Rendu CLASSIFICATION pour la SAÉ S3.02

Groupe H6

Membres :

- **Dassonville Valentin** : conception du squelette du logiciel, codage de la couche d'abstraction, de la partie MVC, des views + des tests
- **Debusschère Sacha** : chargement des données CSV (CSVLoader) et ses tests + méthodes de distance des données + autres tests.
- **Larsonnier Cédric** : codage des classes implémentant les différentes données, création de tests par rapport à ces classes
- **Petit Gwénaël** : méthode Knn, tests correspondants et tests distances. Première version fonctionnelle de la vue et tests divers

Analyse des données

Préparation et typage des données

Les fichiers de données nous sont donnés au format CSV. CSV est un fichier texte représentant des données tabulaires sous forme de valeurs séparées par un caractère délimiteur, le plus souvent une virgule ou un point-virgule.
La première ligne contient les noms de chaque colonne, nous informant sur ce que représentent les valeurs de chaque colonne.
Ensuite, chaque ligne du fichier correspond à une donnée composée de valeurs qui décrivent la donnée.

Pour le projet, il nous est demandé d'implémenter 2 types de données qui sont Iris et Titanic. On remarque très vite en regardant les fichiers, que les différentes valeurs peuvent s'apparenter à des String, des int ou double.
Nous avons donc commencé par créer plusieurs types auxquels toutes les valeurs d'une colonne y sont reliés :

- Number : Les valeurs sont des nombres, décimaux ou non
- Boolean : Les valeurs sont true ou false
- Enum : Les valeurs sont contenus dans une liste connus à l'avance
- String : Les valeurs sont des chaînes de caractères

Pour Iris, nous avons donc :

Colonne	sepal.length	sepal.width	petal.length	petal.width	variety
Type	Number	Number	Number	Number	Enum

Et pour Titanic :

Colonne	PassengerId	Survived	Pclass	Name	Sex	Age
Type	String	Number	Number	String	Enum	Number

Colonne	SibSp	Parch	Ticket	Fare	Cabin	Embarked
Type	Number	Number	String	Number	String	Enum

Normalisation des données

La normalisation des données consiste pour toutes les valeurs d'une même colonne à les représenter par une valeur entre 0 et 1.
Tous les types de données, à l'exception de String, sont normalisables.

La méthode de normalisation est différente pour chaque donnée :

Type Boolean :
- La valeur true devient 1
- La valeur false devient 0

Type Enum :
- Chaque valeur de la liste est représentée par un nombre entre 0 et 1.

Type Number :
- La valeur maximale de la colonne est égale à 1
- La valeur minimale de la colonne est égale à 0.
- Les autres valeurs sont égale à $\frac{valeur - min}{max - min}$ (min et max non normalisés)

Distance entre les données

On appelle la distance, le nombre représentant la similarité entre deux données. Plus ce nombre est grand, plus les deux données ne sont pas similaires et à l'inverse plus ce nombre est petit, plus les deux données sont similaires.

Dans notre cas, la distance est utilisée dans l'algorithme de classification k-NN et plusieurs méthodes pour la calculer existent.
Nous en avons implémenté deux :

- Distance Euclidienne : $d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$
- Distance de Manhattan : $d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$

La distance est donc représentée par une méthode de calcul ainsi que les colonnes qui sont utilisées dans le calcul de distance (le i dans la formule ci-dessus).
Les méthodes de distance utilisent les valeurs normalisées dans leurs calculs.

Nous avons implémenté pour chaque type de donnée une distance par défaut :
- Iris : Distance Euclidienne -> sepal.length, sepal.width, petal.length, petal.width
- Titanic : Distance Euclidienne -> Pclass, Sex, Age, Parch, Embarked

Nous pouvons également, si nous le voulons, créer une autre distance avec des colonnes différentes ou une méthode de calcul différente pour arriver à des résultats différents.

Chargement des données

Notre système de chargement des données trouve tout seul le type de donnée à partir fichier CSV et charge les données sous le bon type.
Pour ce faire, une Enum stocke tous les types de données qu'elle connaît et compare les noms des colonnes sur la première ligne du fichier pour trouver à quel type de donnée correspond le fichier CSV.
Cela permet de ne pas avoir à coder une fonction de chargement de données pour tous les types de données mais une seule fonction qui gère tous les types.

Implémentation de k-NN

Pour implémenter k-NN, nous avons créé une classe qui prend en paramètres un nombre k et une distance qui sera utiliser pour connaître les k plus proches voisins.

Ensuite nous avons une méthode **classifyPoint** prenant en paramètres le point à classifier, la liste des points du modèle, qui sont donc les données d'apprentissage, et également la colonne représentant la donnée sur laquelle on classifie le point (le résultat de la classification est retourné par la fonction).

La méthode va donc récupérer les k voisins les plus proches pour ensuite déterminer la valeur, correspondant à la colonne, qui apparaît le plus de fois parmi les voisins.

Notre implémentation a un point fort qui est qu'elle est générique grâce à la couche d'abstraction du modèle. C'est-à-dire que l'on a une seule implémentation qui gère tous les types de données.

Robustesse de vos modèles

La robustesse est une donnée importante car elle nous informe de la qualité du modèle et donc si la classification de nouveaux points est plutôt sûr ou non.

Notre méthode de robustesse s'appuie sur la validation croisée. C'est-à-dire que l'on va séparer notre modèle en sous-groupes et tester chaque sous-groupes sur l'ensemble de tous les autres sous-groupes.

Dans notre cas, nous avons choisi de séparer le modèle en 5 sous-groupes pour les raisons suivantes :

- Avoir assez de données dans chaque sous-groupes
- Avoir assez de sous-groupes
- Avoir un temps de calcul qui ne soit pas trop long

Nous pouvons représenter la répartition des sous-groupes comme ceci :

Gr 1	Gr 2	Gr 3	Gr 4	Gr 5

Pour tester le groupe 1, nous utiliserons les groupes 2, 3, 4 et 5 comme données d'apprentissage.

Gr 1	Gr 2	Gr 3	Gr 4	Gr 5

Nous allons maintenant décrire la procédure du calcul de la robustesse :

- Utiliser le k et la colonne sélectionner pour k-NN
- Créer une copie du modèle d'apprentissage
- Mélanger de façon aléatoire les données du modèle
- Créer 5 sous-groupes
- Ajouter les données dans les sous-groupes de façon à ce qu'il y ait à le même nombre de données dans chaque sous-groupe (avec une donnée de marge)
- Sur chaque sous-groupe :
    - Pour chaque donnée, on applique k-NN avec comme données d'apprentissage toutes les données des autres sous-groupes

    - On compare si le type retourner par k-NN est le même que le type de la donnée tester
    - Si c'est différent, on ajoute une erreur sur le sous-groupe
- On calcule le taux de réussite sur chaque sous-groupe
- On fait une moyenne des taux de réussite des sous-groupes (afficher en pourcentage)

Appliquons cela sur un modèle de test de type Iris contenant 21 données avec comme paramètres :

- un k égale à 3
- la colonne variety pour la classification
- une Distance Euclidienne sur les colonnes : sepal.length, sepal.width, petal.length, petal.width

Groupe 0
    VERSICOLOR[6.2,2.9,4.3,1.3]
    VERSICOLOR[6.7,3.1,4.4,1.4]
    SETOSA[5.0,3.2,1.2,0.2]
    SETOSA[5.0,3.5,1.6,0.6]
    VIRGINICA[7.2,3.2,6.0,1.8]
Groupe 1
    VIRGINICA[7.4,2.8,6.1,1.9]
    VERSICOLOR[6.0,2.9,4.5,1.5]
    SETOSA[4.9,3.0,1.4,0.2]
    VIRGINICA[5.8,2.8,5.1,2.4]
Groupe 2
    VIRGINICA[6.0,3.0,4.8,1.8]
    SETOSA[5.4,3.4,1.7,0.2]
    SETOSA[5.0,3.3,1.4,0.2]
    VERSICOLOR[5.0,2.0,3.5,1.0]
Groupe 3
    SETOSA[5.1,3.5,1.4,0.2]
    VIRGINICA[6.1,2.6,5.6,1.4]
    VERSICOLOR[6.7,3.1,4.7,1.5]
    VERSICOLOR[6.9,3.1,4.9,1.5]
Groupe 4
    VIRGINICA[6.3,2.9,5.6,1.8]
    SETOSA[5.0,3.4,1.5,0.2]
    VERSICOLOR[6.1,3.0,4.6,1.4]
    VIRGINICA[5.6,2.8,4.9,2.0]
Groupe 0 accurency = 80.0%
Groupe 1 accurency = 100.0%
Groupe 2 accurency = 50.0%
Groupe 3 accurency = 75.0%
Groupe 4 accurency = 100.0%
Final accurency : 81.0%

On obtient ici une moyenne de 81% de réussite mais qui varie d'environ 10%.
Ici pour éviter d'avoir un affichage trop long, nous avons pris que 21 données mais plus de données rendrait la robustesse plus fiable.

D'autres résultats obtenus avec les mêmes paramètres :

```
Groupe 0
        SETOSA[5.4,3.4,1.7,0.2]
        VERSICOLOR[5.0,2.0,3.5,1.0]
        SETOSA[5.0,3.5,1.6,0.6]
        VIRGINICA[5.6,2.8,4.9,2.0]
        VIRGINICA[7.2,3.2,6.0,1.8]
Groupe 1
        VERSICOLOR[6.2,2.9,4.3,1.3]
        VIRGINICA[6.3,2.9,5.6,1.8]
        VERSICOLOR[6.9,3.1,4.9,1.5]
        VERSICOLOR[6.7,3.1,4.7,1.5]
Groupe 2
        SETOSA[5.0,3.2,1.2,0.2]
        SETOSA[5.0,3.4,1.5,0.2]
        VIRGINICA[7.4,2.8,6.1,1.9]
        VIRGINICA[6.1,2.6,5.6,1.4]
Groupe 3
        SETOSA[5.1,3.5,1.4,0.2]
        VERSICOLOR[6.7,3.1,4.4,1.4]
        VIRGINICA[5.8,2.8,5.1,2.4]
        VERSICOLOR[6.1,3.0,4.6,1.4]
Groupe 4
        SETOSA[4.9,3.0,1.4,0.2]
        VERSICOLOR[6.0,2.9,4.5,1.5]
        VIRGINICA[6.0,3.0,4.8,1.8]
        SETOSA[5.0,3.3,1.4,0.2]
Groupe 0 accurency = 80.0%
Groupe 1 accurency = 100.0%
Groupe 2 accurency = 75.0%
Groupe 3 accurency = 100.0%
Groupe 4 accurency = 100.0%
Final accurency : 90.99999999999999%
```

```
Groupe 0
        SETOSA[4.9,3.0,1.4,0.2]
        VERSICOLOR[5.0,2.0,3.5,1.0]
        VERSICOLOR[6.1,3.0,4.6,1.4]
        VERSICOLOR[6.2,2.9,4.3,1.3]
        VIRGINICA[5.8,2.8,5.1,2.4]
```

Groupe 1
    SETOSA[5.4,3.4,1.7,0.2]
    SETOSA[5.0,3.4,1.5,0.2]
    VERSICOLOR[6.7,3.1,4.4,1.4]
    VIRGINICA[6.3,2.9,5.6,1.8]
Groupe 2
    VIRGINICA[6.0,3.0,4.8,1.8]
    VIRGINICA[5.6,2.8,4.9,2.0]
    SETOSA[5.1,3.5,1.4,0.2]
    SETOSA[5.0,3.3,1.4,0.2]
Groupe 3
    VERSICOLOR[6.7,3.1,4.7,1.5]
    VIRGINICA[7.2,3.2,6.0,1.8]
    VIRGINICA[6.1,2.6,5.6,1.4]
    VIRGINICA[7.4,2.8,6.1,1.9]
Groupe 4
    VERSICOLOR[6.9,3.1,4.9,1.5]
    SETOSA[5.0,3.2,1.2,0.2]
    SETOSA[5.0,3.5,1.6,0.6]
    VERSICOLOR[6.0,2.9,4.5,1.5]
Groupe 0 accurency = 80.0%
Groupe 1 accurency = 100.0%
Groupe 2 accurency = 75.0%
Groupe 3 accurency = 25.0%
Groupe 4 accurency = 100.0%
Final accurency : 76.0%