|  |
| --- |
| S3.02 –  Développement d’application  Rendu - Classification |
| BUT Info S3 – IUT de Lille Nathan Hallez  Alexandre Herssens  Maxence Stiévenard  Rémi Vautier |



# Sommaire

[Équipe J2 - *Belamcanda* 3](#_Toc120305607)

[Nathan Hallez 3](#_Toc120305608)

[Alexandre Herssens 3](#_Toc120305609)

[Maxence Stiévenard 3](#_Toc120305610)

[Rémi Vautier 3](#_Toc120305611)

[Analyse des données 4](#_Toc120305612)

[Typage des données 4](#_Toc120305613)

[Énumérations créées 4](#_Toc120305614)

[Calcul des distances 4](#_Toc120305615)

[Normalisateurs 4](#_Toc120305616)

[Comparateurs 4](#_Toc120305617)

[Implémentation de k-NN 5](#_Toc120305618)

[Robustesse de vos modèles 6](#_Toc120305619)

## Équipe J2 - *Belamcanda*

### Nathan Hallez

LOREM ISPUM hihihi

### Alexandre Herssens

LOREM ISPUM hihihi

### Maxence Stiévenard

LOREM ISPUM hihihi

### Rémi Vautier

LOREM ISPUM hihihi

## Analyse des données

Dans cette partie, vous devez relater tout ce qui concerne le chargement et la préparation des données pour vos deux problèmes: présentation des types de données, plages de valeurs et/ou d’énumération, détails sur les distances utilisées et la normalisation éventuellement faite.

### Typage des données

Après avoir consulté toutes les données qui nous ont été fournies dans un tableur, les champs numériques et textuels que nous avons considérés comme basiques, se sont vu attribuer les types **int** pour les entiers, **double** pour les décimaux ainsi que le type **String** pour les chaînes de caractères.

Pour les champs textuels où leurs types étaient moins évidents que les autres, nous nous sommes aidés d’un **filtre**.

S’il n’y avait qu’un nombre limité de valeurs se répétant, nous avons considérés l’option de **l’énumération**, et pour les champs contenant seulement des valeurs uniques, nous leur avons attribuées le type **String**.

#### Énumérations créées

Comme expliqué ci-dessus, nous avons crées des énumérations afin d’implémenter notre algorithme **k-NN**.

Commençons par les **Iris** où nous nous sommes servis d’une seule et unique énumération que l’on a nommé **IrisVariety.** Commele nom l’indique, à l’intérieur de cette énumération nous retrouvons les trois types d’Iris mis à notre disposition par les données qui nous ont étés fournies : *Setosa, Versicolor* et *Virginica.*

Nous nous sommes ensuite intéressés aux données du **Titanic** où nous en avons déduis deux énumérations, la première nommé **Gender** contenant le genre des passagers : *Male* ou *Female.*

La seconde énumération, nommée **Embarked** signifiant le quai d’embarquement des passagers : *S*, *Q*, *C* ou *Null.*

### Calcul des distances

Afin de calculer la distance entre deux Iris ou deux passagers du Titanic, nous avons le choix entre deux types de distance : la **distance Euclidienne** et la **distance de Manhattan** qui somment les différences obtenues grâce aux comparateurs propres à chaque colonne.

Nous avons ainsi délégué les responsabilités aux **colonnes** : lorsqu’un jeu de données est défini, nous préconfigurons les colonnes avec un **normalisateur** et **une méthode de calcul de distance**.

#### Normalisateurs

Nous avons la possibilité d’utiliser 5 normalisateurs. Mais afin de les utiliser nous avons dû créer une classe que nous avons nommée **Amplitude** nous permettant, en grande partie, de mettre à jour la valeur de l’amplitude

Au niveau des normalisateurs, nous avons en avons utilisés 5 :

* **BooleanNormalizer** servantà normaliser la distance soit à 0 ou soit à 1.
* **OrdinalNormalizer** remplaçant une valeur de l’énumération par son classement dans l’énumération.
* **StandardScoreNormalizer** remplaçant une valeur numérique par son écart à la moyenne.
* **NumberNormalizer** prenant les bornes et instancies tout entre 0 et 1 en fonction du minimum et du maximum.
* **NullNormalizer** qui à pour principe de ne rien faire, mais nous permet tout de même d’instancier un normalisateur null.

#### Comparateurs

Les comparateurs nous permettent de faire la différence sur un attribut donné entre deux points.

Nous en utilisons deux :

* **DrasticComparator** permettant de nous indiquer si les attributs ont identiques ou non. Si ils le sont la valeur prend 0 sinon elle prend 1.
* **NonDrasticComparator**: si les attributs ne sont pas identiques, elle retourne la valeur absolue.

#### Poids

Pour le calcul de la distance entre deux points, nous avons le choix de donner plus ou moins d’importance à une colonne donnée, ou bien tout simplement d’ignorer certaines colonnes.

## Implémentation de k-NN

Dans cette partie vous expliquerez votre implémentation de k-NN, et vous focaliserez sur toute optimisation effective que vous avez mise en œuvre dans votre projet.

## Robustesse de vos modèles

Dans cette partie, vous devez évaluer la robustesse de vos modèles et détaillant votre protocole pour l’évaluer. Vous aurez au minimum un scénario d’exécution de votre code qui charge vos données et sort dans la console toutes les valeurs qui apparaissent dans cette section.

Avant de pouvoir calculer la robustesse de nos modèles, il faut savoir que chaque modèle contient une liste de données d’entrainement (forcément remplie) et une de test (parfois vide). Deux calculs de robustesse sont dès lors possibles :

* Le premier : si la liste des données de test **contient des points**, le calcul de robustesse peut s’effectuer à l’aide de ces dernières.
* Le deuxième s’il n’y a aucune donnée de test, nous on utilise la **cross-validation.**

La cross-validation consiste à générer des sets de test à partir des données d’entrainement qui sont découpées en **cinq** sous listes réarrangées.

Ainsi, nous réalisons 5 calculs de robustesse pour chacune des combinaisons possibles d’un sous-jeu de test entrainé par les 4 autres sous-jeux.

La robustesse finale est ainsi **la moyenne des cinq sous-calculs**.

Suite à ça, chaque point du jeu de test est **catégorisé** et le résultat est comparé à sa vraie catégorie. La robustesse correspond donc au **de catégorisations réussis** sur l’ensemble des données de test.