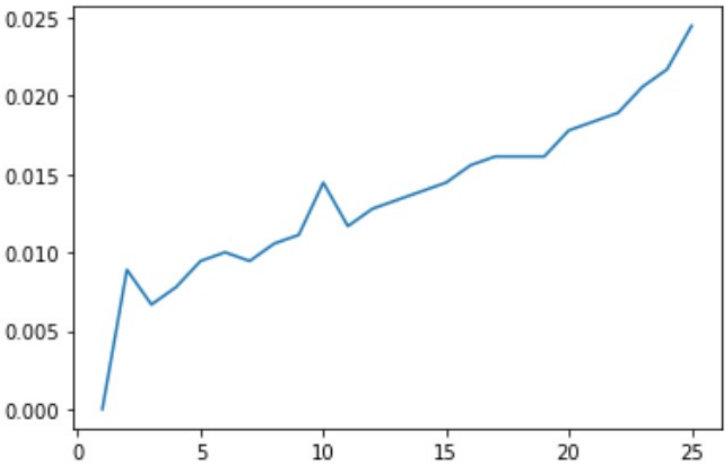
M1 Informatique AMU – TP2 ISD — 2023/24 — Rapport. **Date : 20 octobre 2023.**
Nom, prénom [+éventuel binôme], groupe TP# 1

CLASSIFICATION PAR LES *K* PLUS PROCHES VOISINS

Partie : Variation du nombre de voisins

Le code est disponible dans le fichier partie1_TP2.py.

On obtient la figure ci-contre, avec k qui varie en abscisse, et l'erreur en ordonnée.



On observe sur cette figure que plus k augmente, plus l'erreur semble augmenter, sauf quelques exceptions. Mais les erreurs restent faibles, de l'ordre de 10^{-2}. On remarque que lorsque $k=1$, l'erreur sur l'échantillon d'apprentissage est nulle : c'est normal, puisque le 1-plus proche voisin d'un exemple est lui même dans l'échantillon d'apprentissage, donc sa propre classe lui sera attribuée, donc on ne fait jamais d'erreur !

Ici, le meilleur hyper-paramètre semble être $k=3$, avec une erreur d'environ 0.007 ; mais en réalité ce n'est pas sur l'échantillon d'apprentissage que nous devrions le retenir.

Partie : Evaluation de l'erreur réelle du classifieur appris

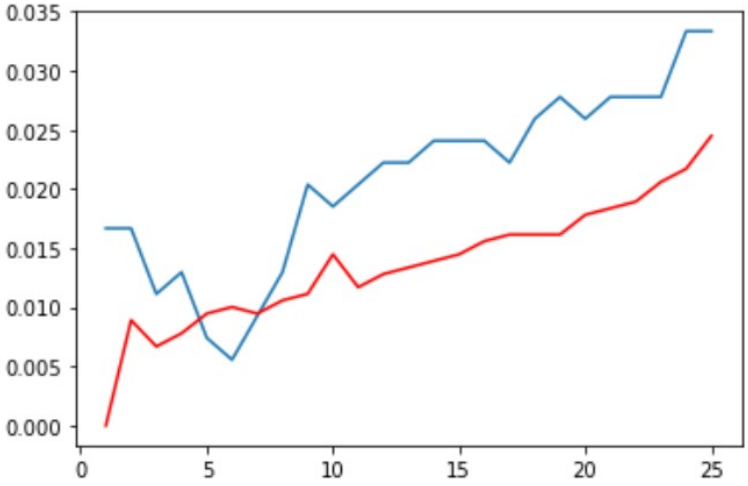
Le code produit est disponible dans le fichier partie2_TP2.py.

Initialisations différentes du hold-out. Pour nous rendre compte de l'importance du paramètre random_state (« graine ») de la fonction séparant l'échantillon, nous avons vérifié que nous obtenions le même *split* si nous appelions toujours avec cette graine à 42, et que nous obtenons effectivement un *split* différent avec cette graine à 18.

Test de l'erreur sur 75 % app – 25 % test, avec k=3. Avec la graine à 42, nous avons aléatoirement découpé le jeu d'apprentissage en 75 % pour apprendre et 25 % pour calculer une estimation de l'erreur : apprenant sur les 75 %, et testant sur les 25 % restants de ce *split*, nous avons obtenu une erreur de 0.0133 avec *k*=3 (le meilleur *k* en apprentissage dans la partie précédente). Nous avions 0.007 avec ce *k*=3 sur l'échantillon d'apprentissage : en mesurant l'erreur sur des données n'ayant pas servi à apprendre (les 25 % du jeu), on obtient une erreur presque doublée ! Cela provient de deux phénomènes conjoints : l'erreur sur

données nouvelles est plus grande naturellement, mais en plus notre modèle appris est peut être moins ajusté que précédemment puisqu'il a été appris avec moins de données.

Visualisation des erreurs sur test 25 %, avec variations de k.
Sur ce même split 75 %-25 %, graine à 42, on fait maintenant varier la valeur de *k* (en abscisse), et pour chaque valeur de *k* on affiche l'erreur faite par l'apprentissage sur ce *split* (en ordonnée). On obtient la courbe ci-contre (en rouge l'erreur sur les données d'apprentissage, en bleu sur les données test.)



L'erreur calculée sur l'échantillon test est globalement toujours plus élevée que celle calculée sur l'échantillon d'apprentissage, mais est généralement plus fiable. Ici, on obtient le meilleur score environ 0.005 pour k=6, alors que nous avions la meilleure erreur d'apprentissage pour k=3. On remarque qu'on n'a plus l'erreur nulle (en test) pour k=1 puisque dans ce cas là, l'exemple sur lequel on teste n'est pas dans l'échantillon d'apprentissage.

Répétitions sur 10 splits différents et erreur moyenne pour k=3. Nous réalisons 10 fois la même expérience que précédemment, pour *k* fixé à 3, et moyennons ces 10 erreurs mesurées sur des *splits* différents (donc apprentissage sur des exemples différents, et tests sur des exemples différents). Nous nous assurons que les 10 *splits* sont à chaque fois différents, en modifiant la graine (instruction random_state = iho dans l'itération de rang iho). Nous obtenons le résultat ci-dessous (tableau des 10 erreurs et moyenne sensée être une bonne estimation de l'erreur réelle).

```
[0.01296296 0.01111111 0.01851852 0.01481481 0.01481481 0.00925926
 0.01111111 0.01296296 0.01481481 0.01666667]
0.013703703703703708
```

Pour *k*=3, on obtient une estimation de l'erreur réelle à 0.0137, sensiblement supérieure à celle précédemment estimée par *hold-out* simple ; c'est un coup de bol puisque nous voyons ci-dessus que selon le *split*, l'erreur estimée est toujours différente.

Partie : Validation croisée

La validation croisée ci-contre, avec *k*=3, indique une estimation de l'erreur réelle à 0.0234, d'un centième au dessus de celle estimée par *hold-out* moyennée sur 10 s*plits*. L'estimation par validation croisée est

```
from sklearn.model_selection import cross_val_score
k = 3
clf = nn.KNeighborsClassifier(k)
erreur = 1. - cross_val_score(clf, X, y, cv=10)
print(erreur)
print(erreur.mean())
```

```
0.02336747361887028
```

réputée plus fiable (faible biais et robuste) que celle par *hold-out* même moyenné, mais ce n'est pas formellement prouvé.

En ajoutant le paramètre scoring = 'f1_macro' dans l'appel de cross_val_score, nous obtenons un F1-score à 0.9665 (compromis rappel / précision sur l'ensemble des classes).

Partie : Matrice de confusion

Le code produit est disponible dans le fichier partie3_TP2.py.

Nous fixons *k*=6 et *p*=5 (degré de la distance de Minkowski). Sur un seul *hold-out* 70 %-30 %, (sans répétition et moyennage) nous obtenons l'estimation et l'affichage de la matrice de confusion, de l'erreur, et du score :

Le score est excellent. Nous retrouvons 6 confusions plutôt éparpillées, mais pas de confusion majeure entre deux classes, alors que les confusions connues sur ce jeu de données sont celles entre les classes (1,7), (4,9) et (3,8), qui n'apparaissent pas ici (où la dernière classe -9- voit certains de ses exemples mal classés vers 3, 4, et 5 ; où un exemple de 8 est classé vers 1 et un autre vers 9, et enfin un exemple de 5 est classé vers 4).

```
[[53  0  0  0  0  0  0  0  0  0]
 [ 0 50  0  0  0  0  0  0  0  0]
 [ 0  0 47  0  0  0  0  0  0  0]
 [ 0  0  0 54  0  0  0  0  0  0]
 [ 0  0  0  0 60  0  0  0  0  0]
 [ 0  0  0  0  1 65  0  0  0  0]
 [ 0  0  0  0  0  0 53  0  0  0]
 [ 0  0  0  0  0  0  0 55  0  0]
 [ 0  1  0  0  0  0  0  0 41  1]
 [ 0  0  0  1  1  1  0  0  0 56]]
0.011111111111111072
0.9888888888888889
```

Pour obtenir une estimation plus fiable, il faudrait répéter l'expérience sur 10 splits différents, et moyenner.