Rapport KNN

Partie lecture du fichier

- **Description du programme**

« train » est le nom du fichier qui contient les ID, caractéristiques et class de chaque point. Pour le lire on utilise la fonction open de python.

On veut créer une matrice « val_train » qui contient tous les éléments du fichier « train ». Pour commencer, on va utiliser la fonction « readlines » qui créer une liste M où chacun de ses éléments est une ligne du fichier « train » de type string.

Ensuite, à l'aide d'une boucle, on va transformer chaque string de la liste M en une sous-liste de float à l'aide de la fonction « split » qui le transforme en liste, la fonction « strip » qui enlève les « \n » de chaque string et la fonction « map » qui transforme chaque caractéristiques de la sous-liste en un float. On ajoute toutes ces valeurs à la matrice val_train et on obtient alors ce que l'on cherchait.

Enfin, on va créer une autre matrice val_train_2 qui est la matrice val_train mais sans l'ID, qui est contenu dans la première colonne.

Le 2ème bloc « test » est le même programme que celui du bloc « train »

- **Changement par rapport à mon premier programme**

Après m'être amélioré en python, dans l'expression de la boucle « for » à la ligne 7, au lieu de faire un « range(1,len(M)) » pour enlever la ligne de mots, j'ai remplacé le « range » par un slicing « M[1 :] » pour plus d'efficacité et de lisibilité.

De plus, au lieu de faire une autre boucle pour convertir les nombres de type string en float, j'ai remplacé la boucle par la fonction « map » de python pour directement convertir les éléments en float afin d'avoir une meilleure lisibilité, efficacité et prise de mémoire.

Partie algorithme KNN

- **Description du programme**

On peut y voir 2 différentes distances. La distance euclidienne qui est la racine carré de la somme des différences entre les 2 points au carré et la distance de Manhattan qui est la somme des valeurs absolues des différences entre les 2 points.

Ensuite on y voit l'algorithme des knn, elle prend en arguments un point P_ref qui est le point auquel on veut déterminer la classe et un entier k qui est le nombre de points voisins que l'on gardera pour déterminer la classe de P_ref.

Tout d'abord, on crée une liste list_dist qui est une liste de tuples contenant la distance à P_ref en premier puis en deuxième l'indice du point de val_train_2 avec lequel on mesure la distance.

Ensuite, on range la liste list_dist par ordre croissant des distances à l'aide la fonction « sorted » de python et de la fonction « lambda » qui retourne la distance de chaque tuple dans list_dist. On peut alors prendre les k voisins ayant la plus petite distance avec P_ref.

Ensuite, on crée le dictionnaire « Classe » pour connaître la poids ou importance de chaque classe dans la détermination de la classe de P_ref. A chaque itération de la boucle de la ligne 56, on relève la classe du point dans val_train_2 avec lequel on a trouvé une des k plus petites distances et on détermine le poids qu'il aura en fonction de sa distance avec P_ref : Poids = 1/(distance = 1e-6) (le 1e-6 permet de ne pas diviser par 0), puis on ajoute ce poids dans le dictionnaire « Classe » en fonction de la classe trouvé juste avant.

Enfin l'algorithme des knn retourne la clé (donc la classe) du maximum des valeurs du dictionnaire Classe.

- **Changement par rapport à mon premier programme**

Au début j'utilisais le nombre d'occurrences pour déterminer la classe d'un point P_ref. Mais en faisant des recherches, j'ai décidé d'utiliser un poids car je pense que pour les k grands, le précision serait bien meilleure car la distance des points augmenteraient et donc leur poids serait plus petit, donc les voisins les plus proches de P_ref auraient plus d'importance. De plus, le poids aide aussi au fait que pour les k paires ou certains k, le nombre d'occurrences pour une classe par rapport à une autre pourrait être égal et donc que déterminer une classe ne serait pas possible.

Ensuite, j'ai aussi décidé d'essayer si la distance de Manhattan donnerait des résultat meilleurs que la distance Euclidienne, ce qui est le cas et je vais le montrer un peu plus loin dans le document.

Utilisation du programme

- **Description du programme**

On décide d'écrire le programme dans un fichier csv « envoi.csv » à l'aide de cette fois-ci aussi la fonction « open » de python. On écrit à la première ligne : « Id, Label\n » puis dans chaque itération de la boucle for, on écrit dans le fichier « envoi » : « ID,classe », où l'ID est trouvé grâce à l'indice de la fonction « enumerate » qu'on utilisera dans la matrice « val_test » (qui contient encore l'ID) et où la classe est trouvé en utilisant la fonction knn écrite auparavant en utilisant comme P_ref les points de « val_test_2 ».

- **Analyse des résultats et détermination de k**

On obtient les précisions selon différents k, distance, poids et occurrences :

	Distance Euclidienne Occurrences	Distance Euclidienne Poids	Distance de Manhattan Poids
k = 1	0.98731		
k = 2	0.98731		
k = 3	0.97756		0.98536
k = 4	0.98146	0.98146	0.98341
k = 5	0.97463		0.98243
k = 6	0.97463		
k = 7	0.97365	0.97658	0.98048
k = 8	0.97463		
k = 9	0.96975		
k = 10	0.97073		

On remarque que lorsqu'on a rajouté le poids à la place des occurrences, la précision est la même pour un k = 4, ce qui est assez petit, mais qu'elle est différente et meilleure pour k = 7 car le système de poids devient utile d'après ce que j'ai expliqué auparavant.

Et si on modifie en plus du poids, la distance Euclidienne par la distance de Manhattan, on obtient une précision encore meilleure pour les mêmes k. Cela peut être dû à l'accumulation des erreurs de calcul dans la distance Euclidienne qu'il n'y aurait pas dans celle de Manhattan car on fait une valeur absolue.

De plus, on remarque que plus le k augmente, plus la précision diminue dans les 3 cas. Donc si on prend un k trop grand c'est mauvais, mais si k est trop petit on peut faire du surapprentissage.

Conclusion

En conclusion, j'ai alors décidé de prendre k = 3 avec la distance de Manhattan et la pondération pour utiliser l'algorithme des knn afin de déterminer la classe d'un élément et obtenir la meilleure précision possible dans mes capacités.