GONZALES Adrien 5SDBD A2
MEGA Adrian

TP Apprentissage Supervisé

Introduction

Ce rapport a pour but de rendre compte de nos travaux en TP d'apprentissage supervisé. Chaque chapitre sera consacré à l'un des classifieurs étudiés à chaque TP (KNN, MLP, SVC) puis une synthèse comparera leur efficacité à chacun. Les scripts des programmes peuvent être retrouvés sur le dépôt suivant :

https://github.com/dx07/5SDBD-Apprentissage

Le jeu de données sur lequel nous travaillons est une base de données de chiffres écrits à la main. Nous avons également la vraie valeur représentée par ces chiffres incluse dans le dataset pour identifier quel chiffre correspond au dessin.

Selon les questions des sujets de TP, les tailles des données sélectionnées seront amenées à être changées pour des raisons de démonstration.

I – Méthode KNN, les k plus proches voisins

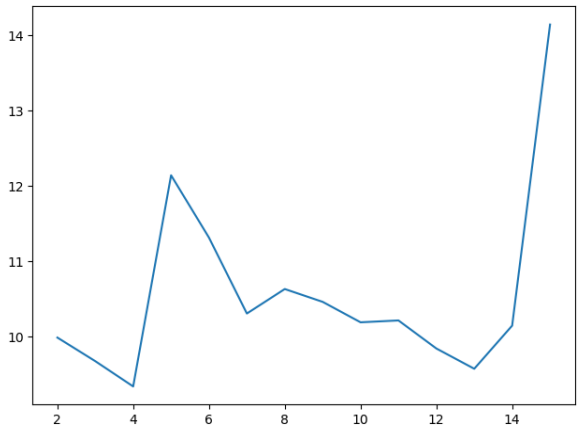
Cette méthode consiste à rechercher les k plus proches exemples labellisés pour identifier le label de l'instance analysée. Nous pouvons faire varier les paramètres suivants :

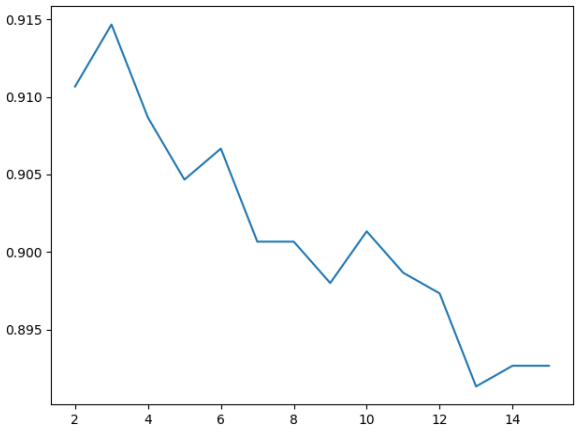
- K : Le nombre de voisins plus proches
- n_jobs : Le nombre de jobs parallèles (1 par défaut, -1 pour tous)

Pour ces premiers tests, nous avons pris un extrait du dataset de taille 5000. L'algorithme étant très long à s'exécuter sur nos machines, nous sommes contraints d'utiliser une petite partie du dataset entier. Le partage du dataset entre données d'entraînement et données de test est respectivement de 70% et 30%.

a) Variation de K

En faisant varier le nombre de plus proches voisins retenus, nous allons analyser les différents scores de fiabilité obtenus ainsi que les temps d'apprentissage nécessaires. Nous avons fait ces tests pour un K valant de 2 à 15. Voici les résultats obtenus.





Concernant le temps (figure de gauche), on ne constate pas de différence significative selon la valeur de K. En revanche, pour le score (figure de droite), nous pouvons observer un pic d'efficacité à K = 3. Il serait le paramètre le plus optimal pour ce jeu de données.

b) Variation de n_jobs

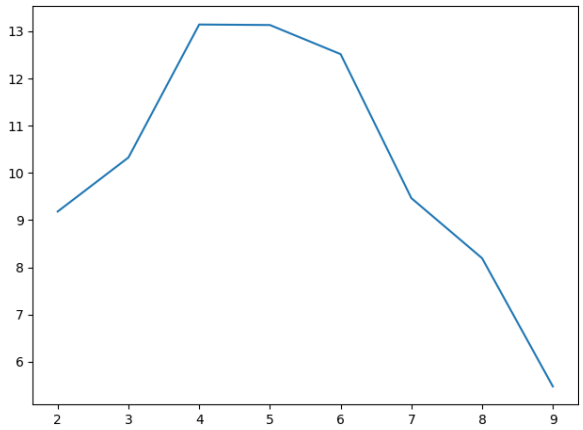
Cette étape va nous montrer si on gagne du temps en changeant le nombre de jobs parallèles utilisés. Pour cela nous allons montrer la différence entre n_jobs = 1 et n_jobs = -1, soit le maximum de jobs en parallèle. Nous gardons K = 3 pour cette étape ainsi que l'échantillon de données sélectionné précédemment. Nous obtenons les temps suivants :

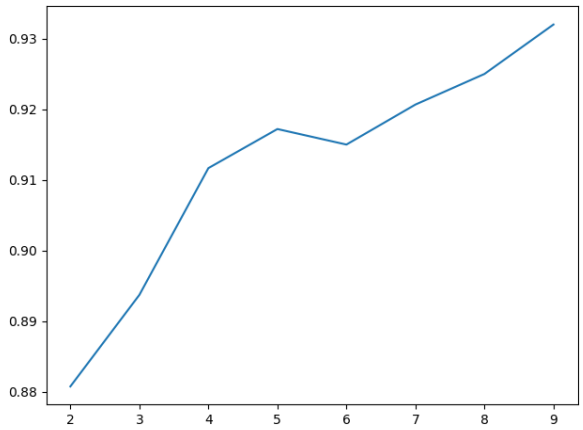
- n_jobs : -1 Temps : 4.365392208099365
- n_jobs : 1 Temps : 9.597620725631714

Cela met clairement en évidence qu'avec davantage de ressources, la puissance de calcul est grandement augmentée et le temps de calcul en est inéluctablement réduit.

c) Variation de la taille de train / test

Exceptionnellement pour cette partie, nous allons tester la variation de la taille d'entraînement pour apprécier l'impact qu'elle peut avoir sur les temps de calcul et le score d'efficacité. Nous conservons toujours K = 3.





A gauche, nous pouvons voir que le temps de calcul se voit un peu augmenté au début, mais significativement diminué avec une plus grande taille de données d'entraînement (deux fois plus rapide avec 90% qu'avec 40%). Pour le score, celui-ci augmente forcement avec davantage de données pour s'entrainer, ce qui est logique dans le fond.

d) Matrice de confusion

Voici à présent la matrice de confusion affichant sur les 10 chiffres possibles de lire le nombre d'occurrences ayant été correctement lues. On obtient les résultats suivants.

- score: 0.9146666666666666
- time: 10.216910362243652

```
[[161   0   1   0   0   1   1   0   1   0]
 [  0 158   0   0   0   0   0   0   0   0]
 [  4   6 143   2   0   1   1   1   1   0]
 [  1   2   1 129   0   5   0   0   1   1]
 [  0   9   1   0 114   0   3   0   1  12]
 [  1   1   0   8   0 110   2   0   0   1]
 [  2   0   0   0   1   0 159   0   0   0]
 [  0   6   1   1   1   0   0 134   0   1]
 [  0  10   2   5   0   7   0   2 126   2]
 [  1   4   0   3   3   0   0   6   0 138]]
```

Par exemple pour la ligne 6, qui correspond au chiffre 5, il y a eu 110 occurrences correctement identifiées contre 13 erronées. Cela met en évidence la bonne efficacité de KNN.

II – Méthode MLP, les couches de neurones

Cette méthode est basée sur une architecture en couches de neurones. Chaque couche de neurone ou couches cachée, est ajoutée entre les entrées et la couche de sortie. La sortie d'un neurone est la combinaison linéaire des p variables d'entrées et d'un biais, toujours pondérés par des poids.

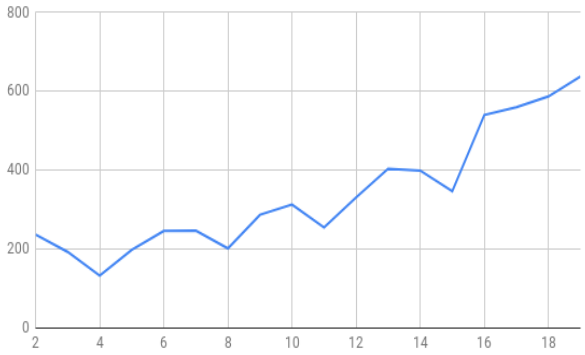
Nous pouvons faire varier les paramètres suivants :

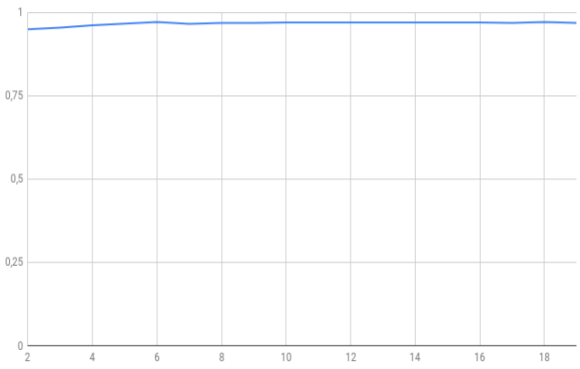
- Le nombre de couches cachées
- Le nombre de neurones par couche cachées

Pour les tests qui vont suivre, nous prenons cette fois un échantillon de 70000 données du dataset, le temps d'exécution nous le permettant (contrairement à KNN). Le partage du dataset entre données d'entraînement et données de test est respectivement de 70% et 30%.

a) Variation du nombre de couches cachées de neurones

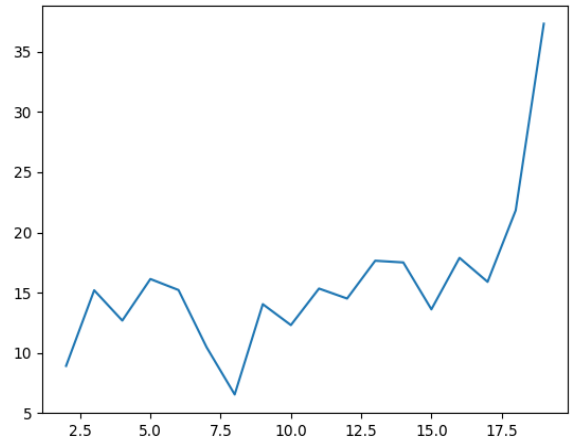
Nous allons pour ce premier test faire varier le nombre couches cachées de neurones et observer les conséquences sur le temps d'exécution ainsi que sur le score de fiabilité. Nous avons fait varier les couches cachées de 2 à 19.

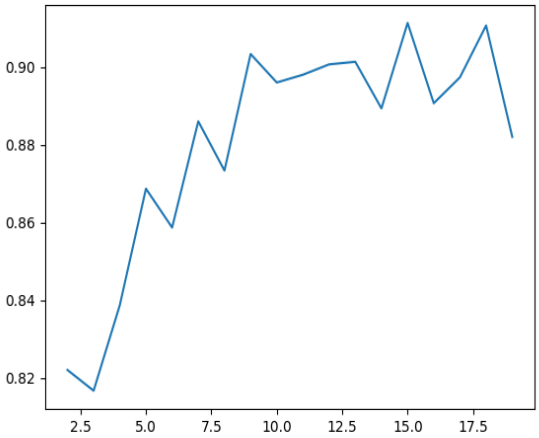




On remarque que plus on augmente le nombre de couches cachées plus le score et l'efficacité augmentent mais aux dépens d'un temps d'exécution plus long. On pourra conclure que sur notre cas précis il ne paraît pas nécessaire d'avoir un grand nombre de couches cachées car le score augmente au final peu de 0,95 à 0,97 mais qui finit par stagner à partir de 10 couches. Le nombre de couches optimal semble être de 11 couches on obtient un temps d'exécution de 255 secondes pour un score final de 0,97.

Pour comparer avec KNN et SVC, nous avons également testé avec un échantillon de 5000.



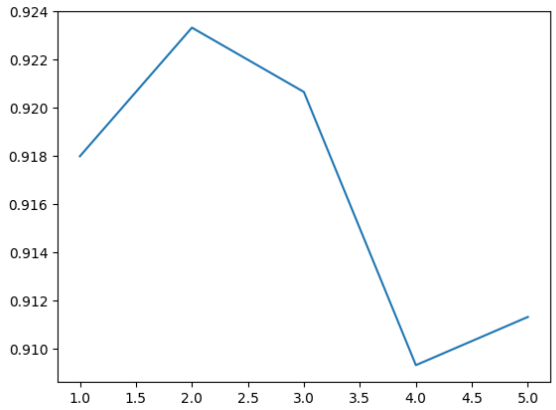


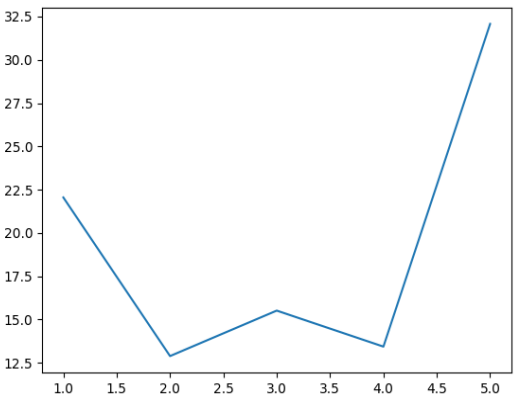
On remarque qu'avec moins de données pour l'entraînement, la méthode du réseau de neurone est moins efficace bien que beaucoup plus rapide. On constate aussi de même qu'avec un échantillon de 70 000 que plus on augmente le nombre de couche plus le temps d'exécution et le score augmentent.

b) Variation du nombre de neurones par couches et du nombre de couches

Pour ce second test nous avons créé cinq réseaux de neurones comprenant entre une et 11 couches avec chaque couche contenant entre 10 et 300 neurones. Le nombre de couches et de neurones par couche est déterminé aléatoirement.

Modèle 1 : [164, 283, 107, 87, 36, 198, 268, 90, 150, 299, 118]
Modèle 2 : [204, 60, 108, 197, 23, 209, 287, 22, 183]
Modèle 3 : [233, 61, 95, 184, 28, 274]
Modèle 4 : [31, 183, 161, 15, 269, 35, 92, 103, 52, 283]
Modèle 5 : [211, 215, 116, 292, 76, 246, 269, 267, 105, 154, 72]

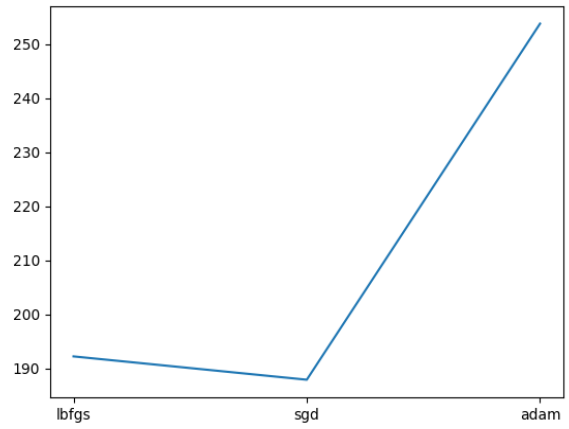


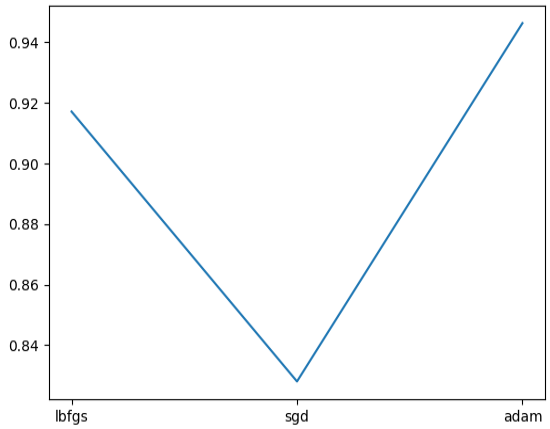


Le modèle le plus efficace est le n°2 que ce soit en terme de temps d'exécution comme de score final. On remarque pour le modèle quatre, qui as un nombre de neurones dans la première couche faible, que son score final est le plus bas. On pourra penser qu'un nombre de neurones trop faible dans la première couche n'est pas bénéfique.

c) Variation des algorithmes d'optimisation

Pour ce troisième test nous faisons varier l'algorithme d'optimisation utilisé afin de comparer leur vitesse de convergence.

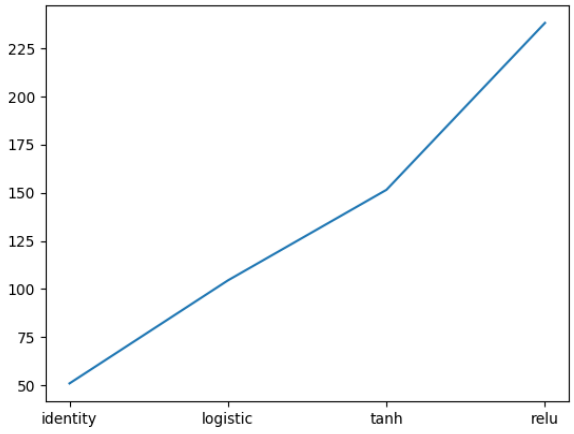


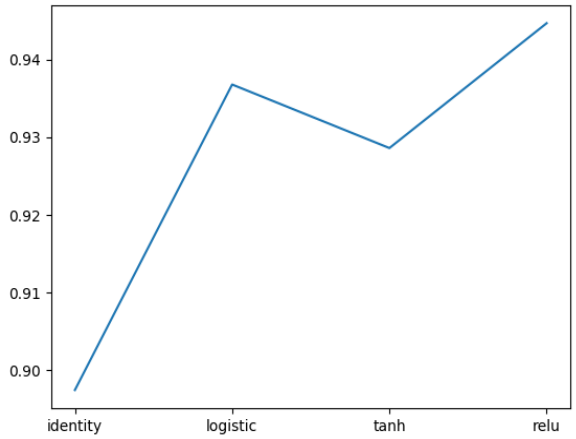


On remarque suite à ces tests que le plus rapide pour converger est l'algorithme sgd, mais c'est aussi celui qui donne le score le moins élevé : 0.83. En ratio temps d'exécution / score c'est lbfgs qui est le meilleur puisqu'il est une minute plus rapide que adam pour un score de 0.917 au lieu de 0.946.

d) Variation des fonctions d'activation

Pour ce test nous faisons varier la fonction d'activation du réseau de neurones.

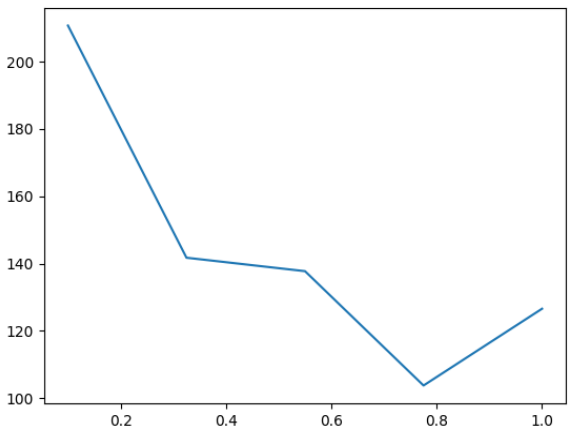


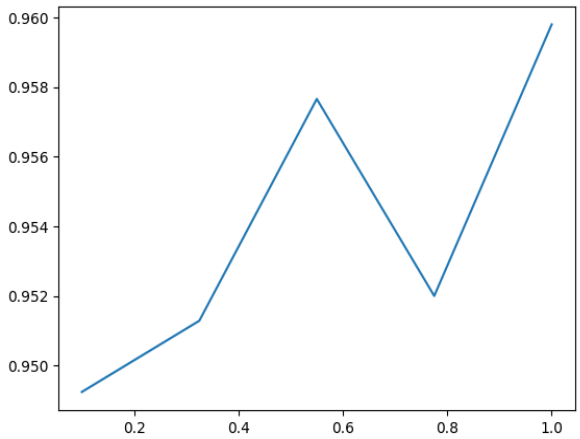


On remarque que la fonction d'activation relu est la plus adaptée à notre problème en termes de score mais qu'elle prend beaucoup de temps. Elle prend en effet deux fois plus de temps que logistic qui permet d'obtenir un score quasi-pareil 0.93 au lieu de 0.94. La fonction tan est la moins bonne puisqu'elle est longue pour le score qu'elle atteint. Identity est très rapide mais ne donne pas un score satisfaisant. On gardera logistic car c'est la meilleure en terme de ratio temps d'exécution / score.

e) Variation de la valeur de la régularisation

Dans ce dernier test nous allons tester l'influence du paramètre de régularisation ou alpha de notre réseau de neurones.





Avec un alpha plus grand, on constate que notre calcul est d'une part plus rapide, mais aussi plus efficace. On atteint alors des scores de 0.96 en presque deux fois moins de temps.

f) Matrice de confusion

```
[[2064    0    5    2    0    2    6    1   12    4]
 [   1 2296   11    7    7    2    1    8   11    9]
 [   5    6 2063   23    5    1    4   26   21    2]
 [   8    5   27 1897    2   38    0   12   31   53]
 [   6    3   19    1 1926    0   11    5    9   39]
 [  15    1    2   23    8 1802    4    4   26   20]
 [  15    2   11    2    5   28 1972    2   26    0]
 [   1    6   15    6   10    1    0 2119    5   22]
 [  19    7    6   15   10   17    3    3 1992   18]
 [   4    2    1   14   41    7    0   35    9 1947]]
```

Pour cette matrice, nous avons pris 1 couche de 50 neurones. Bien que les tests aient été faits avec une taille plus grande d'échantillon de données, on peut voir que MLP se trompe très peu sur l'identification de valeurs, comme vu avec KNN.

III – Méthode SVC, la machine à vecteurs de support

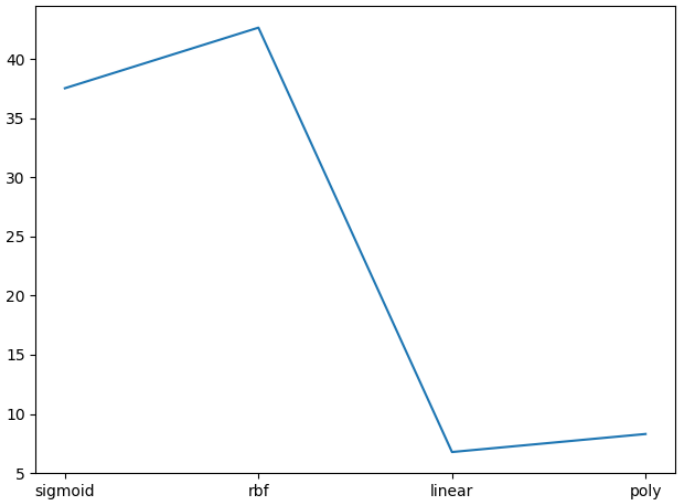
Ces derniers tests se feront sous la méthode d'apprentissage SVC, forme de SVM. Nous pourrons faire varier les paramètres suivants :

- Kernel : Le type de noyau utilisé
- C : La tolérance aux erreurs
- Gamma : Le coefficient utilisé dans les noyaux

Nous gardons également le jeu de données de taille 5000 pour accélérer les tests de calcul, avec une part de données d'entraînement toujours égale à 70%.

a) Temps d'exécution des kernels

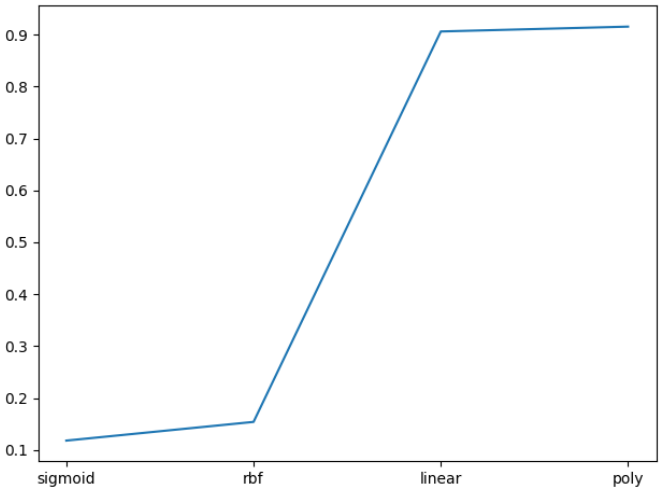
En testant chacun des kernels disponibles dans sklearn, nous avons comparé leur temps d'exécution respectifs afin de constater leur efficacité.



On constate rapidement que le kernel le plus efficace en termes de rapidité est sans conteste linear. Il devance les premiers d'un incroyable écart en s'exécutant en près de 7 secondes. Poly a cependant de très bon résultats également, légèrement supérieurs à linear. L'extrait de données étant resté le même entre les exécutions des 4 cas, nous en concluons tout de même que linear est le plus rapide. Mais cela ne veut pas dire qu'il obtient les meilleurs résultats en termes de fiabilité.

b) Score des kernels

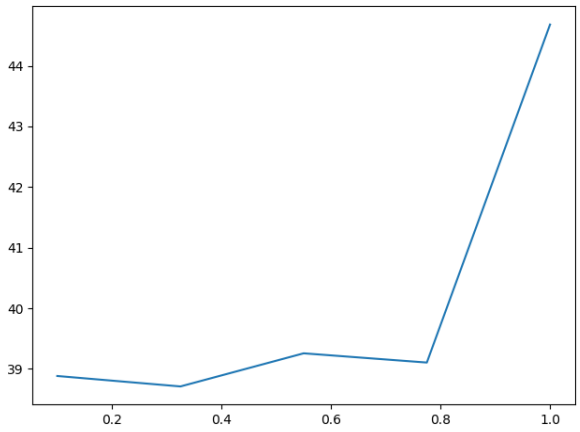
Ici nous allons comparer les scores de fiabilité obtenus par chacun de nos kernels. Sans changer l'échantillon de données test, voici les résultats obtenus.

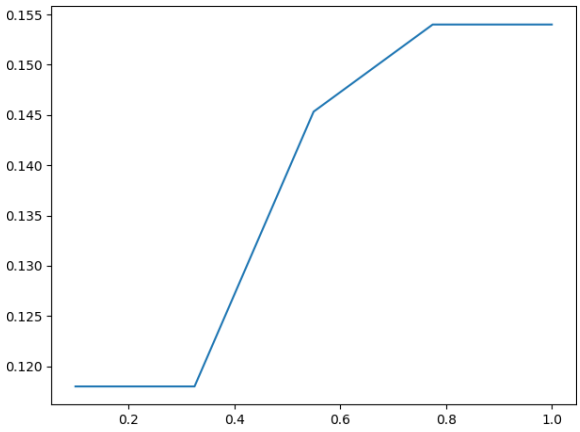


Une fois de plus linear se démarque avec poly pour leur haut taux d'efficacité ! Nous ne dépassons même pas les 20% avec sigmoid et rbf. Cela laisse penser que nous avons tout intérêt à préférer les autres car ils sont meilleurs en tout point.

c) Variation de la tolérance aux erreurs

Pour ces tests, nous avons pris le kernel rbf pour tester l'impact de la variation de C sur l'exécution du solver. Nous avons pris 5 valeurs C entre 0.1 et 1.

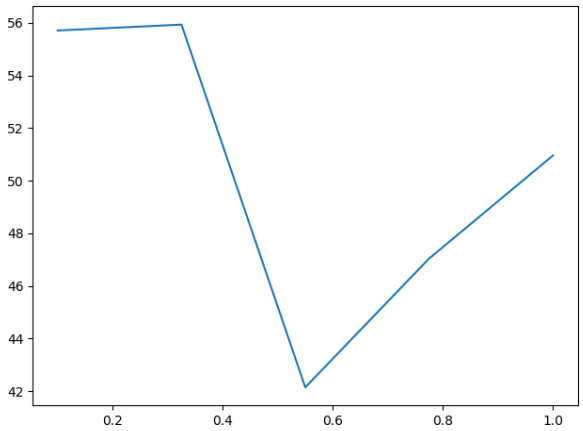


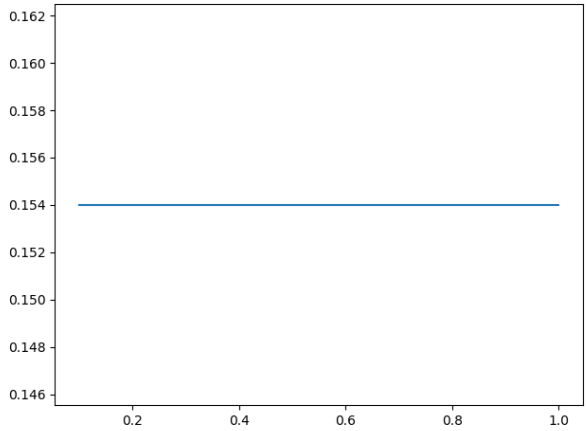


Cela nous permet d'observer le phénomène de plus près concernant la fiabilité. Celle-ci est déjà à son maximum pour 0.8. Le temps de calcul quant à lui varie très peu selon C. Il est évident qu'avec une autre acceptation des erreurs on en arrive à une fiabilité qui n'augmente plus (cela étant rbf n'était pas le kernel le plus fiable …).

d) Variation du coefficient Gamma

Ce coefficient sert pour le calcul via les kernels rbf, poly et sigmoide. Nous allons le tester avec rbf de nouveau en le faisant varier entre 0.1 et 1 à nouveu.





Cette fois, malgré un temps très variant (même assez haut dans les premiers cas) cela n'a aucune incidence sur l'efficacité du solver.

e) GridSearchCV

Ce modèle est supposé trouver de lui-même les paramètres optimaux pour le solver. Nous lui avons donné le choix entre tous les kernels et un C entre 1 et 10. Voici les résultats :

- score: 0.9153333333333333
- time: 441.23244857788086

En soi, ça a duré beaucoup de temps pour une durée équivalente à nos précédents tests avec linear et poly, donc pas plus fructueux que ça pour notre dataset.

f) Matrice de confusion

Enfin, nous allons tester la matrice de confusion sur le kernel linear, sans paramètres additionnels et pour le même extrait de dataset qu'avant.

- score: 0.906
- time: 7.316744327545166

```
[[144   0   0   0   0   2   2   0   1   0]
 [  0 161   1   0   0   1   0   0   2   0]
 [  0   1 122   2   2   0   4   2   5   1]
 [  1   0   5 144   1   5   0   0   2   2]
 [  0   3   1   0 142   0   1   0   0   7]
 [  2   2   2  14   1 111   1   0   2   0]
 [  1   0   0   0   5   2 143   0   2   0]
 [  0   0   2   0   4   1   0 161   0   9]
 [  0   6   1   4   1   4   1   1 115   1]
 [  4   0   2   1   6   0   0   4   1 116]]
```

Cette matrice nous indique par exemple que pour la seconde ligne (correspondant au chiffre 1), 161 valeurs ont été correctement identifiées, tandis que 4 ont été erronées. Globalement, on peut constater l'efficacité de notre SVC sur les données de test (30% de 5000 valeurs).

Synthèse

Nous avons conclu plusieurs choses concernant le travail des 3 méthodes sur notre jeu de données. Nous observons :

- Que la méthode la plus rapide est celle des k plus proches voisins avec des temps d'exécution qui sont entre 5 et 15 secondes et un score maximal d'un peu plus de 0,93
- Que la méthode la plus lente et la plus efficace est celle des couches de neurones avec des temps d'exécutions compris entre 150 et 600 secondes. Ces temps d'exécution commencent à être non négligeable (plus de dix fois supérieures à la méthode KNN) mais donnent des scores qui sont pour le meilleurs à 0,97. Cela étant, avec un échantillon de 5000, les scores sont bien moins bons…
- Que la méthode SVC est au final un entre deux avec des temps d'exécution compris entre 5 et 40 secondes et des scores qui vont jusqu'à 0,90 pour un kernel égal à linear ou poly.

En conclusion générale nous pouvons dire que pour notre problème, si on privilégie la rapidité, nous auront tendance à choisir la méthode des k plus proches voisins. Si on privilégie le score final, nous choisirons d'utiliser un réseau de neurones avec davantage de données. La méthode SVC est un entre-deux des deux autres méthodes qui reste très performante selon le kernel.