CLASSIFICATION CHALLENGE RAPPORT : GREGOIRE CAURIER GROUPE B

INTRODUCTION :

Vu durant le td 3, l'algorithme des K plus proches voisins devaient venir complémenter les explications que l'on avait reçus dans le cadre de l'apprentissage supervisé. Nous avons donc commencé par étudier un dataset Iris.data qui contenait les informations de fleurs répartit en 3 classes avant de passer à des dataset plus conséquents.

MES PREMIERES APPROCHES SUR LE KNN :

Durant la séance de td3, j'ai essayé de voir quelle similarité avec les premiers algorithmes vus pouvaient m'inspirer pour l'implémentation de mon KNN. Je suis donc au début partis au début avec une classe individu dont l'objet était caractérisé par 3 variables float et une dernière variables string pour la classe de la fleur. Une classe permettant selon moi un code plus structuré et plus clair lorsqu'il faut récupérer la valeur d'une certaine variable. Cette première implémentation fonctionnait dès la fin du tp, j'arrivais à prédire une classe à une fleur selon ces k plus proches voisins. J'ai par la suite décidé d'abandonner la classe individu et privilégier plutôt le travail avec des listes puisque les échantillons des nouveaux dataset du défi comportait beaucoup plus de variables et les listes étaient un moyen plus rapide pour la lecture des données .csv à mon goût.

MES FONCTIONS ET MES CHOIX, PARTIE ENTRAINEMENT :

La première partie était donc d'entrainer mon algo knn avec le dataset data.csv. Pour se faire, j'utilisais plusieurs fonctions :

- Loaddata() qui me servait de récupérer les données du dataset et de les rentrer dans une liste. Elle me retournait donc 2 listes contenant des listes (toutes les lignes du fichier data.csv et pretest.csv).
- Separate(data) qui prenait en paramètre les listes provenant de LoadData() afin de séparer en 2 listes mon dataset. Une liste destiner pour l'apprentissage et une autre pour tester mon algorithme. J'ai commencé par une répartition de 10% en test et 90% en apprentissage selon les conseils de mon chargé de td.
- AlgoKnn(data,test,k) qui prenait en paramètre ma liste d'apprentissage et un individu de ma liste de test afin de retourner les k plus proches voisins dans mon apprentissage de cet échantillon de

test. Je calculais donc la distance euclidienne entre ce test et tous les individus de ma liste d'apprentissage et j'ajoutais dans un tableau toutes les distances et l'échantillon de data auquel cette distance correspondait. La construction de ce tableau me permettait donc d'utiliser la fonction sort avec comme clé la distance euclidienne et ainsi récupérer directement les k plus proches échantillons des k plus petites distances en triant qu'une seule liste (sans faire un tri à bulles simultané entre 2 listes).
J'ajoutais ensuite dans une liste knn les classes de ces k plus proches échantillons trouvés puis avec la fonctions ClasseDominante() je retournais la classe en plus grande occurrence dans cette liste.

- ConfusionMatrix(k,afficherMat) qui permet de retourner une matrice de confusion du problème. J'utilise pour ce faire un dictionnaire avec le nom des classes en clé et les indexes des lignes et colonnes en valeur et j'utilise la bibliothèque numpy pour l'affichage de la matrice et qui me sera très utile pour retourner la qualité de l'algorithme (opération facile sur la diagonale avec numpy). La variable afficherMat est booléenne et sert juste à afficher la matrice si sa valeur est True. Cette fonction me retourne la valeur de la diagonale et me sera utile lorsque je chercherai le meilleure k possible.

- BestK() qui est une fonction qui va me permettre d'obtenir le k pour lequel j'obtiens le plus de bonnes réponse soit le k pour lequel la somme de la diagonale de ma matrice de confusion est la plus élevée. Pour cela pour un k allant de 1 à 20 je calcule la qualité de l'algorithme pour chaque k et l'ajoute dans une liste stat. Je retourne à la fin l'index qui contient la valeur max dans stat qui est mon meilleur k (dans mes paramétrages d'apprentissage j'obtiens k=4).

En résumé : ma paramétrisation pour l'apprentissage était :

- Dataset test/validation : 10% de data.csv + 10% de pretest.csv =>160 lignes
- Dataset apprentissage : tout le reste soit => 1446 lignes
- J'obtenais un taux de précision de 93,75% soit 150 sur 160 classes bien prédite pour k=4.

MA PARTIE TEST ET CLASSIFICATION :

Ensuite, le challenge était de déterminer les classes des individus du fichier FinalTest.csv en suivant la meilleure paramétrisation possible effectuée dans la partie apprentissage. J'ai choisi de combiné les fichiers data.csv et pretest.csv afin de générer un dataset assez large et étendu pour avoir une classification plus générale. En plus de celles utilisées en amont j'ai dû générer d'autre fonctions :

- LoadData2() qui faisait la même chose que LoadData() mais renvoyer cette fois ci un seul et même dataset composé des données des 2 fichiers.
- GetClasse() qui est une fonction d'écriture de fichier .txt qui me permettait de générer mon fichier solution. J'appelais la fonction knn qui me retournais la classe et l'écrivais dans une ligne dans mon fichier solution pour chaque élément de finalTest.csv.

En résumé : ma paramétrisation pour ce challenge était la suivante :

- Dataset apprentissage : data.csv + pretest.csv => 1606 lignes
- Test : finalTest.csv => 3000 lignes
- K=4
- Mon fichier solution est généré en 4 secondes.

CONCLUSION

Pour conclure, j'ai trouvé ce challenge très intéressant surtout la partie entrainement qui nous permettait de trouver les meilleurs paramètres pour nos futures prédictions. Le sujet était clair et nous invitait à faire des recherches et résoudre un vrai problème avec la meilleure optimisation.