Rapport de Développement éfficace

Description des implémentations

Implémentation de k-Nn

Dans le travail réalisé, on y trouve deux classes qui ont des méthodes qui permettent de réaliser les calculs de distances **Manhattan** et **Euclidien**. Ces méthodes sont ensuite utilisées dans la classe **KNNAlgo** dans la méthode **calculerDistance(DataPoint a, DataPoint b, String typeCalcul, String xAxis, String yAxis)** qui retourne la distance entre deux points selon le calcul choisi. Dans cette même classe, on trouve des méthodes permettant de récupérer les k voisins les plus proches, la classe majoritaire parmi les voisins, des méthodes permettant de classifier un ou plusieurs éléments et une méthode pour la robustesse, qui retourne un taux de réussite de classification entre 0 et 1. Toutes ces méthodes sont utilisées dans la méthode **classifier()**, de la classe **ScatterView**. Cette méthode permet à l'utilisateur de mettre les paramètres qu'il souhaite tels que le type de calcul, le nombre k de voisins pour classifier et calculer la robustesse.

Validation croisée

Une explication de votre méthode de validation croisée (comment sont calculés les pour- centages que vous donnez ?)

Dans notre classe KnnAlgo est implémentée la méthode ValidationCroisee qui prend en paramètre ; les données, le k choisi, le type de calcul que l'on souhaite effectuer(Manhattan ou Euclidien), la valeur présente sur l'axe des abscisses et la valeur sur l'axe des ordonnées. La méthode créée le maximum de sous ensemble possible (nommée n) dans les données qui sont passées en paramètre et en retire un (n-1), dans la finalité la méthode tourne sur n-1 ensemble. Parmis ces sous-ensembles, on choisit au hasard un des éléments pour lequel on va récupérer son ancien attribut (sur lequel on qualifie) ensuite on effectue l'algorithme knn pour ce point choisit au hasard et on récupère sa nouvelle qualification. Si son ancienne qualification et sa nouvelle qualification sont égales on considère que l'algorithme a bien fonctionné. Pour finir on divise le nombre de cas pour lesquels l'algorithme a eu bon avec le nombre de sous ensemble qui ont été testé.

Exemple
Nos données contiennent 35 points distincts, on souhaite qualifier grâce à l'algorithme knn pour un k égal à 5. Il est possible d'avoir 35/5+1 (5 voisins + 1 point à classifier) ensemble différents soit 5 sous-ensembles de données. Nous avons donc 4 sous-ensembles à tester, admettons que l'algorithme trouve la bonne classification pour 2 des sous-ensembles alors notre robustesse vaut 2/4 soit 50% de réussite.

Choix du meilleur k

Les résultats pour différents k et les deux distances (Manhattan et Euclidienne), pour les iris et les pokemons. Eventuellement, distances basées sur différents attributs et/ou pondération. Conclusion sur le meilleur choix à faire. Pour les données pokemon, la robustesse sera établie pour plusieurs (au moins 2) classifications (différents choix de catégorie).

Choix du meilleur k pour les Iris

Calcul de la distance \ k	1	3	5	7	9	11
Manhattan	0.99	0.99	0.95	0.96	0.97	0.96
Euclidienne	0.99	0.99	0.96	0.96	0.96	0.96

Dans le cas des Iris, il est préférable d'utiliser la distance [distance] avec un k égal à [nombre] qui permet d'avoir plus de chance de trouver la bonne classification avec notre algorithme Knn.

Choix du meilleur k pour les Pokémon

légendaire ou non (axes: sp_attack et sp_defense)

Calcul de la distance \ k	1	3	5	7	9	11
Manhattan	0.99	0.99	0.94	0.94	0.93	0.93
Euclidienne	0.99	0.99	0.94	0.94	0.94	0.94

Dans le cas des Pokémon, il est préférable d'utiliser la distance [distance] avec un k égal à [nombre] qui permet d'avoir plus de chance de trouver la bonne classification avec notre algorithme Knn avec la qualification de légendaire ou non pour le pokémon.

Type 1 (axes: sp_attack et sp_defense)

Calcul de la distance	1	3	5	7	9	11
Manhattan	0.99	0.99	0.94	0.94	0.93	0.94
Euclidienne	0.99	0.99	0.94	0.94	0.94	0.94

Dans le cas des Pokémon, il est préférable d'utiliser la distance [distance] avec un k égal à [nombre] qui permet d'avoir plus de chance de trouver la bonne classification avec notre algorithme Knn avec la qualification du type pour le pokémon.

Efficacité

Pour l'implémentation de notre algorithme, nous avons choisi d'utiliser des ArrayList.
Une première raison de notre choix est que les **ArrayList** permettent d'avoir un accès direct aux éléments dans la liste, avec une complexité de O(1). L'Arraylist à des méthodes qui permettent de simplifier l'ajout, la suppression et l'accès aux données. L'opération de l'ajout à la fin a une complexité de O(1). Dans le cadre de l'implémentation d'un algorithme k-NN, on cherche à pouvoir avoir accès aux données et ajouter à la liste de données les nouveaux points ajoutés. Donc l'utilisation de ArrayList semble être le choix le plus efficace contrairement à, par exemple, une liste chaînée, qui aurait été un meilleur choix dans le cas où nous aurions besoin de réaliser des insertions ou suppression au début ou milieu de la liste.