INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Rapport TP Machine Learning

Carrel Nirina et Antoine Gruet

5 AE - SIEC

I. TP1 - Apprentissage supervisé par la méthode des plus proches voisins

A. Prise en main de la librairie scikit-learn

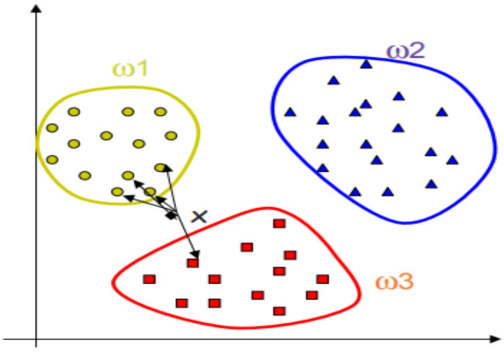
La librairie scikit-learn fournit plusieurs jeux de données. Dans la suite de ce TP, on utilisera l'ensemble mnist_784 pour cela, on charge le jeux de données avec la commande suivante : mnist = fetch_openml('mnist_784')

La variable retournée et stockée dans "mnist" est une classe contenant :

- les images en 28x28, elles sont stockées dans un tableau accessible de cette façon : "mnist.data"
- les classes correspondant aux images, elles sont stockées dans un tableau accessible de cette façon : "mnist.target"

Ainsi on peut effectuer des prédictions sur les images et vérifier si la prédiction faite correspond bien au chiffre contenu dans target.

B. Méthode des k plus proches voisins



Pour un nouvel exemple non étiqueté x, trouver les k "plus proches exemples étiquetés (voisins) de la base d'apprentissage. La classe associée à x est la classe qui apparaît le plus souvent (majoritaire).

Dans l'exemple ci-dessus, nous avons 3 classes (ω1, ω2, ω3) et le but est de trouver la valeur de la classe de l'exemple inconnu x. Nous prenons la distance Euclidienne comme métrique de proximité et k=5 voisins.

Pour rappel, nous sommes dans un apprentissage supervisé, il y a deux grandes étapes dans le programme : l'apprentissage et la prédiction. Avant ça, nous avons séparé le jeu de données en deux, un pour l'apprentissage et un pour les prédictions. On a donc 4 tableaux : xtrain, ytrain, xtest et ytest. Les 'x' correspond aux données et les 'y' aux classes.

Notre programme est construit de cette façon :

```
clf = neighbors.KNeighborsClassifier(10)#Definition de la range pour la méthode k-nn
clf.fit(xtrain,ytrain)#phase d'apprentissage xtrain=data ytrain=target
print(i, ':', clf.score(xtest,ytest))#phase de prédiction
```

On retrouve les deux grandes étapes évoquées précédemment. La fonction ***clf.fit(xtrain, ytrain)*** correspond à la phase d'apprentissage. La fonction ***clf.score(xtest, ytest)*** correspond à la phase de prédiction ; cette fonction compare également les classes prédites avec celle contenue dans ytest et elle retourne un pourcentage correspond à la précision de la prédiction.

C. Effet des différents paramètres sur la précision

Analyse de l'effet du nombre de voisin sur la précision de la prédiction:

Les tests ont été faits pour un k compris entre 2 et 15. On remarque que la valeur optimale est k = 4. Cette valeur peut changer suivant le test effectué. Effectivement, plusieurs tests ont été effectués et parfois, le k optimal est de 8. On remarque également qu'un k élevé n'est pas forcément la meilleure solution dans le cas de la méthode des plus proches voisins.
(cf TP1_Machine_learning.ipynb - 2ème bloc)

Analyse de l'effet du nombre d'échantillons utilisé pour la phase d'apprentissage:

Les tests ont été effectués avec un pourcentage compris entre 10% et 90%. Ce pourcentage correspond à la partie utilisée pour l'apprentissage. La valeur optimale est 80%.
(cf TP1_Machine_learning.ipynb - 3ème bloc)

Analyse de l'effet de la distance entre deux points:

Les tests ont été effectués avec une distance entre deux points comprise entre 1 et 10. La valeur optimale trouvée est p = 7. Encore une fois, un p élevé n'est pas forcément la meilleure solution dans ce cas.
(cf TP1_Machine_learning.ipynb - 4ème bloc)

Analyse de l'effet du paramètre n_jobs:

Les paramètres précédents retenus sont les suivants: k = 4, p = 7, 80% des données sont utilisées pour le training. Il reste donc 20% soit 1000 données à prédire.

```
score n_job=1 : 0.933
time n_job=1 : 34.351948976516724
score n_job=-1 : 0.927
time n_job=-1 : 15.464780807495117
```

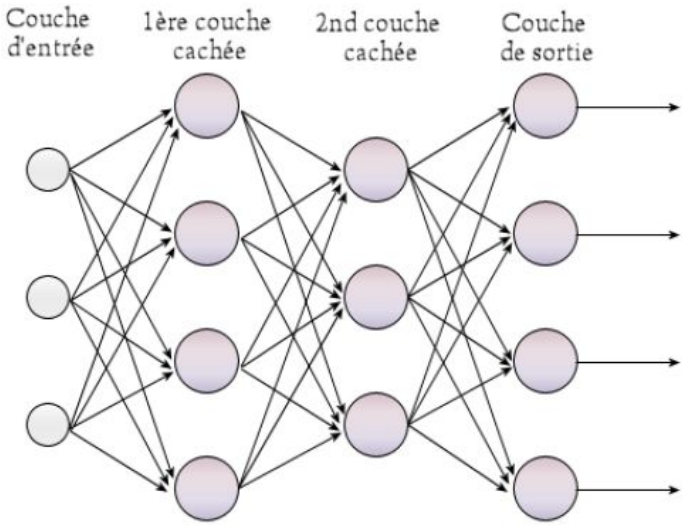
Le n_job influe beaucoup sur le temps d'exécution. On constate que le temps double entre un n_job = -1 et un n_job = 1.
(cf TP1_Machine_learning.ipynb - 5ème bloc)

D. Conclusion sur la méthode des plus proches voisins

La méthode k-nn a l'avantage d'avoir un apprentissage rapide et est également optimale dans le cas des prédictions sur chiffres manuscrits. Cependant, cette méthode nécessite beaucoup de temps pour prédire. On atteint un temps de traitement de 15 secondes minimum pour prédire 1000 données. Cette méthode nécessite également beaucoup de mémoire pour pouvoir être fiable.

II. TP2 - Apprentissage supervisé par perceptron multi-couche

Cette apprentissage est différent de celui utilisé dans le TP précédent. Ici, cet apprentissage supervisé possède des couches de neurones cachées ajoutées entre les entrées et les sorties de l'algorithme. Cela ressemble au schéma suivant :



Architecture d'un apprentissage avec 2 couches cachées

Au cours de ce TP, nous allons utiliser la classe *MLPClassifier*, qui va nous permettre de modifier facilement différents paramètres de notre apprentissage. **Le but du TP est de trouver la configuration permettant d'obtenir les meilleurs résultats possibles**, en modifiant notamment les paramètres suivants :

- *hidden_layer_sizes* : nombre de neurones par couche cachée
- *activation* : permet de définir la fonction d'activation pour les couches cachées
- *solver* : permet de choisir l'algorithme utilisé
- *alpha* : magnitude de la régularisation L2.

Comme lors du TP précédent, nous avons utilisé le jeu de données MNIST contenant 70 000 images (de 784 pixels) de chiffres manuscrits. Au lieu d'utiliser les 70 000 images, nous en avons utilisés 49 000 (70%) pour des soucis de rapidité d'exécution **(voir partie [2])**.

Nous avons ensuite construit un premier modèle de classification en mettant le paramètre *hidden_layer_sizes = (50)*. Cela a pour effet d'utiliser une seule couche cachée de 50 neurones **(voir partie [3])**. Nous avons obtenu un score d'environ : *0.948619*.

Pour tester l'apprentissage, nous avons regardé la valeur prédite pour l'image 4 et la valeur de l'étiquette. On trouve bien la même valeur **(voir partie [4])**.

Le but, comme énoncé précédemment va maintenant être de faire varier le nombre de couches, le nombre de neurones par couches, le type d'activation, le type de solver et l'alpha. Pour comparer les différentes solutions, nous allons nous appuyer sur les mesures suivantes :

- La précision du modèle.
- Le temps d'apprentissage.

a. Le nombre de couches

Dans cette partie, nous avons observé l'impact du nombre de couches sur les mesures énoncées précédemment. Nous avons gardé **50 neurones par couche**. Nous avons testé pour 2, 10, 20, 30, 40 et 50 couches **(voir partie [5])**. On relève les résultats suivants :

Nombre de couches	*Temps d'apprentissage (s)*	*Précision*
2	*77.160786*	*0.953952*
10	***77.905202***	***0.967048***
20	*69.766635*	*0.966810*
30	*52.145304*	*0.955810*
40	*62.716936*	*0.778*
50	*41.946659*	*0.697714*

Au vu des résultats compilés dans le tableau précédent, nous avons choisi de continuer le TP avec **10 couches**, car c'est le nombre de couches qui possède la plus grande précision, et que le temps d'apprentissage est moins important que la précision.

b. Le type d'algorithme

Dans cette partie, nous avons donc pris un réseau avec **10 couches de 50 neurones**, puis nous avons changé le type d'algorithme afin de voir l'influence de celui-ci. Nous avons testé 3 algorithmes : Adam, L-BFGS et SGD. Adam est l'algorithme par défaut et a donc déjà été testé dans la partie précédente **(voir partie [5])**.

L'algorithme **SGD** a été testé dans la **partie [6]**, et le **L-BFGS** dans la **partie [7]**. Nous avons pu mesurer les résultats suivants :

Type d'algorithme	*Temps d'apprentissage*	*Précision*
Adam	***77.905202***	***0.967048***
SGD	*57.534073*	*0.960143*

L-BFGS	*56.829690*	*0.949524*

Comme pour la partie précédente, nous nous sommes surtout appuyés sur la valeur de précision. La meilleure précision étant pour l'algorithme **Adam**, c'est celui que nous allons utiliser par la suite.

c. La fonction d'activation

Nous avons ensuite testé de modifier la fonction d'activation pour encore une fois trouver la plus précise possible. Dans les **parties [8], [9], [10] et [11]**, nous avons respectivement testé les fonctions *identity, logistic, tanh* et *relu*.

Type de fonction	*Temps d'apprentissage*	*Précision*
identity	*89.348637*	*0.905857*
logistic	*31.663937*	*0.112810*
tanh	*36.674516*	*0.930048*
relu	***53.431714***	***0.962381***

On remarque que la meilleure précision est celle de la fonction **relu**. C'est donc celle que l'on a choisi pour la suite.

d. La régularisation

Nous avons enfin agi sur la valeur de la régularisation □. Ces tests ont été effectués dans la **partie [12]**. Nous avons compilé les précisions mesurées dans le tableau suivant :

Régularisation α	*Précision*
0.0001	*0.971905*
0.001	*0.972190*
0.01	***0.972286***
0.1	*0.972238*

Comme il est possible de l'observer dans le tableau précédent, la meilleure valeur pour la régularisation est **0.01**.

e. Meilleure modèle possible

Au vu des résultats précédents, on peut déterminer que le meilleur modèle possible dans notre cas d'étude est un modèle avec **10 couches de 50 neurones**, utilisant l'algorithme **Adam** (qui est une extension de l'algorithme du gradient stochastique) et la fonction **relu**, et avec une régularisation égale à **0.01**.

III. Comparaison entre les deux méthodes

Grâce aux deux TP précédents, nous avons pu observer deux méthodes différentes : la méthodes des plus proches voisins et la méthode des réseaux de neurones.

Les différentes observations et mesures que nous avons pu faire nous ont permis de comparer ces deux techniques et de dégager certains avantages et inconvénients sur plusieurs points.

a. Le temps d'apprentissage

Le premier point sur lequel les techniques peuvent être comparées est le temps d'apprentissage.

Dans l'ensemble, et si on prend en compte les meilleures configurations de réseaux de neurones, le temps d'apprentissage de k-nn est beaucoup plus court que celui des réseaux de neurones.

b. La précision

Le second point de comparaison est celui de la précision.

Cette fois, on peut dire de manière certaine que les réseaux de neurones proposent une précision bien supérieure à celle des k-nn. Mais pour cela, il faut que le réseaux de neurone soit très bien configurés, ce qui nous amène à notre troisième point.

c. La complexité

C'est le dernier point sur lequel les techniques peuvent être comparées.

En effet, la méthode des réseaux de neurones est beaucoup plus complexe, que ce soit pour son fonctionnement ou pour sa configuration. Nous avons pu remarquer lors du second TP qu'il y avait beaucoup de paramètres à gérer afin d'arriver au meilleur résultat, ce qui n'est pas le cas de la méthode des k-nn.

d. Pour conclure

Pour conclure, on pourrait être tenté d'utiliser les réseaux de neurones car ils offrent une meilleure précision. Mais dans le cas de notre problème, nous avons conclu qu'il serait sûrement mieux d'utiliser la méthode de k-nn.

En effet, cette méthode est moins complexe et est suffisante pour notre reconnaissance de chiffres manuscrits.