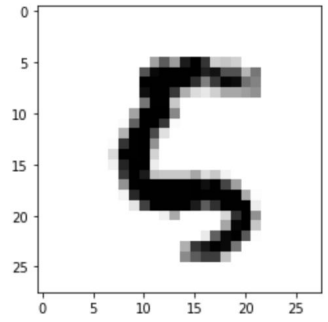
Rapport TP - Apprentissage supervisé KNN

Bravais Jérôme & Shan Xiaoshan - 5 SDBD B1
https://github.com/KrustyLeBot/Apprentissage

1. Jeu de données

Le jeu de données MNIST est composé de 70.000 images de chiffre manuscrit (les data), ainsi que les 70.000 nombre associés (les targets).



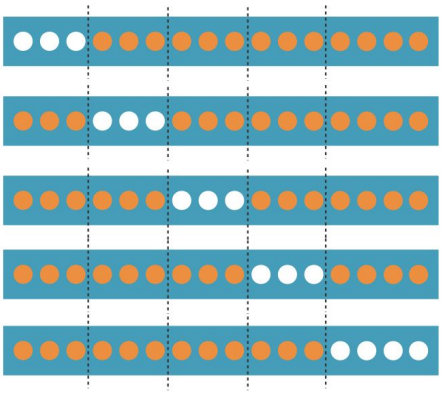

```
Target associé:  5
```

2. Méthode des k-plus proches voisins

Cette méthode consiste à rechercher les plus proches exemples labellisés pour identifier le label de l'instance analysée. Nous pouvons faire varier 2 paramètres:

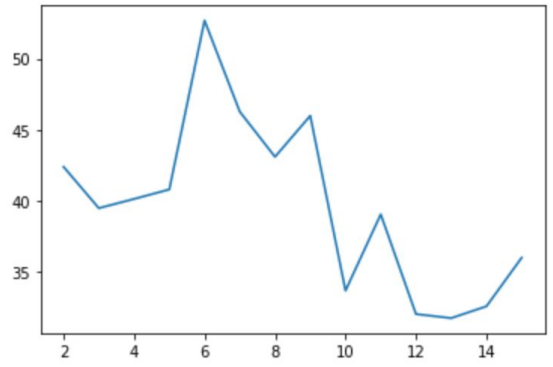
- k : le nombre de voisins plus proche
- n_jobs : le nombre de tâche effectués en parallèle

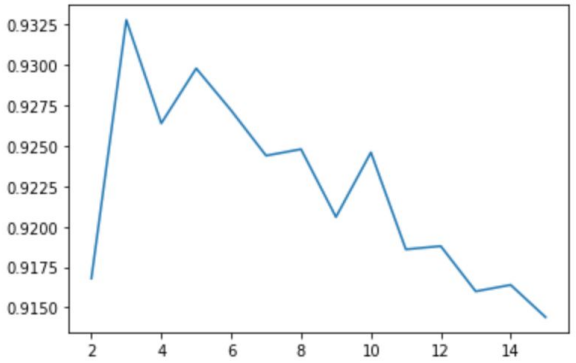
Pour les prochains tests, afin de réduire le temps nécessaire à l'apprentissage, nous avons réduit le dataset à un échantillon de 5000 données. Afin de vérifier la performance de l'algorithme pour les changement de paramètres, nous allons utiliser la méthode des K-Folds avec 10 folds, qui consiste à découper le jeu de données en k parties, puis tour à tour chacune des k partie est utilisée en tant que jeu de test, et le reste est utilisé pour l'entraînement. Les score de chaque fold sont ensuite moyennés afin de renvoyer un score de fiabilité global.



a. Variation de k

Nous allons faire varier le nombre de plus proches voisins, puis analyser le score de fiabilité ainsi que les temps d'apprentissage nécessaires (avec 10 folds, ces temps représentent en réalité 10 apprentissage et 10 calculs de score, mais sont tout de même utiles pour une comparaison entre les k). Nous avons réalisé ces test pour k variant de 2 à 15. Voici les résultats obtenus:

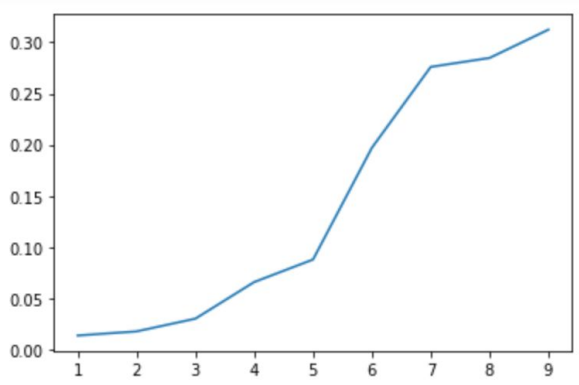


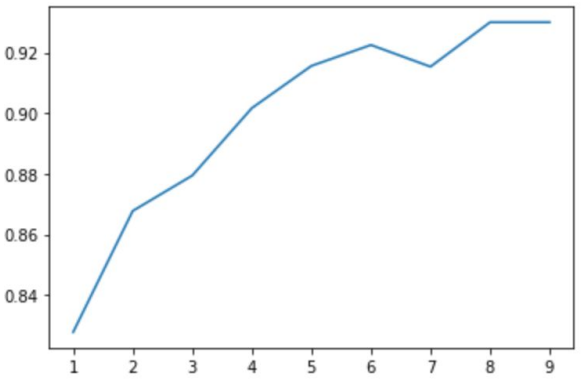


Pour temps (figure de gauche), on ne constate pas de lien significatif selon les valeurs de k. En revanche, pour le score (a droite), on voit clairement que l'algorithme est le plus efficace pour k=3. C'est donc le paramètre qui semble le plus optimal pour ce dataset.

b. Variation de la taille train/test

Nous allons ici regarder l'évolution du temps d'apprentissage ainsi que le score d'efficacité pour un k donné, selon la taille du jeu d'entraînement. Nous utiliserons k=3.





Sur la figure de gauche, nous pouvons observer comme attendu, que plus le set d'entraînement est grand, plus le temps de calcul augmente. Quand au

score de fiabilité, de la même manière, augmente de manière logique, car l'algorithme possède plus de données pour pouvoir s'entraîner.

c. Variation de n_jobs

Pour ce test, nous cherchons à voir si la parallélisation des jobs permet de réduire le temps d'entraînement de l'algorithme. Knn est un cas particulier, car ce paramètre n'a aucun influence lors de l'entraînement, mais uniquement lors du calcul du score avec le dataset de test. Nous allons donc comparer n_jobs = 1 et n_jobs = -1, le nombre maximum de jobs en parallèle, avec une boucle de 10 calcul de score.

- n_jobs : 1 Temps : 64.42629504203796
- n_jobs : -1 Temps : 24.768293857574463

On peut observer en effet une différence entre les 2, avec n_jobs = -1 étant le plus rapide de presque 3 fois. Il est en effet logique que plus on parallélise une tâche, plus elle sera finie vite.

3. Points forts et points faibles de KNN

a. Points forts

- training phase très rapide
- il est inutile d'entraîner un model pour faire de la généralisation
- utilisable pour des données non linéaires

b. Points faibles

- testing phase lente et requiert beaucoup de mémoire pour mettre le dataset en entier dedans
- pas fait pour des datasets très larges
- très sensible au grandeur des distances dans les données