Rapport TP Apprentissage Supervisé

2019/2020
Pierre BINET & Constance GAY
5 SDBD B1
Github : https://github.com/ConstanceGay/Apprentissage.git

TP1 : KNN

1. Jeux de Données

print(mnist) → Cette commande affiche toutes les données dans mnist donc : les images, leurs interprétation en chiffre...

print (mnist.data) → Cette commande affiche les pixels de chaque image (1 image par ligne).

print (mnist.target) → Cette commande affiche les chiffres identifiés sur les images.

len(mnist.data) → Cette commande renvoi le nombre d'images dans le dataset. Il y a donc 70 000 images.

help(len) → aide de la commande len

print (mnist.data.shape) → Cette commande donne d'abord le nombre d'objet puis ensuite le nombre de composantes (pixels) par objet, ici 28*28=784

print (mnist.target.shape) → Comme ici on a uniquement les chiffres identifiés, il n'y a pas de sous-composante.

Mnist.data[0] → Cette commande affiche la matrice de pixel du premier objet (image) du set.

Mnist.data[0][1] → Cette commande renvoi la valeur du premier pixel du premier objet du dataset.

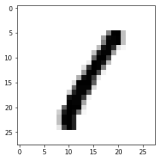
Mnist.data[:,1] → Cette commande renvoi le premier pixel de tous les objets.

Mnist.data[:100] → Chaque ligne est un objet entier, et il y a 100 lignes. Cette commande renvoi donc les 100 premiers objets(images).

2. La méthode des k-plus proches voisins

Prendre un échantillon de données appelé data avec une taille de 5000 exemples, diviser la base de données à 80% pour l'apprentissage (training) et à 20% pour les tests, entrainez un classifieur k-nn avec k = 10 sur le jeu de données chargé.

1) *Afficher la classe de l'image 4 et sa classe prédite.*



La classe prédite par le classifieur est 1, ce qui correspond bien à l'image.

2) *Affiche le score sur l'échantillon de test*

La précision calculée sur l'échantillon de test est : 0.926, ce qui est relativement précis.

3) *Quel est le taux d'erreur sur vos données d'apprentissage ? Est-ce normal ?*

La précision sur l'échantillon d'apprentissage est de : 0.934, ce qui est plus haut que pour l'échantillon de test. Ce n'est cependant pas 100% car sinon on aurait des problèmes d'overfitting (L'échantillon d'apprentissage pouvant contenir du bruit).

4) *Faites varier le nombre de voisins (k) de 2 jusqu'à 15 et afficher le score. Quel est le k optimal ?*

On sait que trop peu de voisins peut donner un sous-apprentissage (underfitting) et que trop de voisins peut donner un sur-apprentissage (overfitting). Ici on constate que la précision est la plus grande entre 3 et 5 voisins.

5) *Utilisez la function KFold(len(X),n_folds=10,shuffle=True)*

La fonction KFold permet de diviser les échantillons en un certain nombre de tranches (ici on a prit 10) et avec la fonction kf.split on peut faire tourner le classifieur en décalant à chaque boucle la tranche de test par rapport à celle de train.
On constate que la précision oscille légèrement selon la tranche utilisée pour l'entraînement.

6) *Faites varier le pourcentage des échantillons (training et test) et afficher le score. Quel est le pourcentage remarquable ?*

On a fait varier le pourcentage de données d'apprentissage de 10% à 90%. On constate que plus le pourcentage de données d'apprentissages est haut, plus le score est élevé. C'est parce que le modèle a pu apprendre avec un plus grand nombre d'exemples.

7) *Faites varier la taille de l'échantillon training et afficher la précision. Qu'est-ce que vous remarquez?*

On a donné les tailles suivantes d'échantillon : 100,1000, 5000 et 10000. On constate que plus le nombre d'échantillons training est élevé, plus la précision est élevée. Cependant le temps d'exécution de la fonction augmente rapidement.

8) *Faites varier les types de distances (p). Quelle est la meilleure distance ?*

On a fait varier les types de distances (p) de la fonction KNeighborsClassifier qui peut prendre la valeur 1 ou 2. La variable p représente le type de distance entre les variables. Quand **p=1** on a une **distance de Manhattan** et quand **p=2** on a une **distance euclidienne**. Normalement la distance la plus appropriée est choisie en fonction du **type de donnée** du dataset. La distance euclidienne est utilisée pour des données du **quantitatives du même type** alors que celle de Manhattan est plutôt pour les données d'entrées qui ne sont **pas du même type**.
D'après les résultats, la distance la plus appropriée ici est la distance euclidienne et c'est logique car nous avons toujours des données d'entrées quantitatives du même type (tableau de pixel 28x28).

9) *Fixer n_job à 1 puis à -1 et calculer le temps dans chacun.*

On fait varier n_jobs de la fonction KNeighborsClassifier qui peut prendre la valeur 1 ou -1. N_jobs représente le nombre de tâche s'exécutant en parallèle en cherchant des voisins. Quand il est à 1 ça signifie qu'il n'y a qu'une tâche alors que -1 signifie que tout le processeur est utilisé. On constate ici qu'il est plus rapide de ne pas utiliser de parallélisation.

10) *A votre avis, quels sont les avantages et les inconvénients des k-nn : optimalité? Temps de calcul ? Passage à l'échelle ?*

En conclusion, nous avons pu voir que KNN est une méthode d'apprentissage rapide pour atteindre une précision acceptable (0.9 à peu près) mais dès qu'on tente d'améliorer encore plus cette précision en agrandissant l'échantillon d'entraînement le système se ralentit. Cela est dû au fait qu'il faut stocker de plus en plus de données et calculer plus de distances.
On voit aussi que plusieurs variables tel que la méthode de calcul de distance sont difficiles à prévoir à l'avance et qu'il faut donc faire des tests de l'algorithme sur différentes méthodes.
Malgré cela on admet que la précision atteinte avec peu d'exemples est haute et que la méthode KNN est une méthode simple à appréhender.

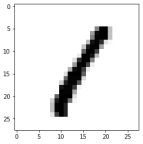
TP2: ANN

Diviser la base de données en 49000 lignes pour l'apprentissage (training) et le reste pour les tests. Construire un modèle de classification ayant comme paramètre : hidden_layer_sizes = (50).

1) *Calculer la précision du classifieur*

La précision calculée est de 0.9451428571428572. C'est plutôt précis.

2) *Afficher la classe de l'image 4 et sa classe prédite.*

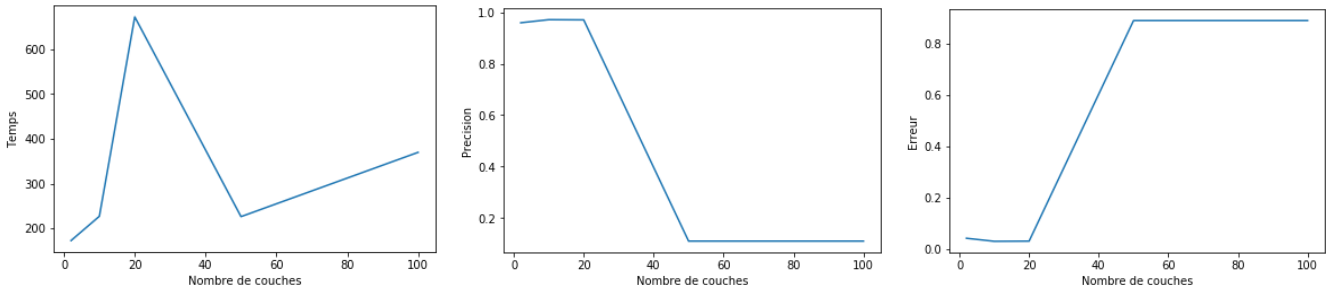


La classe prédite par le classifieur est 1, ce qui correspond bien à l'image.

3) *Calculez la précession en utilisant le package : metrics.precision_score(ytest_pr, ypredTest_pr,average='micro').*

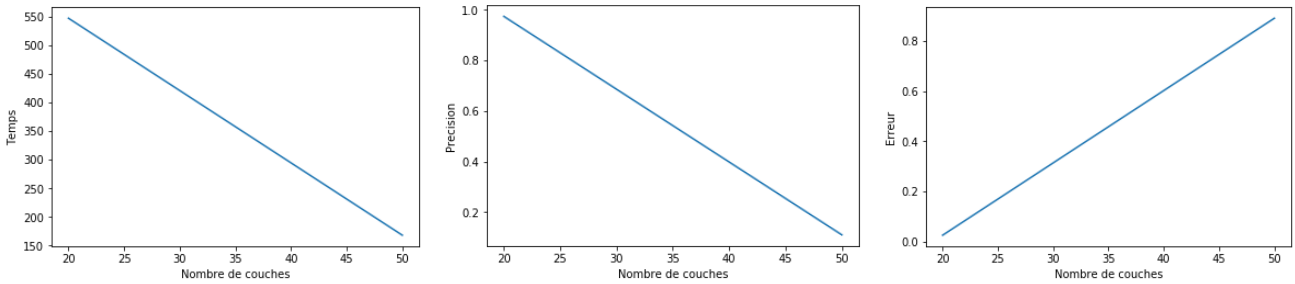
La précision calculée est de 0.9451428571428572 donc on trouve la même précision.

4) *Varier le nombre de couches 2,10,20,50 et 100 avec chacun 50 neurones, et recalculer la précision du classifieur :*



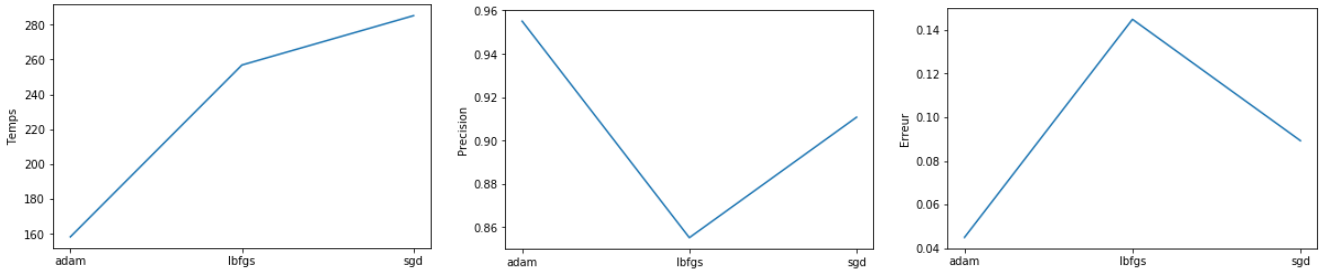
On voit que si il y a trop de couches la précision baisse énormément.

5) *Construire deux modèles de classification des données mnist, avec des réseaux qui ont respectivement 50 et 20 couches cachées, et des tailles de couches commençant à 60 et décrémentant respectivement de 1 et 2. Quelles sont les performances en taux de bonne classification et en temps d'apprentissage obtenus pour chaque modèle ?*



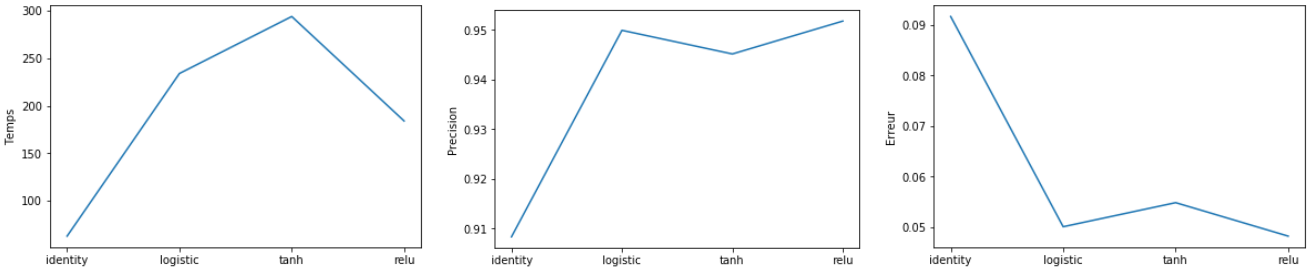
On constate que 50 couches est beaucoup plus rapide que 20 (1 minutes contre 6). Cependant la précision de 50 couches est très faible alors que celle de 20 couches est acceptable.

6) *Étudier la convergence des algorithmes d'optimisation disponibles : L-BFGS, SGD et Adam.*



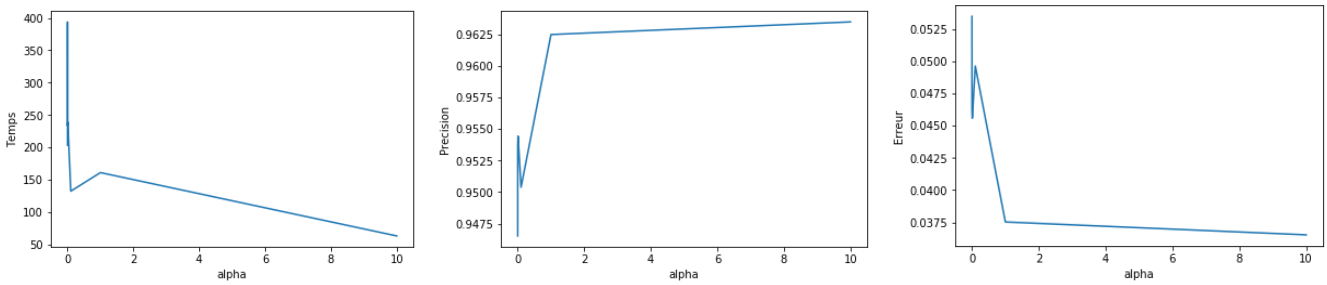
On constate que pour le set de data que nous avons, l'algorithme le plus précis est Adam, ce qui paraît logique car il est approprié pour les datasets larges alors que lbfgs est pour les plus petits.

7) *Varier les fonctions d'activation {'identity', 'logistic', 'tanh', 'relu'}*



La fonction d'activation correspond à la relation entre chaque couche cachée. Ici on constate que la fonction d'activation relu est la plus précise : f(x) = max(0, x).

8) *Changer la valeur de la régularisation L2 (paramètre α)*



L'overfitting est un cas où le modèle correspond trop à l'ensemble de données et qui ne pourrait donc pas correspondre à des données supplémentaires, l'erreur d'apprentissage sera donc nulle.
Le paramètre de régularisation alpha permet de minimiser l'overfitting. Si sa valeur est trop haut il y aura overfitting mais si elle est trop basse il y aura underfitting.
Ici on constate que la valeur alpha=1.0 est la plus précise.

9) Choisissez le modèle qui propose de meilleurs résultats

De ce que nous avons vu pour le moment, le meilleur modèle aurait :

- 20 couches avec le nombre de neurones variant de 60 à 11 (précision et temps de calcul optimaux).
- L'algorithme d'optimisation adam.
- La fonction d'activation 'relu'.
- La valeur alpha = 1.0

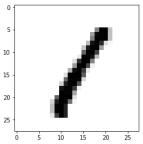
10) A votre avis, quels sont les avantages et les inconvénients des A-nn : optimalité ? temps de calcul ? passage à l'échelle ?

Durant le TP sur la méthode K-NN, le temps de calcul augmentait drastiquement dès que l'on faisait travailler l'algorithme d'apprentissage sur un échantillon de plus de 5000 données. Avec les A-NN, on est capable de réaliser un apprentissage bien plus précis sur 49000 données et en l'espace de seulement quelques minutes. En revanche lorsque les A-NN sont mal paramétrés, les performances sont rapidement dégradées, on perds en précision et on atteints parfois des temps de calculs de l'ordre de la dizaine de minute. Le paramétrage du réseau de neurone est donc essentiel et passe ici aussi par la compréhension des données et leur taille (afin notamment de trouver un paramètre α de régularisation, un algorithme d'optimisation et la fonction d'activation adéquats).

TP3 : SVM

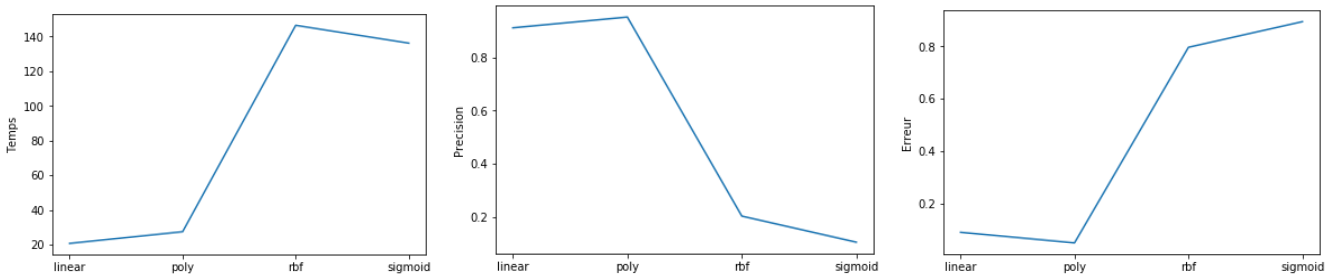
Comme l'utilisation de la base de données en entier prenait trop de temps nos exemples ne se basent que sur l'utilisation de seulement 10000 images.

1) *Construire un modèle de classification ayant comme paramètres un noyau linear et afficher la classe prédite pour l'image 4.*



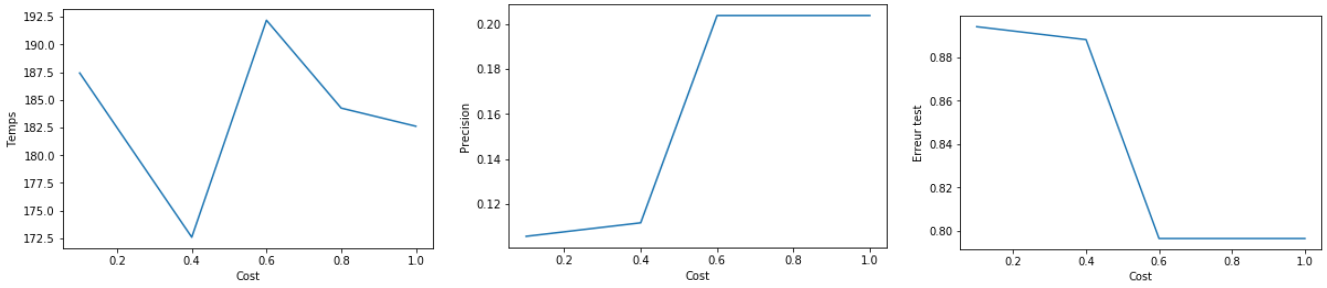
La classe prédite par le classifieur est 1, ce qui correspond bien à l'image.

2) *Tentez d'améliorer les résultats en variant la fonction noyau : 'poly', 'rbf', 'sigmoid', 'precomputed' .*



Quand on utilise SVM il est possible de personnaliser le noyau en lui donnant une fonction ou une matrice de Gram 'precomputed'. Nous n'avons pas traité cet exemple.
Le but de la fonction noyau est de se projeter dans un espace de redescription de très grande dimension. Ici on constate que la fonction noyau la plus précise est la fonction polynomiale même si elle est légèrement plus lente que la fonction linear.

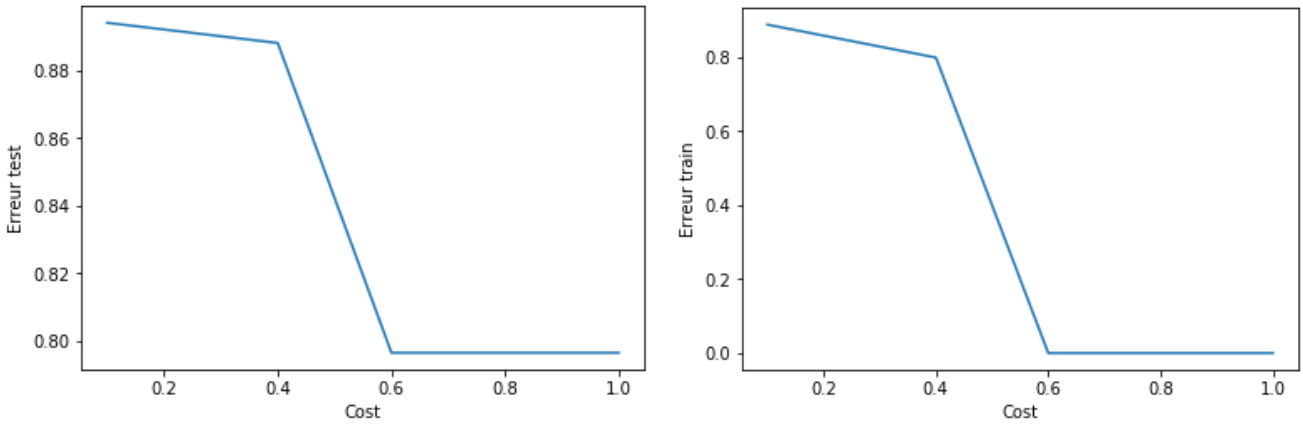
3) *Faites varier le paramètre de tolérance aux erreurs C (5 valeurs entre 0.1 et 1).*



Le paramètre C est un terme de régularisation qui permet de prévenir la mauvaise classification. Pour les valeurs élevées de C, l'espace entre les 'espace de redescription' sera plus petit et donc il y aura moins d'erreur de classification. Inversement une valeur de C basse

donnera des espaces séparés par de plus grosses marges et il y aura plus d'erreur de classification.
On constate cela dans les graphes que nous avons créés. La valeur de C la plus adéquate en temps et en précision serait donc 0.6 donc ce cas là.

4) *Tracez la courbe d'erreur de classification sur les données d'entraînement et de test en fonction de C.*



On constate que pour un C tout petit il y a le même taux d'erreur pour les données d'entraînement et de test. Mais à mesure que C augmente le taux d'erreur devient minimum pour les valeurs d'entraînement.

5) *Construisez la matrice de confusion.*

Nous avons construit la matrice de confusion suivante en prenant la fonction noyau polynomiale et la valeur de C = 0.6 car d'après les questions précédentes ce sont les valeurs les plus appropriées.

```
[[180   0   0   0   0   2   1   0   1   0]
 [  0 211   0   3   0   0   0   0   0   0]
 [  3   3 186   1   1   1   0   3   1   0]
 [  1   1   2 182   0   3   0   0   6   3]
 [  1   1   0   0 191   1   0   0   0   2]
 [  0   0   0   0   0 189   0   0   2   0]
 [  0   2   0   0   0   0 192   0   1   0]
 [  0   5   2   0   3   0   0 207   0   5]
 [  0   2   2   1   0   1   0   1 198   4]
 [  1   4   0   1   2   1   0   3   2 178]]
```

La matrice de confusion permet de juger la qualité d'un système de classification. Chaque ligne L correspond à une classe réelle, chaque colonne C correspond à une classe estimée. Idéalement on souhaite que le système classe correctement chaque membre de chaque classe, et c'est pourquoi on souhaite voir tous les membres sur la diagonales.
Ceux qui n'y sont pas sont ceux qui ont été mal diagnostiqués (ie. dont la classe estimée C ≠ la classe réelle L). Sur notre matrice on constate que la plupart des valeurs sont sur la diagonale avec peu de mauvaises classifications.

6) A votre avis, quels sont les avantages et les inconvénients du SVM ?

Avantages :

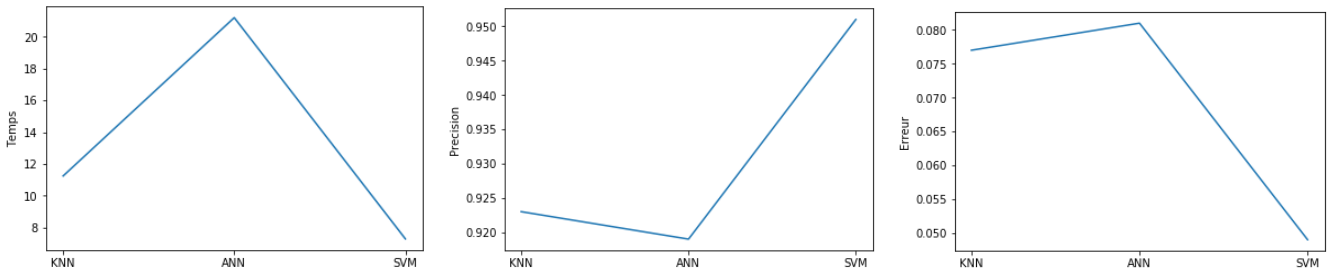
- Capacité à traiter des grandes dimension de données
- Permet de traiter des problèmes non linéaires avec le choix des fonction noyaux
- Le paramètre C permet de combattre l'overfitting

Inconvénients:

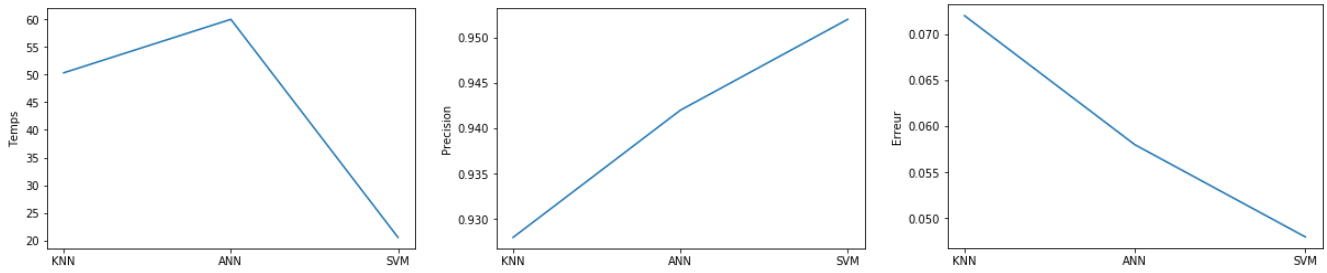
- Les paramètres sont sensibles et il faut les tester afin de choisir les plus appropriés pour la base de données à traiter
- Il est difficile de traiter un trop grand nombre de données. Par exemple quand nous avons voulu utiliser la bibliothèque MNIST entière le système n'avait toujours pas terminé ses calculs au bout d'une demi-heure.

Comparaison KNN - ANN - SVM

Pour 5000 données (4000 d'entraînement et 1000 de test):



Pour 10000 données (8000 d'entraînement et 2000 de test):



Matrices de confusion:

```
[[171   0   0   0   0   1   0   0   0   0]
 [  0 221   0   0   0   0   0   1   0   0]
 [  5   9 154   1   1   0   0   3   1   0]
 [  2   2   3 196   0   7   1   4   0   1]
 [  1   4   0   0 202   0   3   0   0  15]
 [  2   1   0  11   0 185   4   2   2   2]
 [  3   0   0   0   0   1 173   0   0   0]
 [  2   8   2   0   3   0   0 195   0   2]
 [  4  10   1   4   0   3   2   0 179   1]
 [  0   1   0   2   1   1   0   4   0 180]]
```

KNN

```
[[170   0   0   1   0   0   1   0   0   0]
 [  0 216   2   1   1   1   0   0   0   1]
 [  1   2 161   3   3   0   1   1   2   0]
 [  0   3   0 190   0  15   1   3   4   0]
 [  3   0   0   0 214   1   2   1   0   4]
 [  3   1   1   3   1 194   3   1   2   0]
 [  2   0   0   0   2   2 170   0   1   0]
 [  2   5   2   1   2   0   0 199   0   1]
 [  4   0   1   0   1   5   2   0 190   1]
 [  0   0   0   1   4   0   0   3   1 180]]
```

ANN

```
[[171   0   0   1   0   0   0   0   0   0]
 [  0 219   0   1   0   0   0   0   1   1]
 [  3   2 164   1   0   0   1   2   1   0]
 [  1   2   2 201   0   6   0   2   1   1]
 [  2   0   1   0 214   0   2   1   0   5]
 [  2   1   2   8   0 190   3   0   1   2]
 [  2   0   0   0   1   1 173   0   0   0]
 [  1   6   2   0   2   2   0 198   0   1]
 [  2   2   2   1   0   2   1   0 193   1]
 [  0   0   0   0   4   0   0   3   1 181]]
```

SVM

La méthode SVM avec les paramètres C=0.6 et la fonction de noyaux polynomiale est clairement plus performante que ça soit en temps ou en précision.
La fonction ANN est plus précise que KNN uniquement quand il y a un grand nombre de données mais elle est toujours plus lente que les deux autres.

En conclusion on s'aperçoit ici et avec l'expérience des TP précédents que la méthode SVM s'avère optimale dans un très grand nombre de cas, bien qu'elle soit largement plus longue à paramétrer. La méthode KNN quant à elle nécessite peu de paramétrage et est très efficace

sur un petit jeu de donnée. Enfin la ANN représente un bon compromis car elle est très précise pour les traitements des grands jeux de données et malgrés sa lenteur relative à la méthode SVM elle nécessite moins de paramétrage.