Etude Pratique (2) – Master Informatique – IAA
Université d'Aix-Marseille

Cécile Capponi – QARMA, LIS – AMU
Cecile.Capponi@lis-lab.fr

24 février 2021

Objectifs de la séance :
- étudier et comprendre le fonctionnement de l'algorithme du perceptron,
- analyser et implémenter deux variantes de l'algorithme, dédiées à la classification binaire et à la classification multi-classes,
- tester les algorithmes sur des données simulées et des données réelles (convergence, performances)
- pratiquer l'écriture de fonctions Python simples.

Le fichier `perceptron_data.py` contient les jeux de données sur lesquels vous allez appliquer l'algorithme du perceptron.

1 Algorithme du perceptron pour la classification binaire

L'algorithme de perceptron, dans sa version standard, est dédié à la classification binaire. Les données d'apprentissage utilisées dans cette section s'écrivent donc sous la forme $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ avec $\mathbf{x}_i \in \mathbb{R}^d$ (d est le nombre d'attributs des données d'entrée) et $y_i \in \{-1, 1\}$. Les étiquettes y_i ne peuvent prendre que deux valeurs (+1 ou -1), d'où l'appellation classification binaire.

Un perceptron binaire est un classifieur linéaire. A partir des données $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, l'objectif est d'apprendre un vecteur $\mathbf{w} \in \mathbb{R}^d$ tel que $\langle \mathbf{w}, \mathbf{x}_+ \rangle > 0$ pour tout $\mathbf{x}_+$ appartenant à la classe 1 et $\langle \mathbf{w}, \mathbf{x}_- \rangle \leq 0$ pour tout $\mathbf{x}_-$ appartenant à la -1. On rappelle que $\langle \mathbf{w}, \mathbf{x} \rangle = \sum_{j=1}^d w_j x_j$ est le produit scalaire entre les deux vecteurs $\mathbf{w}$ et $\mathbf{x}$, avec w_i et x_i les j-èmes composantes des vecteurs $\mathbf{w}$ et $\mathbf{x}$.

L'idée de l'algorithme du perceptron est d'initialiser $\mathbf{w}$ au vecteur nul, itérer un nombre de fois (fixé a priori ou jusqu'à convergence) sur les données d'apprentissage, et ajuster le vecteur de pondération $\mathbf{w}$ à chaque fois qu'une donnée est mal classée.

1. Implémenter en Python l'algorithme du perceptron décrit ci-dessus, sous forme d'une fonction prenant en paramètres l'échantillon de données, et renvoyant l'hyper-plan séparateur.
2. Soit la fonction `genererDonnees` ci-après qui permet de générer aléatoirement un jeu de données en 2 dimensions. Utiliser cette fonction pour générer un jeu de données d'apprentissage et un jeu de données test. Appliquer l'algorithme du perceptron sur le jeu de données apprentissage. Calculer l'erreur de prédiction du classifieur appris sur les jeux de données d'apprentissage et de test.
3. Représenter sur un graphique les données de test (utiliser deux couleurs différentes pour chaque classe) ainsi que le classifieur appris par le perceptron.

```
from pylab import rand

def genererDonnees(n):
 #générer un jeu de données 2D linéairement séparable de taille n.
 x1b = (rand(n)*2-1)/2-0.5
```

Algorithme Perceptron binaire (sans biais)

Entrée : une liste S de données d'apprentissage, $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ où $\mathbf{x}_i \in \mathbb{R}^d$ et $y_i \in \{-1, 1\}$, le nombre d'itérations N
Sortie : le vecteur de pondération w

---

1. Initialiser le vecteur $\mathbf{w} \leftarrow \mathbf{0}$
2. **Pour** iteration = 1 à N **faire**
3. **Pour** chaque exemple $(\mathbf{x}_i, y_i) \in S$ **faire**
4. Calculer la prédiction $\hat{y}_i = \text{sign}(\langle \mathbf{w}, \mathbf{x}_i \rangle)$
5. **Si** $\hat{y}_i \neq y_i$ **alors**
6. Ajuster $\mathbf{w}$: $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}_i$ si y_i est positive, ou $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{x}_i$ si y_i est négative
7. **Fin si**
8. **Fin pour**
9. **Fin pour**

```
x2b = (rand(n)*2-1)/2+0.5
x1r = (rand(n)*2-1)/2+0.5
x2r = (rand(n)*2-1)/2-0.5
donnees = []
for i in range(len(xb)):
 donnees.append(((x1b[i],x2b[i]),False))
 donnees.append(((x1r[i],x2r[i]),True))
return donnees
```

4. Le fichier `perceptron_data.py` contient un jeu de données mono-dimensionnelles simple nommé `data.bias`. Chaque exemple de ce jeu de données est constitué d'une valeur réelle positive associée à une étiquette binaire. Un classifieur linéaire peut s'écrire sous la forme $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$. Parce que le perceptron binaire de la question 1 ne considère pas le terme biais ($f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$ et $b = 0$), il ne sera pas en mesure de distinguer les deux classes du jeu de données bien que ce dernier soit linéairement séparable. En effet, puisque les données d'entrée sont des réels positifs, si le vecteur de pondération est positif, alors toutes les étiquettes seront marquées comme positives ; alors que si le vecteur de pondération est négatif, les étiquettes seront marquées comme négatives. Il est donc nécessaire d'augmenter la dimension des données d'entrée afin de permettre l'apprentissage d'un terme constant représentant le biais (il faut donc un échantillon complété et ajouter une dimension à l'hyperplan de décision).
   L'objectif ici est de modifier l'algorithme du perceptron binaire afin de tenir compte lors de l'apprentissage le biais du classifieur linéaire. Appliquer le programme modifié sur le jeu de données `data.bias` et montrer qu'il est capable de le séparer linéairement.
5. **Bonus pour ceux qui veulent comprendre la version duale du perceptron**. Vous pouvez très rapidement créer les fonctions d'apprentissage et de prédiction du perceptron dans sa forme duale, et les tester de la même façon que la forme primale.

2 Perceptron multi-classes

Un Perceptron multi-classes généralise le principe de classification linéaire du Perceptron binaire au cas où le nombre de classes peut être supérieur à deux dans l'espace des cibles (mais chaque exemple n'est étiqueté que par une seule classe – nous ne sommes pas en multi-labels). A partir d'un jeu de données $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, où maintenant $y_i \in \{l_1, \ldots, l_m\}$ et m est le nombre de classes, l'objectif est d'apprendre un *ensemble de vecteurs de pondération* $\{\mathbf{w}_{l_1}, \ldots, \mathbf{w}_{l_m}\}$ tel que la classe prédite par $\arg\max_{l_k} \langle \mathbf{w}_{l_k}, \mathbf{x} \rangle$ soit le plus souvent en accord avec la « vraie » classe y d'un exemple $\mathbf{x}$.

Algorithme Perceptron multi-classes

Entrée : une liste S de données d'apprentissage, $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ où $\mathbf{x}_i \in \mathbb{R}^d$ et $y_i \in \{l_1, \ldots, l_m\}$, le nombre d'itérations N.
Sortie : les vecteurs de pondération $\mathbf{w}_{l_1}, \ldots, \mathbf{w}_{l_m}$.

---

1. Initialiser les vecteurs $\mathbf{w}_{l_k} \leftarrow \not\vdash$ pour $k = 1, \ldots, m$
2. **Pour** iteration = 1 à N **faire**
3. **Pour** chaque exemple $(\mathbf{x}_i, y_i) \in S$ **faire**
4. Calculer la prédiction $\hat{y}_i = \arg\max_{l_k} \langle \mathbf{w}_{l_k}, \mathbf{x}_i \rangle$
5. **Si** $\hat{y}_i \neq y_i$ **alors**
6. Ajuster le score pour la *vraie* classe : $\mathbf{w}_{y_i} \leftarrow \mathbf{w}_{y_i} + \mathbf{x}_i$
7. Ajuster le score pour la classe prédite : $\mathbf{w}_{\hat{y}_i} \leftarrow \mathbf{w}_{\hat{y}_i} - \mathbf{x}_i$
8. **Fin si**
9. **Fin pour**
10. **Fin pour**

L'algorithme d'apprentissage pour ce problème est similaire à celui pour le cas binaire. Tous les vecteurs de pondération sont d'abord initialisés à zéro, puis plusieurs itérations sont effectuées sur les données d'apprentissage, avec ajustement des vecteurs de pondération chaque fois qu'une paire de données d'apprentissage est incorrectement étiquetée.

1. Implémenter en Python l'algorithme d'apprentissage du perceptron multi-classes au sein d'une fonction paramétrée judicieusement, qui serait la fonction `fit` de cet algorithme.
2. Ecrire une fonction permettant de prédire, à partir d'un perceptron multi-classes, la classe associée à une donnée d'entrée $\mathbf{x}_{test}$ (fonction `predict_example`)
3. Ecrire la fonction `predict` qui prédit, pour un tableau 2D de données en entrée, le tableau 1D de prédictions pour chacune de ces données avec le modèle préalablement entraîné.
4. Tester l'algorithme sur le jeu de données Iris : temps d'apprentissage, convergence, erreurs et matrices de confusion

3 A rendre le lundi 01/03/2021 – 23h55

Un rapport d'une à trois pages (format pdf) sur les expérimentations menées, qui indiquera les résultats expérimentaux obtenus (sous forme de tableaux et/ou de courbes), et une conclusion générale sur ce que vous avez observé. Le code de l'implémentation des algos doit être fourni en annexe.