Projet Final

IFT-4102 et IFT-7025 : Approche Agent en Intelligence Artificielle

À rendre avant le 26 Avril 2017 à 23h55

Introduction

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

1 Datasets

Comme pour la première partie du projet, vous allez tester votre code sur les datasets Iris, Monks et Congressional.

2 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'entrainement et un ensemble de test. C'est à vous de choisir le ration train/test (70/30 est recommandé), sauf pour le dataset Monks, le train et test sont déjà séparés.

→ Dans ce qui suit, la courbe d'apprentissage et la validation croisée se font seulement sur l'ensemble d'entrainement. L'ensemble test est utilisé seulement pour faire le test final de votre modèle.

→ Assurez vous de garder les mêmes notations adoptée dans le projet partiel, `train(...)`, `predict(...)`, `test(...)`. Vous pouvez rajouter autant de paramètres que vous voulez pour ces méthodes. Vous pouvez aussi rajouter d'autres méthodes au besoin, assurez vous juste que l'entrainement du modèle est fait dans la méthode `train(...)` et le test dans la méthode `test(...)`.

2.1 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.

2.1.1 Implémentation

Implémentez cette technique (sans utiliser des bibliothèques de machine learning). ***Les questions suivantes sont à faire pour chaque dataset.***

2.1.2 Courbe d'apprentissage

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 entrain d'apprendre à partir des exemples que vous lui présenter et à quelle vitesse.

— Tracez la courbe d'apprentissage pour cet algorithme comme décrit dans le dernier paragraphe du *chapitre 18.3.3* du manuel (La courbe va ressembler à la figure 18.7 du manuel). **Commentez le résultat**.

2.1.3 Entrainement et Test

Entrainez votre modèle avec les données d'entrainement. Ensuite testez le avec les données de test, et :

— Donnez la matrice de confusion (pas la peine de donner la précision et le rappel, on sait comment les dériver de cette matrice)

— Donnez l'Accuracy.

Question optionnelle : Mettre en œuvre l'élagage tel que décrit dans la *section 18.3.5* du manuel. Refaire les questions 2 et 3. Ensuite comparer les résultats avec et sans élagage.

2.2 Réseaux de Neurones Artificiels

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

2.2.1 Implémentation

Implémentez cette technique (sans utiliser des bibliothèques de machine learning) et assurez vous que votre code prend en paramètre le nombre de couches du réseau et le nombre de neurones dans chaque couche. Vous en aurez besoin pour faire des tests par la suite.

2.2.2 Choix du nombre de neurones dans la couche cachée

L'architecture d'un Réseaux de Neurones (noté **RN**) joue un grand rôle dans la performance de l'algorithme, le nombre de neurones dans la couche cachée est l'une des plus importante caractéristiques. Il existe plusieurs façons empiriques pour choisir ce nombre. Pour votre cas vous allez utiliser la validation croisée (k-fold cross validation) pour choisir le nombre de neurones (la dimension) 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 dimensions (nombre de neurones) candidates pour votre couche cachée. Choisissez $[4, 5, 6, 7, 8, 9, ..., 50]$. (vous pouvez tester pour d'autres nombres)
→ Pour chacune de ces dimensions candidates, entrainez le **RN** k fois, en utilisant $k-1$ parties comme données d'entraînement et le k^{eme} comme données de test. Notez l'erreur moyenne pour chacune des dimensions.
→ Vous choisissez le nombre de neurones dont l'erreur moyenne de test est la plus faible.
→ Faites ça pour chaque dataset, et notez la meilleure architecture pour chacun.

Questions :

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 architecture choisissez-vous pour chaque dataset ?

Note : 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'entrainement et la validation croisée seulement.

2.2.3 Choix du nombre de couches cachées

Dans un Réseau de Neurones nous pouvons avoir autant de couches cachées que l'on veut. Pour comprendre l'effet de la profondeur d'un **RN**, vous allez comparez cinq (ou plus si vous voulez) architectures de **RNs** : un avec 3 couches (comme celui de la question précédente) soit `RN-3C`, un avec 4 couches (une couche d'entrée, deux couches cachées et une couche de sortie) soit `RN-4C`, ainsi de suite, jusqu'à `RN-7C`. (Pour chaque dataset, gardez le même nombre de neurones dans les couche cachée que vous avez trouvé dans la questions précédente).

Pour chaque dataset :

1. Tracez dans la même figure la courbe d'apprentissage des cinq **RNs**.
   — Commentez les résultats
   — Quel modèle choisiriez-vous pour chacun des dataset ?
   — Googlez le terme *Vanishing Gradient*.
2. Faites la validation croisée comme dans 2.2.2 pour choisir le modèle approprié pour chaque dataset.

2.2.4 Initialisation des poids du Réseau de Neurones

L'initialisation de tous les poids du Réseau de Neurones 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.

1 - Expliquez brièvement la techniques que vous avez choisie.

Pour comprendre l'effet de l'initialisation des poids d'un **RN** vous allez comparer deux **RNs** pour chaque dataset. Appelons votre **RN** avec l'initialisation de poids à zéro `RN-ZERO` et celui avec l'initialisation que vous avez choisie `RN-NON-ZERO`.

Question : pour chaque dataset

— Tracez dans la même figure la courbe d'apprentissage de RN-ZERO et celle de RN-NON-ZERO. Commentez le résultat.

2.2.5 Entrainement et Test

Maintenant que vous avez défini, pour chaque dataset, une architecture appropriée, et une technique d'initialisation des poids, entrainez votre modèle avec les données d'entrainement et testez le avec les données de test, et :

— Donnez la matrice de confusion (pas la peine de donner la précision et le rappel)

— Donnez l'Accuracy.

Note importante : l'entrainement d'un Réseau de Neurones peut nécessiter plusieurs époques d'entrainement. Cela veut dire que vous pouvez entrainer le modèle plusieurs fois sans ré-initialisation avec le même ensemble d'entrainement, 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.

2.3 Comparaison entre Réseaux de Neurones et Arbres de Décision

Comparez dans un tableau, les performances des Réseaux de Neurones contre les Arbres de Décision selon :

— Accuracy ou Taux d'erreur sur l'ensemble de test

— Temps de prédiction d'un seul exemple

— Temps d'apprentissage du modèle

Dans une conclusion, faites un récapitulatifs 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. Implémenter les deux techniques en suivant le squelette des classes tel qu'on vous l'a indiqué dans les fichier python en attaché.

— Utilisez le fichier `load_datasets.py`, que vous avez développé dans la première partie du projet pour lire les datasets.
— Lisez bien le fichier `classifieur.py` (nous l'avons un peu modifié par rapport à la première partie du projet) pour vous aider à implémenter les deux techniques d'apprentissage machine, nommez le fichier selon la technique : `NeuralNet.py` pour le modèle *Réseau de Neurones* et `DecisionTree.py` pour le modèles des *Arbres de Décision*.
— Compléter le fichier `main.py` pour lancer les expérimentations, l'entraînement et le test de vos techniques, c'est le fichier principal pour l'exécution. (c'est le même que le fichier `entrainer_tester.py` de la première partie)

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(...)`, `test(...)`, ...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 2.3