TP4 (comptant pour 10%)

IFT-4102 et IFT-7025 : Techniques avancées en Intelligence Artificielle

À rendre avant le 1 Mai 2022 à 23h59

Introduction

Dans le TP3, nous avons couvert deux techniques d'apprentissage automatique, les K-Plus Proches Voisins et la classification Naive Bayésienne. Dans ce TP4, nous allons voir deux autres techniques plus élaborées et plus performantes, les Arbres de Décision (Decision Trees) et les Réseaux de Neurones (Neural Networks).

Pour toute aide veuillez vous adresser à Mathieu Pagé-Fortin.

Datasets

Comme pour le TP3, vous allez tester votre code sur les datasets Iris, Binary Wine quality et Abalones.

Méthodes à développer

Cette section décrit les méthodes (ou techniques) que vous aurez à implémenter pour mettre en œuvre et tester les datasets fournis.

Notes :

→ Partagez vos données sur deux ensembles, un ensemble d'entraînement et un ensemble de test. C'est à vous de choisir le ratio train/test (70/30 est recommandé).

→ Dans ce qui suit, la courbe d'apprentissage et la validation croisée se font seulement sur l'ensemble d'entraînement. Cet ensemble d'entraînement sera encore divisé en deux ensembles : un pour réellement entraîner le modèle et l'autre pour faire la validation. L'ensemble test est utilisé seulement pour faire le test final de votre modèle.

→ Il est fortement suggéré de vous référer à cette vidéo [1] qui explique bien l'intuition derrière l'utilisation des ensembles d'entraînement, de validation et de test, ainsi que la courbe d'apprentissage. Elle explique également la recherche d'hyperparamètres en grille (*Grid search*) mais nous utiliserons une forme plus simple de recherche pour ce TP.

1. https://www.youtube.com/watch?v=w_bLGK4Pteo&ab_channel=MachineLearnia

1 Arbres de Décision

Pour cette méthode, vous allez mettre en œuvre un arbre de décision qui divise selon l'attribut qui offre le gain d'information maximal comme vu dans le cours et décrit dans *la section 18.3* du manuel. ***Les questions b) et c) sont à faire pour chaque dataset.***

a) Implémentation. Créez une classe qui implémente un arbre de décision à partir du squelette de code fourni dans le fichier `classifieur.py` (sans utiliser de bibliothèques de machine learning).

b) Entraînement. Utilisez le fichier `load_datasets.py`, que vous avez développé dans le TP3 pour lire les datasets. Pour chaque dataset, entraînez votre modèle avec les données d'entraînement.

La courbe d'apprentissage est souvent utile pour faire le *sanity-check* de votre algorithme d'apprentissage. Autrement dit, elle vous permet de vérifier si votre algorithme est en train d'apprendre correctement à partir des exemples que vous lui présentez et à quelle vitesse. Tracez la courbe d'apprentissage comme décrit dans le dernier paragraphe du *chapitre 18.3.3* du manuel (les courbes devraient ressembler à la figure 18.7 du manuel). **Commentez le résultat**.

c) Évaluation. Évaluez votre modèle sur l'ensemble de test en donnant pour chaque classe :
- L'exactitude (Accuracy)
- La précision (Precision)
- Le rappel (Recall)
- Le F1-score
- La matrice de confusion (Confusion matrix)

d) Visualisation. Tracez l'arbre de décision obtenu avec le jeu de données Iris.

e) Scikit-learn. Utilisez maintenant la classe de `scikit-learn`[2] pour entraîner un arbre de décision. Comparez les résultats avec ceux obtenus avec votre implémentation.

Question pour IFT-7025 seulement : Mettez en œuvre l'élagage tel que décrit dans la *section 18.3.5* du manuel et refaites les questions b) et c). Comparez les résultats avec et sans élagage.

2 Réseaux de Neurones Artificiels

Pour cette méthode, vous allez implémenter un réseau de neurones (**RN**) tel que décrit dans la *section 18.7* du manuel. Assurez vous d'utiliser au moins une couche cachée parce qu'un **RN** avec seulement deux couches (couche d'entrée et couche de sortie) n'est qu'un classificateur linéaire.

a) Implémentation. Créer une classe qui implémente un **RN** à une couche cachée à

2. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn-tree-decisiontreeclassifier

partir du squelette de code fourni dans le fichier `classifieur.py` (sans utiliser des bibliothèques de machine learning).

b) Initialisation des poids du réseau de neurones. L'initialisation de tous les poids du **RN** à la même valeur (zéro ou autre) n'est pas une bonne pratique. Il existe plusieurs techniques d'initialisation dans la littérature, faites alors une recherche rapide sur ces techniques et choisissez en une qui vous convient.

i) Expliquez brièvement la technique que vous avez choisie.

ii) Pour comprendre l'effet de l'initialisation des poids d'un **RN**, tracez pour chaque jeu de données la courbe d'apprentissage d'un **RN** initialisé avec des poids de zeros et celle d'un **RN** initialisé avec la technique que vous avez choisie. Commentez le résultat.

b) Entraînement. Pour chaque dataset, choisissez arbitrairement les hyperparamètres (nombre de neurones, nombre d'itérations d'entraînements, etc.) et entraînez votre modèle avec les données d'entraînement.

c) Évaluation. Évaluez votre modèle sur l'ensemble de test en donnant pour chaque classe :
- L'exactitude (Accuracy)
- La précision (Precision)
- Le rappel (Recall)
- Le F1-score
- La matrice de confusion (Confusion matrix)

2.1 Recherche d'hyperparamètres

Les hyperparamètres qui définissent l'architecture d'un Réseaux de Neurones (noté **RN**) jouent un grand rôle dans la performance de l'algorithme. Il existe plusieurs façons empiriques pour choisir ces hyperparamètres. Pour ce TP, utilisez la validation croisée (k-fold cross validation) pour choisir le nombre de neurones (la dimension) et le nombre de couches cachées (la profondeur) pour chaque dataset. Voici brièvement comment procéder :

→ Vous divisez vos données d'entraînement en k parties de taille égale (appelées folds). Choisissez k entre 5 et 10.
→ Vous choisissez un nombre approprié de valeurs candidates pour l'hyperparamètre.
→ Pour chacune de ces valeurs candidates, entrainez le **RN** k fois, en utilisant à chaque fois $k-1$ parties comme données d'entraînement et la k^{eme} comme données de validation. Notez l'erreur moyenne pour chacune des dimensions.
→ Vous choisissez la valeur d'hyperparamètre dont l'erreur moyenne de validation est la plus faible.
→ Faites ça pour chaque dataset, et notez la meilleure architecture pour chacun.

Note : Pour cette section, vous pouvez utiliser le réseau de neurone de scikit-learn [3] mais vous devez implémenter la méthode de recherche d'hyperparamètres vous-mêmes.

3. https://scikit-learn.org/stable/modules/neural_networks_supervised.html

2.1.1 Choix du nombre de neurones dans la couche cachée

Explorez par validation-croisée au moins cinq valeurs pour l'hyperparamètre du nombre de neurones pour un **RN** à une couche cachée.

1. Pour chaque dataset, tracez la courbe [Erreur Moyenne (sur l'axe des y)/Nombre de neurones dans la couche cachée (sur l'axe des x)]. Commentez chacun des tracés.
2. Quelle dimension choisissez-vous pour chaque dataset ?

2.1.2 Choix du nombre de couches cachées

Dans un Réseau de Neurones nous pouvons avoir autant de couches cachées que l'on veut. Utilisez la validation croisée pour comparer cinq (ou plus si vous voulez) architectures de **RNs** : un avec 1 couche cachée (comme celui de la question précédente) soit `RN-1C`, un avec 2 couches cachées soit `RN-2C`, et ainsi de suite jusqu'à `RN-5C`. Gardez le même nombre de neurones dans les couche cachée que vous avez trouvé dans la questions précédente.

1. Pour chaque dataset, tracez la courbe [Erreur Moyenne (sur l'axe des y)/**Nombre de couches cachées** (sur l'axe des x)]. Commentez chacun des tracés.
2. Quel modèle choisiriez-vous pour chacun des datasets ?
3. Expliquez le phénomène de *Vanishing Gradient*. L'observez-vous dans vos expérimentations ?

2.1.3 Entraînement et Test

Notez bien que vous n'avez pas touché aux données de test jusqu'ici, vous avez sélectionné la meilleure architecture à l'aide des données d'entraînement et de la validation croisée seulement. Maintenant que vous avez défini, pour chaque dataset, une architecture appropriée et une technique d'initialisation des poids, entraînez votre modèle avec les données d'entraînement et testez le avec les données de test, et donnez :

- L'exactitude (Accuracy)
- La précision (Precision)
- Le rappel (Recall)
- Le F1-score
- La matrice de confusion (Confusion matrix)

Note importante : l'entraînement d'un Réseau de Neurones peut nécessiter plusieurs époques d'entraînement. Cela veut dire que vous pouvez entraîner le modèle plusieurs fois sans ré-initialisation avec le même ensemble d'entraînement, en mélangeant les exemples dans chaque époque. Mais cela peut induire un sur-apprentissage. Pour palier à ça, vous pouvez utiliser la validation croisée pour choisir le nombre d'époques approprié pour chacun des réseaux.

2.2 Comparaison entre les algorithmes expérimentés dans le cours

Comparez dans un tableau les performances des Réseaux de Neurones, des Arbres de Décision, des K-Plus Proches Voisins et de la classification Bayésienne Naïve selon :

- L'exactitude sur l'ensemble de test

— Temps de prédiction d'un seul exemple
— Temps d'apprentissage du modèle

Dans une conclusion, faites un récapitulatif de ce que vous appris dans ce projet.

3 Directives

Commencez par voir et examiner le code (dans le dossier `Code`) qui vous est donné en attaché. Implémentez les deux techniques en suivant le squelette des classes tel qu'indiqué dans les fichiers en attaché.

— Utilisez le fichier `load_datasets.py` que vous avez développé dans le TP3 pour lire les datasets.
— Utilisez le squelette de code fournit dans `classifieur.py` pour implémenter les deux algorithmes. Nommez le fichier selon la technique : `NeuralNet.py` pour le modèle *Réseau de Neurones* et `DecisionTree.py` pour le modèles des *Arbres de Décision*.
— Complétez le fichier `entrainer_tester.py` pour lancer les expérimentations, l'entraînement et le test de vos techniques, c'est le fichier principal pour l'exécution.

4 Livrables

Le travail doit être rendu dans un dossier compressé (.zip), contenant :

— README [Optionnel] : un fichier texte contenant une brève description des classes, la répartition des tâches de travail entre les membres d'équipe, et une explication des difficultés rencontrées dans ce travail.
S'il y a des erreurs ou des bogues dans votre code, mentionnez les, et expliquez les démarches que vous avez prises pour déboguer le code (cela vous permettra d'avoir des points même si votre code n'a pas fonctionné)
— Tout le code que vous avez écrit doit être remis dans le dossier `Code`, vous avez le droit de modifier le code que nous vous avons fournis, mais les noms des méthodes (tels que : `train(...)`, `predict(...)`, `evaluate(...)`, ...etc) doivent rester inchangés.
— Un document PDF contenant :
  — Les réponses aux questions
  — Les discussions des résultats obtenus
  — Une comparaison entre les deux techniques en terme de performances tel que demandé dans la sous-section 2.2.