IFT615 : Intelligence Artificielle

Université de Sherbrooke

Été 2022

TP1

Date limite	24 mai 2022 à 23 :59
Mode de soumission	Turnin TP1
Total de points	50
Pondération	8%
Email du correcteur	jean-charles.verdier@usherbrooke.ca
Fichiers à soumettre	`models.py`, `q1-calculs.{png, jpeg, pdf}`, `q1-b.{png, jpeg, pdf}`

Tout retard vaudra 0. Le nom respect des noms de fichiers à soumettre peut entrainer des pénalités, allant jusqu'à une note de 0.

Introduction et rappel théorique

Dans ce travail pratique, vous êtes invité à implémenter un modèle de classification linéaire appelé le perceptron, et le tester sur des données. Un perceptron peut classifier deux ou plusieurs classes à partir d'une matrice de données $\mathbf{X} \in \mathbb{R}^{N \text{x} D}$ où N représente le nombre d'observations et D dénote le nombre d'attributs de chaque observation. Dans le cas binaire, un perceptron recherche un vecteur de poids $\boldsymbol{w}^*$ et un biais b^* qui minimisent les erreurs de classification. Les classes à prédire sont 0 et 1. La prédiction $\hat{y}$ associée à une donnée quelconque $\boldsymbol{x_i} \in \mathbf{X}$ est effectuée à partir des formules suivantes :

$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x_i}) = Threshold\Big(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x_i} + b\Big) = Threshold\Big(\sum_{j=1}^{D}(w_j x_j) + b\Big)$$

avec la fonction $Threshold$ définie comme suit :

$$Threshold(x) = \mathbb{1}_{[x \geq 0]}(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Lors d'une erreur de classification, c'est-à-dire lorsque $\hat{y}_i \neq y_i$, on doit mettre à jour les paramètres $\boldsymbol{w}$ en additionant ou en soustrayant $\boldsymbol{x_i}$ qu'on pondère par le taux d'apprentissage α. Formellement, la mise à jour des paramètre $\boldsymbol{w}$ et b à l'itération t est donnée par

$$\boldsymbol{w}^{(t)} = \boldsymbol{w}^{(t-1)} + \alpha(y_i - \hat{y}_i)\boldsymbol{x_i}$$
$$b^{(t)} = b^{(t-1)} + \alpha(y_i - \hat{y}_i)$$

Ces étapes sont résumées dans l'algorithme 1 ci-dessous :

Algorithme 1 Perceptron

procedure PERCEPTRON($\mathcal{D}, nIter, \alpha$) ▷ Les données étiquetées, le nombre d'itérations, le taux d'apprentissage
  $\boldsymbol{w} \leftarrow \begin{pmatrix} 0.0 & \dots & 0.0 \end{pmatrix}$ ▷ Initialisation des poids à zéro
  $b \leftarrow 0.0$ ▷ Initialisation du biais à zéro
  for $i \dots nIter$ **do**
    for $\boldsymbol{x_i}, y_i \in \mathcal{D}$ **do**
      $score \leftarrow \boldsymbol{w} \cdot \boldsymbol{x_i}^{\mathrm{T}} + b$
      $\hat{y}_i \leftarrow Threshold(score)$
      if $\hat{y}_i \neq y_i$ **then**
        $b \leftarrow b + \alpha(y_i - \hat{y}_i)$
        $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha(y_i - \hat{y}_i)\boldsymbol{x_i}$
      end if
    end for
  end for
end procedure

Installation

Les travaux du cours IFT615 sont tous effectués à l'aide de `Python 3.9` et requièrent les dépendances Numpy, Matplotlib et Scikit-Learn. Le fichier `requirements.txt` est utilisé afin d'installer les dépendances du projet. Pour les installer, nous vous recommendons fortement d'utiliser un environnement virtuel à l'aide de conda ou de pip. La manière la plus simple revient à utiliser pip. Exécutez les commandes suivantes à partir de la racine du dossier du TP :

```
python -m venv env
pip install matplotlib scikit-learn numpy
```

Sur Windows, remplacez `source env/bin/activate` par `./env/Scripts/activate` à partir de Powershell. Ces commandes permettent de créer et d'activer un nouvel environnement virtuel tout en installant les dépendances du projet. Vous devez toutefois vous assurez d'utiliser une version Python supérieure ou égale à 3.9.

Alternativement, conda permet d'installer des environnements pour des versions spécifiques de Python. Ainsi, pour créer un nouvel environnement à l'aide de conda qui utilise `Python 3.9`, exécutez la commande suivante.

```
conda create --name ift615 python=3.9
```

Vous pouvez ensuite activer l'environnement créé en exécutant :

```
conda activate ift615
```

Vous devez finalement installer les dépendances :

```
conda install scikit-learn
conda install matplotlib
```

Une feuille de raccourcis permettant de se familiariser avec conda se trouve ici. Un excellent tutoriel sur `Numpy` est offert sur ce lien. Finalement, vous pouvez tester l'installation des dépendances à l'aide de la commande :

```
python autograder.py --check-dependencies
```

Auto-correcteur

Un auto-correcteur est fourni afin d'évaluer votre code. Le résultat affiché ne garantit toutefois pas la note finale. Il indique seulement si le code fonctionne ou non. Pour vérifier votre code sur toutes les questions, exécutez

```
python autograder.py
```

N'oubliez pas d'activer votre environnement virtuel sans quoi cette commande risque d'échouer. Vous pouvez optionnellement spécifier la question à valider à l'aide du flag `-q`. Par exemple, pour tester seulement la première question :

```
python autograder.py -q q1
```

Finalement, certaines questions affichent des graphiques pour faciliter votre développement. Vous pouvez en tout temps arrêter cet affichage à l'aide du flag `-no-graphics`.

```
python autograder.py --no-graphics
```

Question 1 : Perceptron binaire (24 pts)

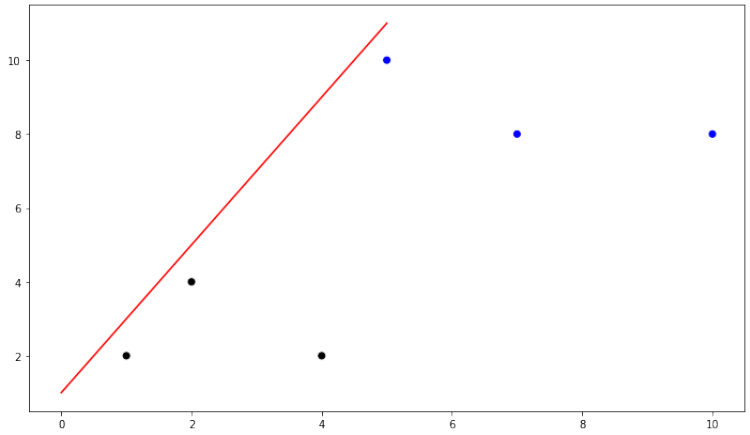
a) Mise à jour manuelle du perceptron (6 pts)

À partir du tableau ci-dessous, exécutez une itération complète de l'algorithme du perceptron avec un taux d'apprentissage $\alpha = 0.1$, un vecteur de poids nul $\boldsymbol{w} = \begin{pmatrix} 0.0 & 0.0 \end{pmatrix}$ et avec un biais $b = 0.0$. Donnez la valeur des poids $\boldsymbol{w}$ et du biais b après chaque donnée parcourue. Prenez en photo vos calculs et incluez les dans un fichier nommé `tp1-calculs`. Les formats acceptés sont `png`, `jpeg` et `pdf`.

x_1	x_2	classe
1	1	0
2	2	0
4	4	1
5	4	1
5	5	1

b) Visualisation de la barrière de décision (5 pts)

Il est souvent utile de visualiser le critère de décision d'un modèle d'apprentissage automatique. Cela nous permet notamment d'évaluer visuellement la capacité de généralisation et la qualité de notre modèle. Supposez que les paramètres optimaux du perceptron sont $\boldsymbol{w}^* = \begin{pmatrix} 0.0001 & 0.0001 \end{pmatrix}$ et $b^* = -0.0006$. Dessinez un nuage de points avec les données du tableau ci-dessus et tracez la droite représentant la barrière de décision. Dessinez le graphique sur une feuille et prenez-la en photo ou utilisez un logiciel pour produire la droite et le nuage de points. Nommez le fichier `q1b.png` et ajoutez le à votre soumission. Prenez en photo vos calculs et incluez les dans le fichier `tp1-calculs`. Les formats de fichiers acceptés sont `png`, `jpeg` et `pdf`. Voici un exemple de solution avec des données différentes :



Conseils

- Définissez l'équation de la droite et isolez en fonction de x_1 et x_2. Déterminer l'ordonnée et l'abscisse à l'origine et finalement trouver la pente de la droite.
- Je vous recommande d'utiliser les logiciels Geogebra, Desmos ou encore la librairie Matplotlib (si vous êtes à l'aise à Python) afin de produire le nuage de points.

c) Implémentation informatique du perceptron binaire (13 pts)

Maintenant que vous maîtrisez la procédure de mise à jour du perceptron, vous pouvez implémenter l'algorithme complet. Un squelette de code est déjà fourni. Vous devez le compléter.

À réaliser

Complétez les fonctions `init_params`, `threshold`, `predict` et `fit` de la classe `BinaryPerceptron` à partir du fichier `models.py`.

Auto-correcteur

Pour évaluer votre code, vous pouvez utiliser l'auto-correcteur automatique à l'aide de la commande

```
python autograder.py -q q1
```

Critères d'évaluation

La librairie `Numpy` profite d'accélération matérielle afin de rendre les calculs mathématiques très rapides. Pour cette raison, vous êtes encouragé à utiliser son API et éviter les boucles et itérations qui ralentissent significativement l'exécution de votre code. Un excellent tutoriel sur Numpy est offert sur ce lien.

Critères	Points
`fit`	5
`predict`	3
`init_params`	2
`threshold`	1
aucune boucle/itération	1
qualité / lisibilité du code	1
Total	13

Question 2 : Perceptron multi-classe (13pts)

Nous avons traité le cas du perceptron binaire. Nous pouvons maintenant généraliser son application à plusieurs classes. Dans cet exercice, nous entraînons un perceptron à reconnaître des chiffres manuscrits. Pour ce faire, nous utilisons la célèbre base de données NIST (à ne pas confondre avec MNIST) qui contient 1797 exemples de chiffres manuscrits en 16 tonalités de gris. Chaque image est de taille $(8, 8)$ et par conséquent nos données sont de la forme $\mathbf{X} \in \mathbb{R}^{1797\text{x}64}$.

Le perceptron multiclasse se comporte de manière similaire au perceptron binaire, mais utilise plutôt un vecteur de poids par classe, un vecteur pour le biais et un critère de décision différent. Formellement, soit C le nombre de classes distinctes à prédire, $\mathbf{W} \in \mathbb{R}^{C\text{x}D}$ la matrice des poids et $\boldsymbol{b} \in \mathbb{R}^{C}$ le vecteur de biais. L'objectif du percetron multiclasse est de trouver la matrice $\mathbf{W}^*$ et le vecteur $\boldsymbol{b}^*$ qui minimisent les erreurs de classification.

Pour le critère de décision, on abandonne la fonction *Threshold* et nous choisissons plutôt la classe associée aux poids $\mathbf{W}_c$ qui génèrent le plus grand score, c'est-à-dire

$$\hat{\boldsymbol{y}} = \underset{\boldsymbol{w} \in \mathbf{W}}{\operatorname{argmax}}\, \phi_{\mathbf{W}}(\mathbf{X})$$

La logique du score demeure la même, mais on effectue maintenant un produit matriciel plutôt qu'un produit vectoriel.

$$\phi_{\mathbf{W}}(\mathbf{X}) = \mathbf{X}\mathbf{W}^T + \boldsymbol{b}$$

Lors d'une itération sur une donnée $\boldsymbol{x_i} \in \mathbf{X}$, on compare l'étiquette y avec sa prédiction $\hat{y}$. Si $y = \hat{y}$, on ne fait rien. Sinon, on doit mettre à jour les poids correspondant à y et $\hat{y}$ afin d'éviter prévenir cette erreur de classification.

$$\begin{aligned}
\mathbf{W}_{\hat{y}} &= \mathbf{W}_{\hat{y}} - \alpha \mathbf{X}_i \\
\boldsymbol{b}_{\hat{y}} &= \boldsymbol{b}_{\hat{y}} - \alpha \\
\mathbf{W}_{y} &= \mathbf{W}_{y} + \alpha \mathbf{X}_i \\
\boldsymbol{b}_{y} &= \boldsymbol{b}_{y} + \alpha
\end{aligned}$$

À réaliser

Dans le fichier `models.py`, implémentez les méthodes `init_params`, `predict` et `fit` de la classe `MulticlassPerceptron`.

Critères d'évaluation

Critères	Points
`fit`	5
`predict`	4
`init_params`	2
aucune boucle/itération	1
qualité / lisibilité du code	1
Total	13

Auto-correcteur

Pour évaluer votre code, vous pouvez utiliser l'auto-correcteur automatique à l'aide de la commande

```
python autograder.py -q q2
```

Question 3 : Génération de caractéristiques discriminantes (13 pts)

La qualité de votre classifieur dépend largement de la qualité des données qu'il reçoit en entrée. Il est souvent possible d'améliorer les performances d'un modèle simplement en manipulant les données sans introduire de biais ou d'effet Hans le Malin. Il existe en fait une littérature riche dans le domaine du traitement des données (feature engineering en anglais).

Vous êtes donc invité dans cet exercice à modifier les données originales afin d'améliorer les performances du perceptron. Vous devez justifier chacune des modifications apportées en commentant la méthode `preprocess`. Expliquer clairement pourquoi votre technique devrait augmenter la performance du modèle. La majorité des points sont accordés aux explications et non pour les résultats obtenus.

Afin de vous aider dans cette recherche, vous pouvez observer les erreurs de classification des modèles lors de l'entraînement. Déterminez, par exemple, si une classe est surepresentée dans les erreurs et pourquoi. Il est permis de s'inspirer d'articles trouvés sur le web ou dans la littérature, mais il faut absolument laisser une référence dans le code.

À réaliser

Dans le fichier `models.py`, implémentez la fonction `preprocess` de la classe `FeatureEngPerceptron`. Ajoutez des commentaires dans l'en-tête de la méthode `preprocess` afin d'expliquer votre intuition. Laissez aussi les sources (liens vers les articles) de vos inspirations à cet endroit.

Critères d'évaluation

Critères	Points
`preprocess`	5
justification et intuition	5
amélioration performance	2
qualité / lisibilité du code	1
Total	13

Auto-correcteur

Pour évaluer votre code, vous pouvez utiliser l'auto-correcteur automatique à l'aide de la commande

```
python autograder.py -q q3
```