IFT 3395 Fondements de l'apprentissage machine
Prof. Ioannis Mitliagkas

Devoir 2 - Partie pratique

- Ce devoir doit être fait et envoyé sur Gradescope. Pour IFT3395, il doit être fait en groupe de 2-3 étudiants. Pour IFT6390, il doit être fait en groupe de 2 étudiants. Vous pouvez discuter avec d'autres étudiants mais les réponses et le code que vous soumettez doivent être les vôtres. A noter que nous utiliserons l'outil de détection de plagiat de Gradescope. Tous les cas suspectés de plagiat seront enregistrés et transmis à l'Université pour vérification.

- La partie pratique doit être codée en python (avec les librairies numpy et matplotlib), et envoyée sur Gradescope sous fichier python. Pour permettre l'évaluation automatique, vous devez travailler directement sur le modèle donné dans le répertoire de ce devoir. Ne modifiez pas le nom du fichier ou aucune des fonctions signatures, sinon l'évaluation automatique ne fonctionnera pas. Vous pouvez bien sur ajouter de nouvelles fonctions et importations python

- Les figures, courbes et parties pratiques du rapport doivent être envoyées au format pdf sur Gradescope. Pour le rapport il est recommandé d'utiliser un Jupyter notebook, en écrivant les formules mathématiques avec MathJax et en exportant vers pdf. Vous pouvez aussi écrire votre rapport en LaTeX; LyX; Word. Dans tout les cas, exportez votre rapport vers un fichier pdf que vous enverrez. Vous êtes bien sur encouragés à vous inspirer de ce qui a été fait en TP.

- Vous pouvez générer des données synthétiques en utilisant la fonction Python suivante qui devrait être fournie : *generate_data*. Ce script crée un ensemble de données de mélange gaussien à 3 classes. Ne changez rien.

- Vous devez soumettre vos solutions sur Gradescope en utilisant le devoir intitulé `Devoir 2 - Pratique - 3395` pour le code et `Devoir 2 - Pratique - Rapport - 3395` pour le rapport.

Vous devez travailler sur le modèle `solution.py` du répertoire et compléter les fonctions basiques suivantes en utilisant numpy et python

Un-contre-tous, Perte L2 SVM

Cette partie consiste à implémenter le SVM un-contre-tous avec pénalité L2, qui est couramment utilisé pour faire de la classification multi-classe. La fonction de perte d'un SVM avec pénalité L2 est différentiable et impose une plus grande pénalité sur les points qui sont à l'intérieur de la marge maximale. Dans l'approche un-contre-tous (*one-versus-all* ou *OVA* en anglais), nous entraînons m classificateurs binaires, soit un pour chaque classe, et lors de la prédiction, nous sélectionnons la classe qui maximise la marge pour un point test.

Considérant un jeu d'entraînement $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, où $\mathbf{x}_i \in \mathbb{R}^p$, $y_i \in \{1, .., m\}$, p est le nombre de traits (ou attributs) et m est le nombre de classes, nous voulons minimiser la fonction objectif suivante:

$$\frac{1}{n} \sum_{(\mathbf{x}_i, y_i) \in S} \sum_{j'=1}^{m} \mathcal{L}(\mathbf{w}^{j'}; (\mathbf{x}_i, y_i)) + \frac{C}{2} \sum_{j'=1}^{m} \|\mathbf{w}^{j'}\|_2^2$$

où

$$\mathcal{L}(\mathbf{w}^{j'}; (\mathbf{x}_i, y_i)) = \Big(\max\{0, 2 - (\langle \mathbf{w}^{j'}, \mathbf{x}_i \rangle) \mathbb{1}\{y_i = j'\}\}\Big)^2$$

et

$$\mathbb{1}\{y_i = j'\} = \begin{cases} 1 & \text{if } y_i = j' \\ -1 & \text{if } y_i \neq j' \end{cases}$$

Afin de mettre à jour les paramètres $\mathbf{w}$ de notre fonction objectif, nous utiliserons des techniques de descente de gradient. (Note: Vous remarquerez que dans le jeu de données fourni pour cette partie, le dernier élément de chaque ligne est un 1. Cette notation permet de ne pas avoir de paramètre de biais b séparé, mais plutôt de l'inclure implicitement dans $\mathbf{w}$ comme étant un autre poids.)

Le fichier solution contient une fonction appelée `load_data()` pour lire et effectuer quelques transformations sur les données telle que la normalization. Pour que `load_data()` fonctionne bien, vous devez mettre le jeu de données

généré dans le même dossier que votre fichier `solution.py`. En résumé, vous devez avoir la structure de projet suivante:

```
[dossier du devoir]/
├── solution.py
├── Data_classification/
└── ...
```

1. [5 pts] Quelle est la dérivée du terme de régularisation de la fonction de perte

$$\frac{C}{2}\sum_{j'=1}^{m}\|\mathbf{w}^{j'}\|_2^2$$

par rapport à w_k^j? (le $k^{ième}$ poids du vecteur de poids pour la $j^{ième}$ classe)? Écrivez tous les étapes et mettez la réponse dans votre fichier PDF.

2. [10 pts] Quelle est la dérivée du terme appelé *hinge loss*

$$\frac{1}{n}\sum_{(\mathbf{x}_i,y_i)\in S}\sum_{j'=1}^{m}\mathcal{L}(\mathbf{w}^{j'};(\mathbf{x}_i,y_i))$$

de la fonction de perte par rapport à w_k^j?

Exprimez votre réponse en termes de $\mathbf{x}_{i,k}$ (la $k^{ième}$ entrée du $i^{ième}$ exemple $\mathbf{x}_i$).

Assumez que

$$\frac{\partial}{\partial a}\max\{0,a\}=\begin{cases}1 & \text{if } a>0\\ 0 & \text{if } a\leq 0\end{cases}$$

(Cette dernière affirmation n'est pas exactement vraie: à a=0, la dérivée n'est pas définie. Cependant, pour ce problème, nous allons assumer qu'elle est correcte.)

3. [30 pts] Complétez les méthodes suivantes dans le code:

   (a) [5 pts] Practical.make_one_versus_all_labels: Étant donné un tableau d'étiquettes qui sont des entiers et le nombre de classes m, cette fonction devrait retourner un tableau 2-d qui correspond

au terme $\mathbb{1}\{y_i = j'\}$ défini plus haut. Dans ce tableau, chaque ligne contient des -1 à l'exception de l'élément qui correspond à la bonne classe et qui devrait être un 1. Par exemple, si le tableau que l'on donne en entrée est $[1, 0, 2]$ et que $m = 4$, la fonction retournera le tableau suivant:$[[-1, 1, -1, -1], [1, -1, -1, -1], [-1, -1, 1, -1]]$. Les entrées de la fonction sont y (un tableau numpy de dimension (nombre de classes,)) et m (un entier représentant le nombre de classes), et la sortie devrait être un tableau numpy de dimension (nombre d'exemples, m). Pour ce devoir, m sera égal à 3, mais vous devriez implémenter cette fonction pour qu'elle puisse fonctionner avec n'importe quel $m > 2$.

(b) [5 pts] Practical.compute_loss : Étant donné un minibatch d'exemples, cette fonction devrait calculer la perte. Les entrées de la fonction sont x (un tableau numpy de dimension (minibatch size, 17)) et y (un tableau numpy de dimension (minibatch size, 3)) et la sortie devrait être la perte calculée, un scalaire.

(c) [10 pts] Practical.compute_gradient: Considérant un minibatch d'exemples, cette fonction devrait calculer le gradient de la fonction de perte par rapport au paramètre **w**. Les entrées de la fonction sont X (un tableau numpy de dimension (minibatch size, 17)) et y (un tableau numpy de dimension (minibatch size, 3)) et la sortie devrait être le gradient calculé, un tableau numpy de dimension $(17, 3)$, soit la même dimension que celle du paramètre **w**. (Indice: utilisez les expressions que vous avez dérivées précédemment.)

(d) [5 pts] Practical.infer: Étant donné un minibatch d'exemples, cette fonction devrait prédire la classe de chaque exemple, c'est-à-dire la classe qui a le plus haut score. L'entrée de la fonction est X (un tableau numpy de dimension (minibatch size, 17)) et la sortie est *y_inferred* (un tableau numpy de dimension (minibatch size, 3)). La sortie devrait être en format un-contre-tous, c'est-à-dire -1 pour les classes qui ne sont pas prédites et +1 pour la classe prédite.

(e) [5 pts] Practical.compute_accuracy: Étant donné un tableau de classes prédites et un tableau des vraies classes, cette fonction devrait retourner la proportion de classifications correctes, soit un scalaire entre 0 et 1. Les entrées de cette fonction sont *y_inferred*

(un tableau numpy de dimension (minibatch size, 3)) et y (un tableau numpy de dimension (minibatch size, 3)) et la sortie est un scalaire.

4. [5 pts] La méthode Practical.fit utilise le code que vous avez écris ci-dessus pour entraîner le SVM. Après chaque époque (après avoir passer à travers tous les exemples du jeu de données), SVM.fit calcule la perte et l'exactitude des points d'entraînement et la perte et l'exactitude des points tests.

   Faites le graphique de ces quatre quantités en fonction du nombre d'époques, pour $C = 1, 5, 10$. Utilisez comme hyperparamètres 200 époques, un taux d'apprentissage de 0.0001 et une longueur de minibatch de 100.

   Vous devriez avoir 4 graphiques, soit un graphique pour chaque quantité, incluant les courbes pour les 3 valeurs de C. Ajoutez ces 4 graphiques dans votre rapport.

   Sur la base de ces graphiques, le surapprentissage semble-t-il être un problème pour ce jeu de données et cet algorithme? Expliquez brièvement.