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

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

15 mars 2023

Objectifs de la séance :
- pratiquer la génération de données artificielles pour étudier des algorithmes
- pratiquer les protocoles d'apprentissage dans le cadre multi-classes
- analyser et implémenter la variante de l'algorithme pour la classification multi-classes,
- tester les algorithmes sur des données simulées et des données réelles (convergence, performances)
- pratiquer l'écriture de fonctions Python simples.

1 Evaluation de l'apprentissage multi-classes

Ici, aucune implémentation d'algorithmes, de mesures, etc., n'est demandée, sauf si elle n'existe pas sur `sklearn`.

1.1 Génération de jeux de données artificiels

1. Comprendre et utiliser la fonction `make_classification` de `ssklearn` pour générer quatre jeux de données multi-classes, et créer une fonction pour visualiser les jeux de données S1 et S2 :
   (a) un jeu S1 avec 5 classes, 100 exemples décrits en 3 dimensions réelles, avec un peu de confusion entre les classes (trouver dans les paramètres de la fonction une façon d'apporter de la confusion, justifier).
   (b) un jeu S2 avec 5 classes, 100 exemples décrits en 3 dimensions réelles, sans aucune confusion entre les classes (trouver dans les paramètres de la fonction une façon d'apporter de la confusion, justifier).
   (c) S3 comme S1, mais exemples décrits en 10 dimensions
   (d) S4 comme S3, mais 1000 exemples.
2. Sans même lancer des apprentissages, ordonner ces jeux de données relativement aux performances qu'on pourrait attendre de classifieurs qui seraient appris sur eux (du meilleur classifieur au moins bon). Justifier.

1.2 Expérimentations du one versus all

L'objectif ici est de pratiquer une bon protocole expérimental de comparaison de modèles en classification multi-classes, quand le nombre de classes est 5.

1. Pour chacun de ces jeux de données, il s'agit de comparer les modèles appris en one-vs-all. Il faut (1) écrire la fonction permettant d'apprendre les 5 modèles (puisque one-vs-all avec 5 classes), (2) écrire la fonction permettant de prédire le tableau des classes pour un tableau d'exemples, par vote des 5 modèles, et (3) d'estimer les erreurs du classifieur final par 10 hold-out moyennés et intervalles de confiance de l'erreur estimée. On testera ces fonctions avec les arbres de décision et k plus proches voisins (hyper-paramètres par défaut).
2. Même chose en ajoutant la norme de la matrice de confusion moyenne sans diagonale (CMNwD) comme mesure de performance.

3. Maintenant, on va utiliser directement la version multi-classe des arbres de décision et des k plus proches voisins, de `sklearn`, et étudier l'évolution des performances MACC (score moyenné sur toutes les classes) selon le nombre d'exemples (de S4) à faire varier de 100 à 1000 par pas de 100, avec les intervalles de confiance. Produire un graphique pour visualiser cette évolution, où chaque mesure de performance sera estimée et encadrée par 10 hold-out moyennés. Pour quel nombre d'exemples a-t-on de meilleurs classifieurs ? Pour ce nombre d'exemples, indiquer le meilleur modèle avec intervalle de confiance.
4. Même chose, mais avec la CMNwD à la place du score.

2 Perceptron multi-classes

Algorithme Perceptron multi-classes

Entrée : une liste S de données d'apprentissage, $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ où $\mathbf{x}_i \in \mathbb{R}^{d+1}$ 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 $\mathbf{w}_{l_k} \leftarrow \mathbf{0}$ 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**

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 $\mathbf{x}$.

L'algorithme d'apprentissage pour ce problème est similaire à celui pour le cas binaire vu en cours. 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 (5 hold-out) l'algorithme sur les jeux de données S1 et S2 : temps d'apprentissage, score moyen et CMNwD, et produire une visualisation des séparateurs du nuage de points dans le cas de S1 et de S2.
5. Tester ensuite sur S3 en faisant varier le nombre d'exemples de 100 à 1000 par pas de 100. Observe-t'on une corrélation entre le score et la CMNwD ?
6. Tester sur le jeu de données digits et produire un rapport de classification.

3 A rendre le vendredi 17/03/2021 – 21h00

Un rapport d'une à trois pages (format pdf) sur les expérimentations menées, qui indiquera les résultats expérimentaux obtenus choisis (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.