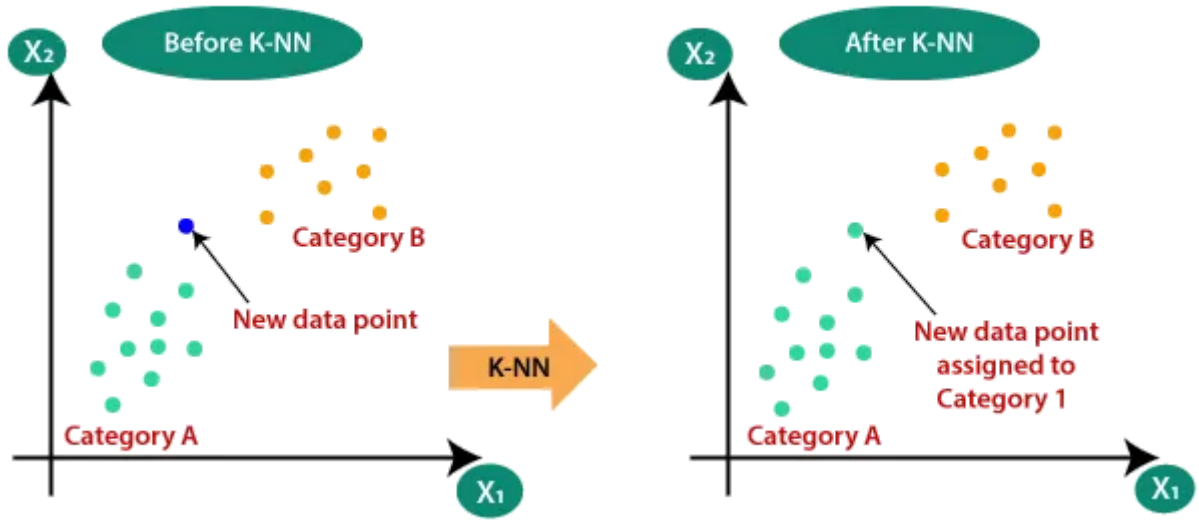
Rapport sur l'implémentation de la classification des données.

Groupe J5



Membres :
- Mehdi ZAIDI :
-> Implémentation du chargement des données et de l'algorithme k-NN

- Benoît MISPLON :
-> Implémentation du calcul de robustesse

- Loïc LECOINTE :
-> Implémentation de la classification

- Sacha BOUTON :
-> Implémentation des calculateurs (Manhattan[7], Euclidienne[8]).

Université de Lille

Lille iut

Analyse des données

Le chargement des données, que ce soit pour les données de référence (connue) ou pour les données à classifier (non connue), se fait depuis un fichier au format CSV.
Nous avons une énumération permettant d'assigner pour chaque type de données supportées (Iris, passager du Titanic, pokémon), le nom de son Csv de référence.
Ainsi, lorsque l'utilisateur veut charger des données à classifier, il choisit depuis cette énumération le type de données qu'il souhaite classifier ainsi qu'un fichier au format CSV.
Le programme construit deux datasets à partir de cette action, un dataset servant de référence et un dataset contenant les données non classifiées. C'est depuis ces deux datasets que nous allons baser le reste du fonctionnement afin de calculer les distances, générer les plus proches voisins et enfin classifier les données.

En ce qui concerne l'initialisation des données et leurs valeurs, nous utilisons la librairie opencsv utilisée en TP afin de construire pour chaque ligne du CSV, un objet java (Data) contenant pour chaque colonne du CSV un attribut.

Ainsi, le chargement du fichier CSV :

```
1   "sepal.length","sepal.width","petal.length","petal.width","variety"
2   5.1,3.5,1.4,0.2,
```

Créera l'objet suivant :

```java
public class IrisData extends AbstractData {
    @CsvBindByName(column = "sepal.length")
    protected double sepalLength = 5.1;

    @CsvBindByName(column = "sepal.width")
    protected double sepalWidth = 3.5;

    @CsvBindByName(column = "petal.length")
    protected double petalLength = 1.4;

    @CsvBindByName(column = "petal.width")
    protected double petalWidth = 0.2;

    @CsvBindByName(column = "variety")
    protected IrisVariety variety = null;
```

Nous utilisons parfois des énumérations pour des ensembles courts de valeur comme la variété des Iris.

Le calcul des distances se fait selon deux calculs différents : Manhattan[7] ou Euclidienne[8], Nous envoyons dans une méthode “distance” deux données, une donnée de référence, une donnée à classifier. En utilisant de la reflection nous accédons aux valeurs des différents attributs des objets afin d’y appliquer la formule correspondante et d’en ressortir la distance.

Exemple avec les deux données suivantes :

```java
public class IrisData extends AbstractData {
    @CsvBindByName(column = "sepal.length")
    protected double sepalLength = 4.7;

    @CsvBindByName(column = "sepal.width")
    protected double sepalWidth = 2.4;

    @CsvBindByName(column = "petal.length")
    protected double petalLength = 2.5;

    @CsvBindByName(column = "petal.width")
    protected double petalWidth = 1.2;

    @CsvBindByName(column = "variety")
    public IrisVariety variety = IrisVariety.SETOSA;
```

```java
public class IrisData extends AbstractData {
    @CsvBindByName(column = "sepal.length")
    protected double sepalLength = 5.1;

    @CsvBindByName(column = "sepal.width")
    protected double sepalWidth = 3.5;

    @CsvBindByName(column = "petal.length")
    protected double petalLength = 1.4;

    @CsvBindByName(column = "petal.width")
    protected double petalWidth = 0.2;

    @CsvBindByName(column = "variety")
    protected IrisVariety variety = null;
```

Si nous envoyons ces données par l'intermédiaire d’un calcul des distances de Manhattan[7] ainsi que les champs “sepalLength, variety, petalLength”, le calcul sera alors :

distance = |4.7-5.1| + |1| + |2.5-1.4| (1 car les variétés sont différentes)
= 2.5

La normalisation suit le même principe, en utilisant de la reflection.

Lorsque nous souhaitons normaliser les données, nous appelons la méthode “normalizeData” du WorkingDataset[4] correspondant. La normalisation va dans un premier temps calculer les “deltas”, c'est-à-dire, trouver le minimum et le maximum de chaque attribut pour les données de référence ainsi que le “delta”, la différence du minimum et du maximum.

Lors de la normalisation, si une donnée à classifier ne rentre pas dans cette plage (si un de ses attributs est plus grand que le maximum ou plus petit que le minimum), alors cette donnée ne sera pas classifiée.

Enfin, une fois ces “deltas” calculés, la normalisation se fait avec la formule suivante :
Normalisation -> value - minimum / (maximum - minimum)
Dénormalisation -> value * (maximum - minimum) + minimum

Ces nouvelles valeurs sont directement écrites dans la donnée à l’aide de la reflection.

Implémentation de k-NN

Afin d'implémenter cet algorithme, nous avons réfléchi à une solution afin d'avoir le plus de modularité possible lors de son utilisation.

En effet, notre défi était de pouvoir modifier n'importe quelle pièce du puzzle afin de garder un algorithme totalement fonctionnel mais différent.
De ce fait, nous avons découpé l'algorithme des plus proches voisins plusieurs sous modules, l'assemblage de ces modules forment l'algorithme final.

Pour arriver à cette abstraction "presque" totale, nous avons utilisé des interfaces dans le but de rendre le code le plus modulable possible en réduisant la quantité de code à écrire et en évitant de nombreuses conditions difficiles à maintenir dans le futur.

Ainsi, notre algorithme k-NN se décompose en quatre sous modules:

- Distance (geometry)
- Calculateur (calculator)
- Classifieur (classifier)
- Robustesse (strengthCalculator)

Fonctionnement général :

Lorsqu'un utilisateur souhaite classifier des données, il les envoie au programme, sous forme d'un fichier CSV des données.
Le programme construit un WorkingDataset(4) contenant ces données et récupère le ReferenceDataset(3) correspondants à ces données.
Le WorkingDataset(4) se verra assigner un attribut sur lequel classer les données ainsi qu'une liste d'attributs servant au calcul des distances entre une WorkingData(1) du WorkingDataset(4) et une ReferenceData du ReferenceDataset(3).

Distance :

Le sous module distance concerne le calcul de la distance entre deux points (une WorkingData(1) et une ReferenceData(3)) en fonction de la liste d'attribut du WorkingDataset(4).

Ce sous module se décompose lui-même en sous modules étant les différents types de calcul de distances, dans notre cas, Manhattan(7) ou Euclidienne(8).
D'autres types de calcul peuvent être ajoutés très facilement en implémentant l'interface IGeometry.

Calculateur :

Le calculateur se sert de la distance afin d'assigner à chaque WorkingData(1) du WorkingDataset(4), sa distance avec chaque ReferenceData(2) du ReferenceDataset(3).

Nous avons découpé cette partie en un sous module afin de pouvoir y apporter des modifications plus facilement. Le projet contient notamment un calculateur aléatoire afin de pouvoir faire du classement de données de manière aléatoire.

Classifieur :

Le classifieur se sert des plus proches voisins de chaque WorkingData[1] afin de dresser une liste de la catégorie la plus présente au sein de ses voisins et lui attribue cette dernière.

Robustesse :

Le calcul de robustesse est lui aussi un sous module.
En effet, dans notre projet nous utilisons le KFold Cross-Validation[6], cependant il en existe d'autres. Ce découpage en module permet de les implémenter plus facilement par la suite.
Nous reparlerons de ce module dans la prochaine partie de ce rapport.

k-NN Algorithme[5] :

Nous avons découpé cet algorithme en plusieurs sous modules cependant, il est lui-même un sous module du projet.
En effet, nous avons prévu le programme afin de pouvoir implanter d'autres algorithmes assez facilement. L'algorithme SVM nous fait notamment de l'œil.

Le fonctionnement de cet algorithme est assez simple.
Nous utilisons le calculateur afin de récupérer la liste des distances pour chaque WorkingData[1] avec chaque ReferenceData[2], de cette liste nous en récupérons les "k" plus proches que nous envoyons ensuite au classifieur afin de classer la donnée.

Robustesse de vos modèles

Dans un premier temps nous avions essayé un ShuffleSplit Cross-Validation, cette technique consiste à mélanger puis à découper le ReferenceDataset[3] en deux parties : une partie de Train, et une partie de Test. Une fois l'entraînement puis l'évaluation complétée, on rassemble nos données, on les mélange à nouveau, puis on redécoupe le ReferenceDataSet[3] dans les mêmes proportions que précédemment. On répète ainsi l'action pour autant d'itérations de Cross-Validation que l'on désire. On peut ainsi retrouver plusieurs fois les mêmes données dans le jeu de validation à travers les Itérations.
Le principal problème est que si les classes sont déséquilibrées, alors on risque de manquer d'informations dans le jeu de Validation.

Nous avons donc voulu essayer une autre technique qui se nomme Leave One Out Cross Validation.
Cette technique est un cas particulier du K-Fold (que nous avons utilisé au final). En fait, il s'agit du cas où K = "nombre d'échantillons du ReferenceDataset[3]". Par exemple, si un ReferenceDataset[3] contient 100 échantillons, alors K =100. L'algorithme s'entraîne donc sur 99 échantillons et s'évalue sur le dernier. Elle procède ainsi à 100 entraînements (sur les 100 combinaisons possibles) ce qui peut prendre un temps considérable à la machine.
C'est pourquoi cette technique est déconseillée donc au final nous avons pris la décision d'implémenter le KFold Cross-Validation[6].

Cette technique consiste donc à mélanger le Dataset, puis à le découper en k parties (K-Fold). Par exemple, si notre ReferenceDatset[3] contient 100 échantillons,et que notre k vaille 5, alors nous aurons 20 paquets de 5 échantillons.
Ensuite, notre algorithme s'entraîne sur 19 paquets, puis s'évalue sur le paquet restant, et alterne les différentes combinaisons de paquets possibles.
Au final, il effectue donc un nombre k d'entraînements (20 entraînement dans cette situation).
Si nous avions 103 échantillons dans notre ReferenceDataset[3] alors il y aurait donc 20 paquets de 5 échantillons et 1 paquet de 3, ce qui veut dire que notre algorithme s'entraînera 21 fois au total.
Cette technique présente un léger désavantage: si le ReferenceDataset[3] est hétérogène et comprend des classes déséquilibrées, alors il se peut que certains split de Cross-Validation ne contiennent pas les classes minoritaires par exemple, si un Dataset de 100 échantillons contient seulement 10 échantillons de la classe 0, et 90 échantillons de la classe 1, alors il est possible que sur 5 paquets, certains ne contiennent pas d'échantillons de la classe 0.

Démo d’un calcul de robustesse :

```
Strength calculation launched with groups of 10 data for dataset : DefaultIris
Started generation of groups...
15 groups generated.
Launch of strength test for group : 0
Got 10.0 good data classified out of 10 strength = 100.0
Launch of strength test for group : 1
Got 10.0 good data classified out of 10 strength = 100.0
Launch of strength test for group : 2
Got 8.0 good data classified out of 10 strength = 80.0
Launch of strength test for group : 3
Got 9.0 good data classified out of 10 strength = 90.0
Launch of strength test for group : 4
Got 10.0 good data classified out of 10 strength = 100.0
Launch of strength test for group : 5
Got 10.0 good data classified out of 10 strength = 100.0
Launch of strength test for group : 6
Got 9.0 good data classified out of 10 strength = 90.0
Launch of strength test for group : 7
Got 10.0 good data classified out of 10 strength = 100.0
Launch of strength test for group : 8
Got 10.0 good data classified out of 10 strength = 100.0
Launch of strength test for group : 9
Got 9.0 good data classified out of 10 strength = 90.0
Launch of strength test for group : 10
Got 10.0 good data classified out of 10 strength = 100.0
Launch of strength test for group : 11
Got 10.0 good data classified out of 10 strength = 100.0
Launch of strength test for group : 12
Got 9.0 good data classified out of 10 strength = 90.0
Launch of strength test for group : 13
Got 9.0 good data classified out of 10 strength = 90.0
Launch of strength test for group : 14
Got 10.0 good data classified out of 10 strength = 100.0
End of the strength test, strength : 95.0
```

Annexes de vocabulaire

(1) WorkingData : Donnée à classifier, dont nous voulons connaître à quelle catégorie elle appartient.

(2) ReferenceData : Donnée de référence, chaque WorkingData se verra comparer à celle-ci afin de connaître la distance entre les deux.

(3) ReferenceDataset : Regroupement de ReferenceData, utilisé par le WorkingDataset.

(4) WorkingDataset : Regroupement de WorkingData. Il contient aussi un ReferenceDataset ainsi qu'un champ sur lequel classifier les données et une liste de champs sur lesquels calculer les distances.

(5) K-NN : Algorithme des k plus proches voisins.

(6) K-Fold : Validation croisée K-Fold est une méthode de calcul de robustesse d'un algorithme K-NN.

(7) Manhattan : Calcul la distance entre deux points correspondante au nombre de déplacements élémentaires horizontaux ou verticaux pour rejoindre ces deux points.

(8) Euclidienne : Calcul la distance entre deux points correspondante à la longueur qui sépare deux points (une droite directe entre les deux points).

Zaidi Mehdi - Bouton Sacha - Misplon Benoît - Lecointe Loïc.