une fois les paramètres du modèle fixés, on simule son utilisation sur un cas réel en utilisant un ensemble de test. On estime la qualité du modèle lors de son utilisation en étudiant ses prédictions sur des objets dont on connaît les labels de classe (mais que l'on ne fournit pas au modèle). En comparant les erreurs faites par le modèle une fois ses paramètres fixés, on obtient l'erreur de test.

On voit donc que l'ensemble des données de test sera l'ensemble des caractéristiques des objets exemples pour l'entraînement avec leur label de classe (`0` ou `1`) que l'on ne fournira pas au modèle pour s'entraîner.

1.3 Utilisation

Une fois le modèle (et ses paramètres) validé(s) par une erreur de test acceptable, il peut être utilisé dans les conditions réelles. On reçoit un objet dont on mesure les caractéristiques que l'on fournit au modèle, qui produit une prédiction du label de classe de l'objet. Ne connaissant pas la classe réelle, on se fie à la qualité de l'entraînement et du test pour accepter cette prédiction.

Example 4 (Lecture de plaque minéralogique)**.** Une société technologique entraîne le modèle pour reconnaître les chiffres de la plaque minéralogique (leur attribuer une classe `0`, `1`, ...). La société de parking utilisant ce modèle de prédiction fera des tests avec des plaques connues pour tester la fiabilité du système. Elle pourra alors utiliser ce système pour reconnaître et facturer automatiquement les véhicules entrant et sortant des parking.

On voit donc clairement que l'on a besoin de jeux de données d'entraînement et de test contenant les caractéristiques des objets et leurs labels. En pratique, on obtiendra une ensemble de données contenant les données (data) et les labels correspondants (labels) que l'on partagera en données d'entrainement (80% par exemple) et test (20%).

On s'attache donc d'abord a obtenir et préparer ces données pour l'entraînement et le test des modèles étudiés.

2 Préparer les données

Le but de cette partie est d'accéder à des données sur lesquelles appliquer des algorithmes de traitement. Les données seront soit chargées à partir de fichiers, soit générées aléatoirement selon un modèle de données.

Pour l'apprentissage statistique, une donnée est un vecteur $\mathbf{x}_i \in \mathbb{R}^D$ qui contient les caractéristiques de l'objet i. Son label sera noté $y_i \in \{0, 1\}$. On notera N le nombre de données disponibles pour l'entraînement.

Remark 5 (Dimension des données). Dans ce travail on se limitera aux valeurs de dimension $D = 1$ ou $D = 2$ afin de pouvoir visualiser les résultats (1D ou 2D) mais les procédures définies ici se généralisent sans difficulté aux cas $D > 2$.

2.1 Charger des données à partir de fichiers

Les données sont souvent accessibles à partir de fichiers informatiques qu'on lit pour obtenir les données. Les formats de ces fichiers peuvent être très variables (`.txt`, `.ods`, `.xslx`, ...) et on utilisera l'outil de lecture adapté au format. Dans ce travail on utilise un format texte [a] générique : le format `CSV` (comma separated values). Dans ce format, chaque ligne contient des valeurs séparées par des virgules. Dans notre cas, chaque ligne i contient les D données d'un vecteur $\mathbf{x}_i$ séparées par des virgules.

(a). Les fichiers CSV peuvent être ouverts avec un éditeur de texte type `notepad` ou `vscode` ou être importés dans `open calc` ou `Excel`

Exercise 1 (① Lecture de fichier CSV)**.** Étant donnés les fichiers fournis :

(i) Écrire un programme qui, à partir d'un nom de fichier `xxx` lit les données dans le fichier `xxx-data.csv` et les labels correspondant dans le fichier `xxx-label.csv`. On stockera le résultat dans les variables (*numpy arrays*) `data` et `label`.

Hint. On pourra utiliser la fonction `numpy genfromtxt` pour lire ces fichiers

(ii) Afficher le nombre N et la dimension D de ces données

(iii) Sachant que pour ces données on a soit $D = 1$, soit $D = 2$, afficher graphiquement ces données en utilisant une couleur par label.

Hint. ② On pourra prévoir que les données peuvent éventuellement être de dimension $D > 2$ et comporter plus que 2 classes. Dans ce cas, on proposera d'afficher 2 dimensions et les classes au choix.

Exercise 2 (① Utilisation de packages)**.** Certaines librairies `python` offrent des facilités pour charger des ensembles de données standard.

(i) Utilisez le package `datasets` de la librairie `sklearn` pour charger les ensembles `iris` et `wine`

(ii) ② Testez l'affichage de ces données avec les outils de l'exercice 1

Exercise 3 (② Lecture des données `MNIST`)**.**

(i) Utilisez le package `datasets` de la librairie `keras` pour charger l'ensemble `MNIST`

(ii) Transformez cet ensemble en un ensemble de données vectorielles en alignant les colonnes des images en un seul vecteur

Hint. On pourra utiliser la fonction `numpy reshape` qui réorganise les valeurs dans les tableaux

2.2 Générer des données automatiquement

Si on connaît les statistiques des données que l'on veut générer, on peut utiliser le générateur aléatoire (package `random`) pour tirer des valeurs au hasard tout en respectant les statistiques de l'étude.

Exercise 4 (Données de classification pour les fruits ①)**.** On sait qu'une prune a un diamètre moyen égal à 6cm pour un poids moyen de 30g alors qu'une cerise a un diamètre moyen de 2cm pour un poids moyen de 5g.

(i) En utilisant les fonctions `numpy randint` et `randn` du package `random` de `python`, générez des ensembles de taille (N) variable ou fixée de caractéristiques (diamètre et poids) et labels de cerises et de prunes

(ii) Écrivez ces données dans des fichiers de format `CSV` de façon à pouvoir les lire et les afficher avec les programmes de l'exercice 1

Exercise 5 (② Données sur des cercles)**.**

(i) Grâce aux fonctions aléatoires `randint` et `rand`, générez des données se situant autour de cercles séparés, s'intersectant ou concentriques [(b)]

(ii) Écrivez ces données dans des fichiers de format `CSV` de façon à pouvoir les lire et les afficher avec les programmes de l'exercice 1

Exercise 6 (① Utilisation de packages)**.** Certaines librairies `python` offrent des facilités pour générer des ensembles de données de modèles standard.

(i) Utilisez le package `datasets` de la librairie `sklearn` pour générer les différents types de données proposées.

(ii) Écrivez ces données dans des fichiers de format `CSV` de façon à pouvoir les lire et les afficher avec les programmes de l'exercice 1

Exercise 7 (② Utilisation de `Open Office` ou `Excel`)**.**

(i) Utilisez `Open Calc` ou `Excel` pour générer des données de votre choix. Documentez

(ii) Écrivez ces données dans des fichiers de format `CSV` de façon à pouvoir les lire et les afficher avec les programmes de l'exercice 1

(b). On rappelle que les points $\mathbf{x}$ d'un cercle de rayon r sont à une distance $d(\mathbf{x}, \mathbf{c}) = r$ du centre $\mathbf{c}$. Les coordonnées $\mathbf{x} = [\mathbf{x}_{[0]}, \mathbf{x}_{[1]}]$ de ces points et du centre $\mathbf{c} = [\mathbf{c}_{[0]}, \mathbf{c}_{[1]}]$ sont donc tels que $d(\mathbf{x}, \mathbf{c}) = \sqrt{(\mathbf{x}_{[0]} - \mathbf{c}_{[0]})^2 + (\mathbf{x}_{[1]} - \mathbf{c}_{[1]})^2} = r$

2.3 Extraire les statistiques de données

Les données sont donc un tableau `numpy` `data` (*numpy array*) de dimension $N \times D$ dont chaque ligne est une donnée (les caractéristiques de l'objet correspondant) et chaque colonne les valeurs de cette caractéristique prises par les données [(c)]. On va faire des statistiques pour étudier la dispersion de ces données.

Exercise 8 (① Boites englobantes et intersection)**.**

(i) Pour un choix de données obtenues dans les étapes précédentes, calculez la boîte englobante des données (minimum, maximum) de chaque classe, ainsi que la médiane de chaque caractéristique. Pour ce faire, utilisez les fonctions `numpy` `median`, `min` et `max` sur les variables `data` et `label`

(ii) Pour chacune des situations ci-dessus vérifiez si des données sont présentes dans l'intersection des boites englobantes des données. Quelle est votre interprétation ?

On peut enrichir ces statistiques par des statistiques plus globales.

Exercise 9 (① Moyenne et variance)**.**

(i) Pour un choix de données obtenues dans les étapes précédentes, calculez la moyenne (et la variance) de chacune des caractéristiques des données pour chaque classe.

Hint. On pourra utiliser simplement les fonctions `numpy` ou réimplémenter ces fonctions et comparer les résultats.

(ii) Pour les données générées aléatoirement (section 2.2) vérifiez que les paramètres retrouvés correspondent bien aux paramètres de génération

(iii) ② Variez le nombre de données considérées et comparez la qualité des l'estimation des paramètres de départ. Quelle est votre interprétation ?

On peut aussi visualiser la distribution des caractéristiques des données.

Exercise 10 (① Histogrammes)**.**

(i) Pour un choix de données obtenues dans les étapes précédentes, tracer les histogrammes de la distribution des caractéristiques des données, par classes.

Hint. On utilisera la fonction `hist` du package `pyplot` de la librairie `matplotlib`. On pourra varier le paramètre `bins` pour visualiser au mieux l'histogramme des données

(ii) Quelle est votre interprétation de la lecture de ces histogrammes par rapport aux résultats des exercices 8 et 9 ?

3 Le classifieur "k-plus proches voisins"

Le classifieur des "k-plus proches voisins" kNN (k-*nearest neighbors*) utilise la similarité entre les objets pour effectuer la classification. Pour un objet inconnu $\mathbf{x}_q$, on cherche les k objets $\mathbf{x}_i$ les plus proches (voisins) dans notre ensemble de données (pour lesquels on connaît les labels) et on choisit pour $\mathbf{x}_q$ le label majoritaire (le plus fréquent) dans cet ensemble.

Remark 6. ⚠ Ce classifieur ne nécessite pas d'entraînement, mis à part le choix de la valeur k. De plus, ce classifieur ne définit pas explicitement de frontière de classification globale mais attribue un label à chaque donnée selon le label de ses voisins (localement).

3.1 Proximité entre objets

La proximité $\text{prox}(\mathbf{x}_i, \mathbf{x}_j)$ entre les objets $\mathbf{x}_i$ et $\mathbf{x}_j$ est mesurée par la distance entre leurs caractéristiques. Plus la valeur est faible, plus les objets sont proches. On a plusieurs choix [(d)] [(e)] :

— La différence : $\text{prox}(\mathbf{x}_i, \mathbf{x}_j) = \sum_{d=1}^{D} |\mathbf{x}_{i\,[d]} - \mathbf{x}_{j\,[d]}|$

— La distance Euclidienne : $\text{prox}(\mathbf{x}_i, \mathbf{x}_j) = d(\mathbf{x}_i, \mathbf{x}_j)^2 = \sum_{d=0}^{D-1} (\mathbf{x}_{i\,[d]} - \mathbf{x}_{j\,[d]})^2$

Exercise 11 (Calcul de distance Euclienne entre objets)**.** Ecrivez un programme qui

— Tire aléatoirement les caractéristiques de dimension $D = 2$ pour N objets

— Calcule leur distance Euclienne par la formule $d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_{i\,[0]} - \mathbf{x}_{j\,[0]})^2 + (\mathbf{x}_{i\,[1]} - \mathbf{x}_{j\,[1]})^2}$

(c). Attention : pour l'analyse de données, les vecteurs sont classiquement des colonnes. Le premier indice est l'indice de ligne
(d). la notation $\sum_{i=1}^{N} \nu_i$ se lit "somme pour i égal 1 à N des valeurs ν_i", soit $\sum_{i=1}^{N} \nu_i = \nu_1 + \nu_2 + \nu_3 + \cdots + \nu_N$
(e). la notation $\mathbf{x}_{[k]}$ se lit "la coordonnée k du vecteur $\mathbf{x}$"

— Fait la même chose pour $D > 0$ par la formule $d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{d=0}^{D-1} (\mathbf{x}_{i[0]} - \mathbf{x}_{j[0]})^2}$

Exercise 12 (Voisinage)**.** Écrivez un programme qui, pour un ensemble de données obtenu précédemment :

— Choisit une donnée aléatoirement (fonction `numpy.random` `randint` pour le choix d'indice)

— Calcule la proximité avec chacune des données de l'ensemble (cf au choix ci-dessus)

— Trie les valeurs de proximité dans l'ordre croissant (fonction `numpy` `argsort`)

— Imprime les labels des k plus proches voisins ($k < N$ au choix)

— Affiche les données sous forme d'un `scatter` plot (si 2D) avec la donnée choisie en rouge (`c='r'`), les k plus proches voisins en vert (`c='g'`), le reste en noir (`c='k'`). On pourra varier en affichant les données de la classe `0` avec des cercles (`marker='o'`) et les données de la classe `1` avec des croix (`marker='x'`)

3.2 Classification par "k-plus proches voisins" (kNN)

L'algorithme de classification par k plus proches voisins d'une donnée $\mathbf{x}$ sera donc comme suit :

Algorithm 1 Algorithme de classsification par k plus proches voisins

procedure kNearestNeighbours(k,$\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$)
  for chaque donnée $\mathbf{x}_i$ **do**
    Calculer $\mathbf{d}_{[i]} = \text{prox}(\mathbf{x}, \mathbf{x}_i)$
  Trier $\mathbf{d}$ par ordre croissant
  $\{i_1, \cdots, i_k\} \leftarrow$ indices des données correspondant aux k plus petites valeurs de $\mathbf{d}$
  $y_{\text{classif}} \leftarrow$label majoritaire parmi $\{y_{i_1}, \cdots, y_{i_k}\}$

Exercise 13 (Classification kNN)**.** En utilisant le programme précédent pour $k < N$ fixé, écrire un programme qui :

(i) Choisit une donnée $\mathbf{x}$ aléatoirement

(ii) Sélectionne les labels des k données les plus proches

(iii) Attribue à (prédit pour) $\mathbf{x}$ le label majoritaire dans cet ensemble

(iv) Affiche le label prédit pour $\mathbf{x}$ et le compare au label connu

(v) Comparer les résultats avec ceux de la fonction `neighbors.KNeighborsClassifier` du package `sklearn` [1]

3.3 Test

Comme décrit en section 1, tout classifieur doit être testé avant de pouvoir être utilisé. Pour le test, on dispose d'un ensemble de données de test qui ont la même structure que les données d'entraînement : pour chaque donnée $\mathbf{x}_i$ on connaît son label y_i avec $i = 1, \cdots, N_{\text{test}}$.

Le test du classifieur classif se déroule donc selon l'algorithme suivant : *TBD*

Algorithm 2 Algorithme de test d'un classifieur

procedure Test_classifieur(classif, $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N_{\text{test}}}$)
  $E_{\text{test}} \leftarrow 0$
  for chaque donnée $\mathbf{x}_i$ **do**
    $y_{\text{classif}} \leftarrow \text{classif}(\mathbf{x}_i)$
    $E_{\text{test}} \leftarrow E + |y_i - y_{\text{classif}}|$

4 Le perceptron

Le perceptron est un algorithme de classification binaire (2 classes) dont l'entrainement détermine les paramètres d'une frontière de classification. Dans ce cas précis la frontière de classification est linéaire et partage l'espace des données en 2 parties (de chaque côté de la frontière pour la classe `0` et la classe `1`.

4.1 Cas 1D

Dans le cas le plus simple où chaque donnée n'a qu'une seule caractéristique $\mathbf{x}_{[1]}$ (par exemple le diamètre pour des fruits), l'entrainement du classifieur revient à déterminer la valeur d'un seuil (noté w_0) sur la caractéristique de classification afin que les prédictions correspondent aux valeurs connues (lors de l'entraînement). Si la valeur de caractéristique de la donnée $\mathtt{i}$ est inférieure à ce seuil on attribue le label $y_{\text{classif}} = 1$ à cette donnée sinon $y_{\text{classif}} = 0$. On écrit donc le test de classification comme

$$\mathbf{x}_{[1]} < w_0 \quad \text{alors} \quad y_{\text{classif}} = 1 \quad \text{sinon} \quad y_{\text{classif}} = 0$$

Afin de généraliser (cf plus bas) on fixe $\mathbf{x}_{[0]} = 1$ et on réécrit le test de classification comme [(f)]

$$w_1\mathbf{x}_{[1]} + w_0\mathbf{x}_{[0]} > 0 \quad \text{alors} \quad y_{\text{classif}} = 1 \quad \text{sinon} \quad y_{\text{classif}} = 0$$

Example 5. (Classification de fruits) On veut classifier des cerises et de prunes par leur taille. On postule que la classe 1 correspond à "être une cerise". Si on fixe $w_0 = 4$ (cm) et $w_1 = -1$, un fruit de diamètre $\mathbf{x}_{\mathtt{i}[1]} = 2$ (cm) sera classifié comme

$$w_1\mathbf{x}_{\mathtt{i}[1]} + w_0\mathbf{x}_{[0]} = -1 \cdot 1.9 + 4 \cdot 1 = 2.1 > 0 \quad \text{alors} \quad y_{\text{classif}} = 1 \quad \text{(cerise)}$$

alors qu'un fruit de diamètre $\mathbf{x}_{\mathtt{i}[1]} = 5.5$ (cm) sera classifié comme

$$w_1\mathbf{x}_{\mathtt{i}[1]} + w_0\mathbf{x}_{[0]} = -1 \cdot 5.5 + 4 \cdot 1 = -1.5 > 0 \quad \text{alors} \quad y_{\text{classif}} = 0 \quad \text{(prune)}$$

4.1.1 Entrainement

Les paramètres w_0 et w_1 vont donc contrôler le test de classifcation d'un objet de caractéristique $\mathbf{x}_{[1]}$ (on rappelle que l'on fixe toujours $\mathbf{x}_{[0]} = 1$). Comme on ne connait pas a priori la valeur des ces paramètres, on va les apprendre grâce à des exemples de fruits dont on connait la classe (l'ensemble d'entrainement).

Soit $\{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ ces données et $\{y_1, \cdots, y_N\}$ les labels correspondants. On se fixe l'agorithme d'entraînement suivant : On constate que l'algorithme a les propriétés suivantes

Algorithm 3 Algorithme d'entraînement du Perceptron pour le cas 1D

```
procedure ENTRAINEMENT_PERCEPTRON_1D(({(x_i, y_i)}_{i=1}^{N_train}))
    Choisir des valeurs aléatoires pour les paramètres w_0 et w_1        ▷ Initialisation
    Fixer x_i[0] = 1 pour tout i
    while les valeurs de paramètres ne sont pas fixes do
        for chaque donnée d'entraînement (x_i, y_i) do
            if w_1 x_i[1] + w_0 x[0] > 0 then                            ▷ Classification
                y_classif = 1
            else
                y_classif = 0
            w_0 ← w_0 + (y_i − y_classif) · x_i[0]                       ▷ Mise à jour
            w_1 ← w_1 + (y_i − y_classif) · x_i[1]
```

— Si la donnée $\mathbf{x}_\mathtt{i}$ est bien classifiée ($y_\mathtt{i} = y_{\text{classif}} \Rightarrow y_\mathtt{i} - y_{\text{classif}} = 0$), rien ne change. Une fois toutes les données d'entraînement classifiées correctement, l'algorithme s'arrête

— Si $y_\mathtt{i} = 1$ et $y_{\text{classif}} = 0$ ($y_\mathtt{i} - y_{\text{classif}} = 1$) alors $w_\mathtt{k} \leftarrow w_\mathtt{k} + \mathbf{x}_\mathtt{i}$. Ceci est cohérent avec le fait que comme $y_{\text{classif}} = 0$, le seuil est a gauche de la donnée selon le test de classification. Or comme $y_\mathtt{i} = 1$, le seuil devrait être à droite de la donnée. La règle de mise à jour est bien cohérente avec le fait de ramener le seuil à gauche de la donnée.

— Si $y_\mathtt{i} = 0$ et $y_{\text{classif}} = 1$ ($y_\mathtt{i} - y_{\text{classif}} = -1$) alors $w_\mathtt{k} \leftarrow w_\mathtt{k} - \mathbf{x}_\mathtt{i}$. On a le raisonnement inverse de ci-dessus, donc on respecte bien le sens du problème

Remark 7. Si les données ne sont pas séparables (par un seuil dans le cas 1D), l'algorithme peut ne pas converger. L'erreur d'entraînement n'est pas nulle. Il convient donc dans ce cas d'arrêter l'algorithme après un nombre fixé d'itérations.

Exercise 14 (Entraînement du perceptron)**.**

(i) Implémentez l'algorithme d'entraînement du perceptron

(f). On voit que pour $w_1 = -1$ les deux écritures sont équivalentes.

(ii) En utilisant les données chargées lors des exercices précédents, réalisez l'entraînement du perceptron pour ces données

(iii) Donnez l'erreur d'entraînement pour chaque ensemble de données d'entraînement

(iv) Visualisez le seuil et les données d'entraînement

(v) Stockez et visualisez l'évolution de l'erreur d entraînement (classe `0` en rouge, classe `1` en vert)

4.1.2 Test

Le test du classifieur Preceptron s'effectue exactement comme celui du classifieur kNN, selon l'algorithme 3.3. Le test servira de confirmer que le classsifieur a de bonnes performances sur des données différenetes des données d'entrainement. On parle de capacité de généralisation du classifieur.

Exercise 15 (Test du perceptron)**.** Une fois l'entraînement terminé et les paramètres (w_0 et w_1) du perceptron fixés :

(i) Effectuez le test du classifieur selon l'algorithme 3.3

(ii) Visualisez le seuil et les données de test (classe `0` en rouge, classe `1` en vert) et les données d'entraînement (carré - `marker='s'` - pour les données de test, croix - `marker='x'` - pour les données d'entraînement)

4.2 Cas 2D

Algorithm 4 Algorithme d'entraînement du Perceptron pour le cas 1D

procedure ENTRAINEMENT_PERCEPTRON_2D($\{(\mathbf{x}_i, y_i)\}_{i=1}^{N_{train}}$)
  Choisir des valeurs aléatoires pour les paramètres w_0, w_1 et w_2 ▷ Initialisation
  Fixer $\mathbf{x}_i[0] = 1$ pour tout i
  while les valeurs de paramètres ne sont pas fixes **do**
    for chaque donnée d'entraînement $(\mathbf{x}_i, y_i)$ **do**
      if $w_1\mathbf{x}_i[1] + w_0\mathbf{x}[0] > 0$ **then** ▷ Classification
        $y_{classif} = 1$
      else
        $y_{classif} = 0$
      $w_0 \leftarrow w_0 + (y_i - y_{classif}) \cdot \mathbf{x}_i[0]$ ▷ Mise à jour
      $w_1 \leftarrow w_1 + (y_i - y_{classif}) \cdot \mathbf{x}_i[1]$
      $w_2 \leftarrow w_2 + (y_i - y_{classif}) \cdot \mathbf{x}_i[2]$

4.3 Cas général

Algorithm 5 Algorithme d'entraînement du Perceptron pour le cas 1D

procedure ENTRAINEMENT_PERCEPTRON($\{(\mathbf{x}_i, y_i)\}_{i=1}^{N_{train}}$)
  Choisir des valeurs aléatoires pour les paramètres w_0, w_1 et w_2 ▷ Initialisation
  Fixer $\mathbf{x}_i[0] = 1$ pour tout i
  while les valeurs de paramètres ne sont pas fixes **do**
    for chaque donnée d'entraînement $(\mathbf{x}_i, y_i)$ **do**
      if $w_1\mathbf{x}_i[1] + w_0\mathbf{x}[0] > 0$ **then** ▷ Classification
        $y_{classif} = 1$
      else
        $y_{classif} = 0$
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + (y_i - y_{classif})\mathbf{x}_i$ ▷ Mise à jour

5 Le neurone logistique

TBD

A Rappels mathématiques

A.1 Notation

Une variable $\nu \in \mathbb{R}$ représente un nombre réel (élément de l'ensemble $\mathbb{R}$ des nombres réels). Si on doit énumérer des valeurs, on notera $\nu_i \in \mathbb{R}$ afin que ν_0, ν_1, ... représentent les différentes valeurs à énumérer.

Example 6. (diamètre des cerises). Etant données N cerises, on mesure leur poids et on obtient $p_0 = 5.1$g pour la première cerise, $p_1 = 4.9$g pour la deuxième cerise, ..., $p_{N-1} = 5.3$g pour la Nième cerise.

Remark 8 (Listes `python`). Les indices commencent avec l'indice 0 pour rester cohérent avec la notation des listes `python`.

```
x = [5,4,6,2,3,7]
print(x[0])
# retourne 5
```

Cette notation permet d'écrire facilement la somme ($\sum$) ou le produit ($\prod$)

$$\sum_{i=0}^{N-1} p_i = p_0 + p_1 + \cdots + p_{N-1} \qquad \prod_{i=0}^{N-1} p_i = p_0 \cdot p_1 \cdots p_{N-1}$$

```
x = [5,4,6,2,3,7]
print(sum(x))     # ecrit 27
```

A.2 Vecteurs et matrices

Dans ce travail, on definit un vecteur $\boldsymbol{x}$ comme un élément de $\mathbb{R}^D$, c'est à dire une liste de D nombres réels. Par convention un vecteur s'écrit en colonnes

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_{[0]} \\ \vdots \\ \boldsymbol{x}_{[D-1]} \end{bmatrix}$$

$\boldsymbol{x}_{[i]} \in \mathbb{R}$ est la $i^{\text{ième}}$ composante de $\boldsymbol{x}$.

```
import numpy as np
x = np.array([5,4,6,2,3,7])
print(x[3])        # ecrit 2
print(x.sum())     # ecrit 27
```

On peut faire les opérations d'addition et de soustraction entre des vecteurs de même taille.

On peut étendre la notion de vecteurs à celle de matrices comme tableau de chiffres en accolant des vecteurs les unsà coté des autres. Par exemple la matrice $\boldsymbol{A} \in \mathbb{R}^{N \times M}$ est une aggrégation de M vecteurs de taille N :

$$\boldsymbol{A} = \begin{bmatrix} a_{00} & \cdots & a_{0M} \\ \vdots & & \vdots \\ a_{N0} & \cdots & a_{NM} \end{bmatrix}$$

Pour `python`, une matrice est un vecteur (`numpy array`) de vecteurs :

```
import numpy as np
A = np.array([[5,4,6],
              [2,3,7],
              [2,8,4],
              [7,10,12]])
print(A.shape)    # donne la taille de la matrice soit (4,3)
print(A[2,1])     # ecrit 8
```

On peut faire les opérations d'addition et de soustraction entre des matrices de même taille. On notera aussi a_{ij} le $j^{\text{ième}}$ élément de la $i^{\text{ième}}$ ligne de $\boldsymbol{A}$ ($\boldsymbol{A}_{[i,j]}$).

A.3 Distance

Dans un repère Cartésien d'origine $\mathbf{o} = [0, 0]$, un vecteur $\mathbf{x} = [\mathbf{x}_{[0]}, \mathbf{x}_{[1]}]$ est équivalent au point de coodonnées $(\mathbf{x}_{[0]}, \mathbf{x}_{[1]})$. La distance entre les points $\mathbf{x} = [\mathbf{x}_{[0]}, \mathbf{x}_{[1]}]$ et $\mathbf{y} = [\mathbf{y}_{[0]}, \mathbf{y}_{[1]}]$ est donc la longueur du vecteur $\mathbf{x} - \mathbf{y}$, que l'on peut facilement calculer par le théorème de Pythagore.

$$\mathrm{d}(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x}_{[0]} - \mathbf{y}_{[0]})^2 + (\mathbf{x}_{[1]} - \mathbf{y}_{[1]})^2}$$

```python
import numpy as np

def distance(x,y):
    return np.sqrt((x[0]-y[0])**2+(x[1]-y[1])**2 )

x = np.array([6,4])
y = np.array([2,3])
print(distance(x,y))     # retourne sqrt(17)
```

Que l'on peut généraliser (les deux vecteurs doivent avoir la même taille) :

```python
import numpy as np

def distance(x,y):
    return np.sqrt(((x-y)**2).sum())

x = np.array([5,4])
y = np.array([2,3])
print(distance(x,y))   # ecrit sqrt(10)
```

A.4 Aléa et Statistiques

Références

[1] kNN classifier from the `sklearn` package. `sklearn.neighbors.KNeighborsClassifier`.

[2] Daniel Smilkov and Shan Carter. A neural network playground, 2018. (online : https://playground.tensorflow.org).