



**PROGRAMMATION ORIENTÉE OBJET II**  
**INF11207 (MS)**  
**UNIVERSITÉ DU QUÉBEC À RIMOUSKI**  
**DÉPARTEMENT DE MATHÉMATIQUE,**  
**D'INFORMATIQUE ET DE GÉNIE**  
**RAPPORT**  
**CLASSIFICATION AUTOMATIQUE DES**  
**GRAINS DE BLÉ**

ÉTUDIANT I : ATTA Fidèle  
ÉTUDIANT II : RASANDIMANANA Tafta

PROFESSEUR :  
Yacine Yaddaden, Ph. D.

Février 2026

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Analyse du problème et cahier des charges</b>	<b>4</b>
2.1	Cahier des charges . . . . .	4
2.2	Gestion de projet avec Trello . . . . .	4
<b>3</b>	<b>Gestion de version avec Git et GitHub</b>	<b>4</b>
3.1	Organisation du dépôt . . . . .	4
3.2	Accès au code source . . . . .	5
<b>4</b>	<b>Modélisation UML</b>	<b>6</b>
4.1	Classe Grain.cs . . . . .	7
4.2	Énumération TypeDeGrain.cs . . . . .	7
4.3	Diagramme de classes pour l'algorithme KNN . . . . .	7
<b>5</b>	<b>Implémentation de la classification KNN</b>	<b>7</b>
5.1	Architecture générale . . . . .	7
5.1.1	Interface IDistance . . . . .	7
5.1.2	DistanceEuclidienne.cs . . . . .	8
5.1.3	DistanceManhattan.cs . . . . .	8
5.2	Classe ClassifierKnn.cs . . . . .	9
5.3	Classes auxiliaires . . . . .	10
5.3.1	Echantillon.cs . . . . .	10
5.3.2	Voisin.cs . . . . .	11
5.3.3	EnsembleDonnees.cs . . . . .	11
5.4	Traitemet des données CSV . . . . .	12
<b>6</b>	<b>Tests et validation</b>	<b>13</b>
6.1	Jeu de données . . . . .	13
6.2	Résultats de classification . . . . .	13
<b>7</b>	<b>Program.cs</b>	<b>14</b>
7.1	Menu principal . . . . .	14
<b>8</b>	<b>Conclusion</b>	<b>16</b>
<b>9</b>	<b>Références</b>	<b>17</b>
9.1	Documentation KNN . . . . .	17
9.2	Matrice de confusion . . . . .	17
9.3	JSON : Sérialisation et écriture . . . . .	17
9.4	Spectre.Console . . . . .	17

# 1 Introduction

Ce projet a pour but de créer une application capable de classer des grains de céréales en fonction de leurs propriétés géométriques. Pour ce faire, nous avons choisi l'algorithme KNN (k-Nearest Neighbors), un algorithme de classification supervisée à la fois simple et efficace.

## 2 Analyse du problème et cahier des charges

### 2.1 Cahier des charges

Le projet répond aux exigences suivantes :

- Importation de données à partir d'un fichier CSV.
- Implémentation de l'algorithme KNN avec deux mesures de distance.
- Architecture orientée objet respectant les meilleures pratiques.
- Interface console pour la présentation des résultats.
- Gestion de version avec Git et GitHub.

### 2.2 Gestion de projet avec Trello

Pour organiser notre travail à distance, nous avons utilisé Trello comme outil de gestion de projet.

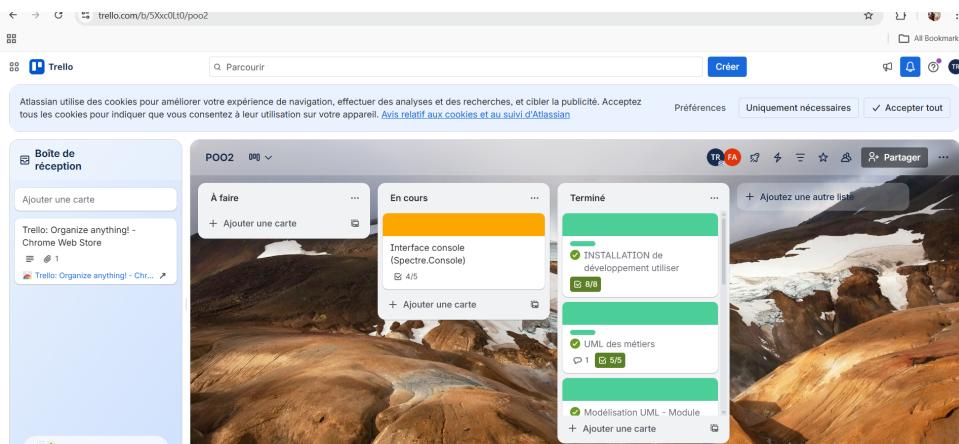


FIGURE 1 – Tableau Trello avec les tâches listées et leur répartition

Lien du tableau : <https://trello.com/b/5Xxc0Lt0/poo2>

## 3 Gestion de version avec Git et GitHub

Pour assurer un travail collaboratif efficace et sécurisé, nous avons mis en place un workflow Git structuré.

### 3.1 Organisation du dépôt

Le dépôt GitHub contient tout le code source, la documentation et les ressources nécessaires au projet.

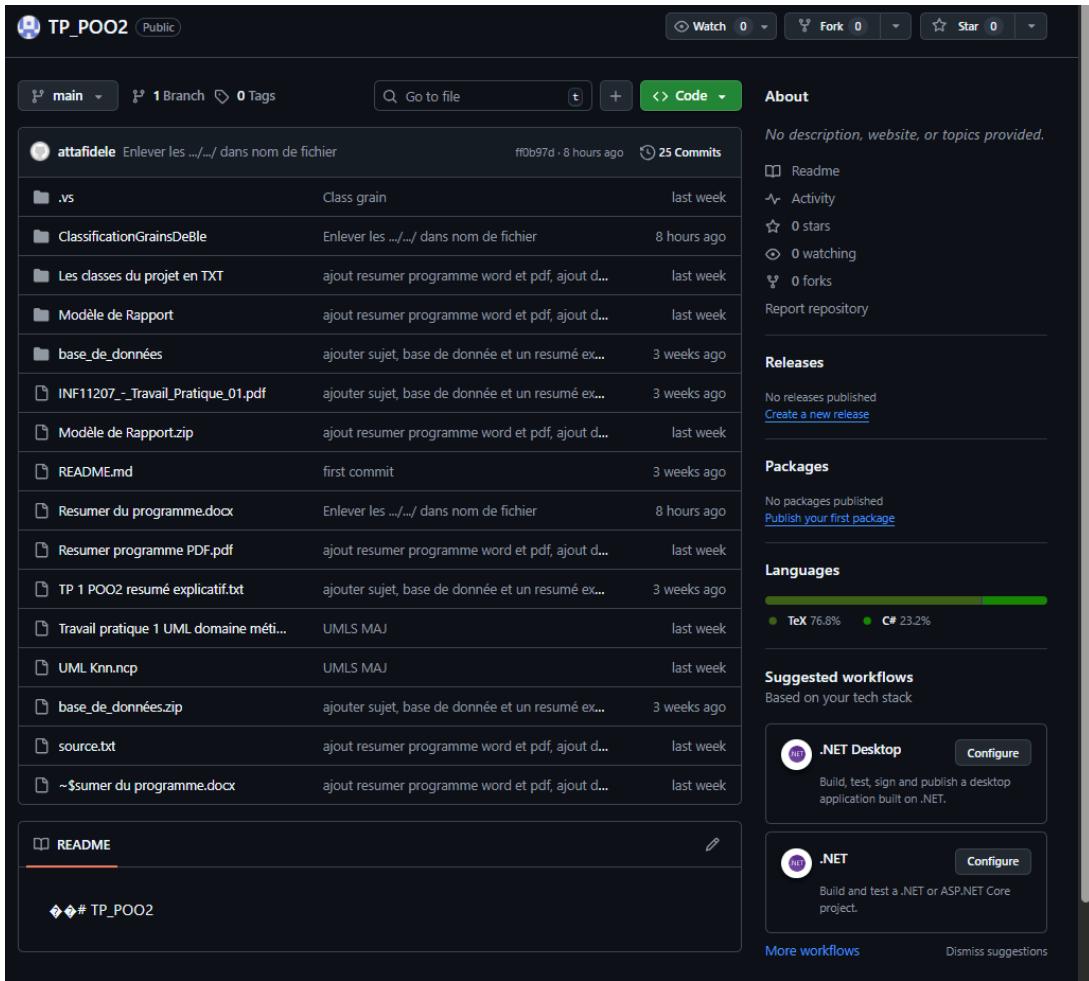


FIGURE 2 – Capture d’écran du dépôt GitHub montrant la structure du projet

### 3.2 Accès au code source

Le code source complet est disponible à l’adresse suivante :

[https://github.com/Rablou03/TP\\_P002.git](https://github.com/Rablou03/TP_P002.git)

## 4 Modélisation UML

Nous présentons ici les diagrammes UML du projet, incluant toutes les classes de métier.

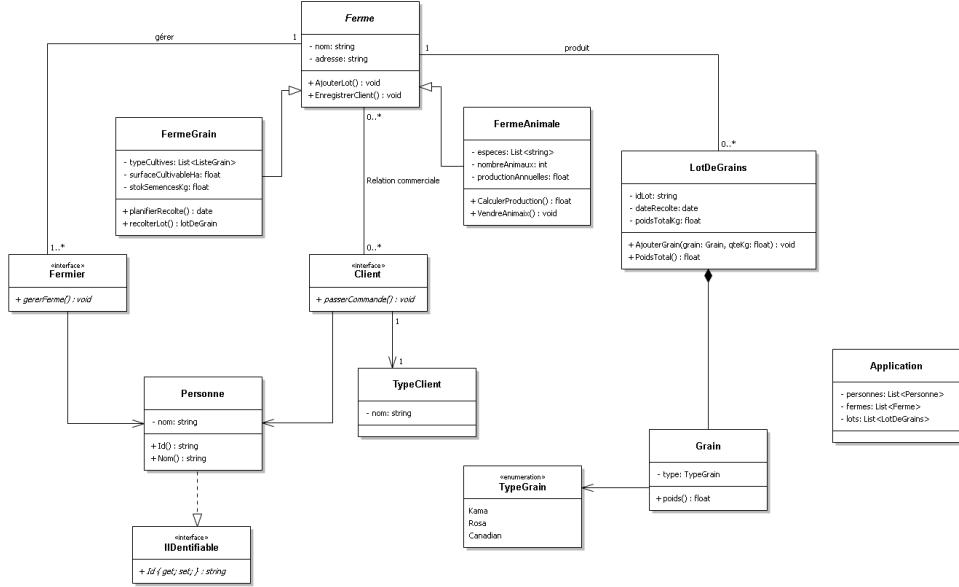


FIGURE 3 – Diagramme UML du domaine métier

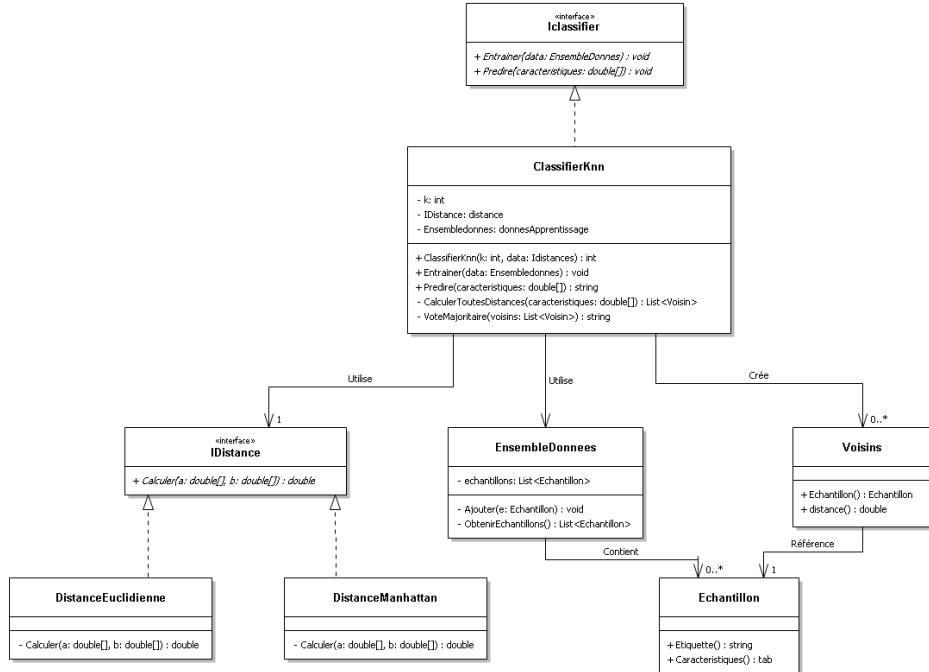


FIGURE 4 – Diagramme UML de l'architecture KNN

## 4.1 Classe Grain.cs

Cette classe modélise un grain avec ses caractéristiques géométriques. **Attributs :**

- TypeDeGrain
- Area
- Perimeter
- Compactness
- LongueurNoyau
- LargeurNoyau
- AsymetryCoefficient
- GrooveLength

## 4.2 Énumération TypeDeGrain.cs

```
public enum TypeDeGrain
{
    Kama,
    Rosa,
    Canadian
}
```

## 4.3 Diagramme de classes pour l'algorithme KNN

L'architecture de l'algorithme KNN suit le pattern Strategy pour la gestion des distances.

# 5 Implémentation de la classification KNN

## 5.1 Architecture générale

### 5.1.1 Interface IDistance

Interface définissant le contrat pour les calculs de distance :

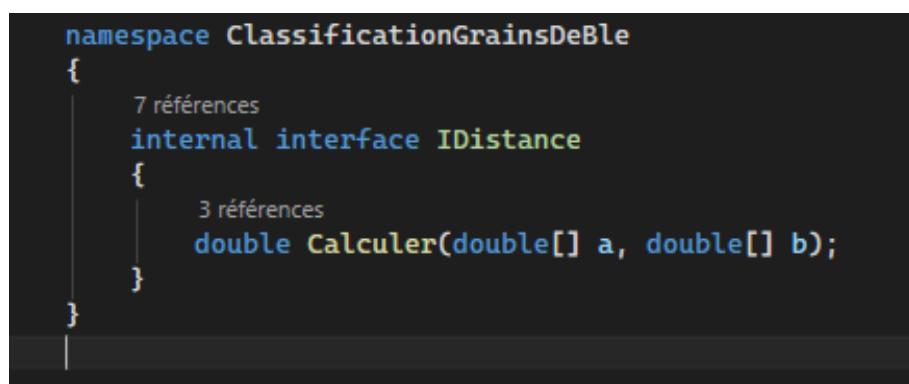


FIGURE 5 – Interface IDistance

### 5.1.2 DistanceEuclidienne.cs

Implémentation de la distance euclidienne :

$$d(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^n (a_i - b_i)^2}$$

```
namespace ClassificationGrainsDeBle
{
    2 références
    internal class DistanceEuclidienne : IDistance
    {
        2 références
        public double Calculer(double[] a, double[] b)
        {
            double somme = 0;
            for (int i = 0; i < a.Length; i++)
            {
                somme += Math.Pow(a[i] - b[i], 2);
            }
            return Math.Sqrt(somme);
        }
    }
}
```

FIGURE 6 – Classe DistanceEuclidienne

### 5.1.3 DistanceManhattan.cs

Implémentation de la distance de Manhattan :

$$d_1(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^n |a_i - b_i|$$

```

namespace ClassificationGrainsDeBle
{
    1 référence
    internal class DistanceManhattan : IDistance
    {
        2 références
        public double Calculer(double[] a, double[] b)
        {
            double somme = 0;
            for (int i = 0; i < a.Length; i++)
            {
                somme += Math.Abs(a[i] - b[i]);
            }
            return somme;
        }
    }
}

```

FIGURE 7 – Classe DistanceManhattan

## 5.2 Classe ClassifierKnn.cs

C'est le cœur du système de classification. **Attributs :**

- IDistance distance - Stratégie de calcul de distance
- EnsembleDonnees donneesApprentissage - Données d'entraînement
- int k - Nombre de voisins à considérer

**Méthodes principales :**

- public void Entrainer(EnsembleDonnees data) - Stocke les données d'apprentissage
- public string Predire(double[] caractéristiques) - Prédit le type d'un grain
- public List<Voisin> ListeTousVoisin(double[] caractéristiques) - Calcule tous les voisins
- public List<Voisin> TrierListVoisinParDistance(List<Voisin> voisins) - Tri les voisins (QuickSort)
- public TypeDeGrain VoteMajoritaire(List<Voisin> voisins) - Vote majoritaire parmi les k voisins

```

namespace ClassificationGrainsDeBle
{
    7 références
    internal class ClassifierKnn : IClassifier
    {
        IDistance distance;
        EnsembleDonnees training;
        int k;

        4 références
        public ClassifierKnn(int k, IDistance distance)
        {
            this.k = k;
            this.distance = distance;
        }

        3 références
        public void Entrainer(EnsembleDonnees donnees)
        {
            training = donnees;
        }

        1 référence
        public List<Voisin> ListeTousVoisin(double[] caractéristiques)
        {
            List<Voisin> liste = new List<Voisin>();

            foreach (Echantillon e in training.ObtenirEchantillon())
            {
                double d = distance.Calculer(caractéristiques, e.Caractéristiques);
                liste.Add(new Voisin(e, d));
            }

            return liste;
        }

        3 références
        public List<Voisin> TrierListVoisinParDistance(List<Voisin> voisins)
        {
            if (voisins.Count <= 1)
                return voisins;

            Voisin pivot = voisins[0];

            List<Voisin> plusPetits = new List<Voisin>();
            List<Voisin> plusGrands = new List<Voisin>();

            for (int i = 1; i < voisins.Count; i++)
            {
                if (voisins[i].Distance < pivot.Distance)
                    plusPetits.Add(voisins[i]);
                else

```

```

                plusGrands.Add(voisins[i]);
            }

            List<Voisin> resultat = new List<Voisin>();
            resultat.AddRange(TrierListVoisinParDistance(plusPetits));
            resultat.Add(pivot);
            resultat.AddRange(TrierListVoisinParDistance(plusGrands));

            return resultat;
        }

        1 référence
        public TypeDeGrain VoteMajoritaire(List<Voisin> voisins)
        {
            TypeDeGrain typeDuGrain = new TypeDeGrain();
            int compteurKama = 0;
            int compteurRosa = 0;
            int compteurCanadian = 0;

            foreach (var voisin in voisins)
            {
                if (voisin.Echantillon.Étiquette.ToString() == "Kama")
                {
                    compteurKama++;
                }
                if (voisin.Echantillon.Étiquette.ToString() == "Rosa")
                {
                    compteurRosa++;
                }
                if (voisin.Echantillon.Étiquette.ToString() == "Canadian")
                {
                    compteurCanadian++;
                }
            }

            if (compteurKama >= compteurRosa && compteurKama >= compteurCanadian)
                typeDuGrain = TypeDeGrain.Kama;
            else if (compteurRosa >= compteurKama && compteurRosa >= compteurCanadian)
                typeDuGrain = TypeDeGrain.Rosa;
            else
                typeDuGrain = TypeDeGrain.Canadian;

            return typeDuGrain;
        }

        2 références
        public TypeDeGrain Prédire(Echantillon e)
        {
            List<Voisin> voisins = ListeTousVoisin(e.Caractéristiques);
            List<Voisin> voisinsTries = TrierListVoisinParDistance(voisins);

            List<Voisin> kVoisins = new List<Voisin>();
            for (int i = 0; i < k; i++)
            {
                kVoisins.Add(voisinsTries[i]);
            }

            return VoteMajoritaire(kVoisins);
        }
    }
}

```

FIGURE 8 – Classe ClassifierKnn

### 5.3 Classes auxiliaires

#### 5.3.1 Echantillon.cs

Représente un échantillon avec ses caractéristiques et son étiquette.

```

namespace ClassificationGrainsDeBle
{
    15 références
    internal class Echantillon
    {
        3 références
        public double[] Caractéristiques { get; set; }

        5 références
        public TypeDeGrain Étiquette { get; set; }

        1 référence
        public Echantillon(double[] caractéristiques, TypeDeGrain étiquette)
        {
            Caractéristiques = caractéristiques;
            Étiquette = étiquette;
        }
    }
}

```

FIGURE 9 – Classe Echantillon

### 5.3.2 Voisin.cs

Structure un voisin avec l'échantillon et la distance calculée.

```
namespace ClassificationGrainsDeBle
{
    19 références
    internal class Voisin
    {
        4 références
        public Echantillon Echantillon { get; set; }

        3 références
        public double Distance { get; set; }

        1 référence
        public Voisin(Echantillon e, double d)
        {
            Echantillon = e;
            Distance = d;
        }
    }
}
```

FIGURE 10 – Classe Voisin

### 5.3.3 EnsembleDonnees.cs

Gère une collection d'échantillons avec des méthodes d'ajout et de consultation.

```

namespace ClassificationGrainsDeBle
{
    13 références
    internal class EnsembleDonnees
    {
        1 référence
        private List<Echantillon> echantillons = new List<Echantillon>();
        public void AjouterUnEchantillon(Echantillon e)
        {
            echantillons.Add(e);
        }

        0 références
        public void AjouterListEchantillon(List<Echantillon> echantillons)
        {
            foreach (Echantillon e in echantillons)
            {
                echantillons.Add(e);
            }
        }
        2 références
        public List<Echantillon> ObtenirEchantillon()
        {
            return echantillons;
        }
        4 références
        public int Taille()
        {
            return echantillons.Count;
        }
    }
}

```

FIGURE 11 – Classe EnsembleDonnees

## 5.4 Traitement des données CSV

La classe `Convert.cs` centralise toutes les opérations de chargement et conversion :

- `conversion_liste(string nom_fichier)` - Charge et convertit le CSV en liste de grains
- `ConstruireTableauDeGrain(List<Grain> grains)` - Affiche les grains en tableau
- `saveEchantillon(List<Grain> grains, EnsembleDonnees grainsTraining)` - Convertit les grains en échantillons
- `Afficher(List<Grain> grains)` - Affichage debug

```

7 références
internal class Convert
{
    2 références
    public static List<Grain> conversion_liste(string nom_fichier)
    {
        List<Grain> grains = new List<Grain>();
        string[] lignes = File.ReadAllLines(nom_fichier);

        string[] headers = lignes[0].Split(';');

        int idxVariety = Array.IndexOf(headers, "variety");
        int idxArea = Array.IndexOf(headers, "area");
        int idxPerimeter = Array.IndexOf(headers, "perimeter");
        int idxCompactness = Array.IndexOf(headers, "compactness");
        int idxKernelLength = Array.IndexOf(headers, "kernel_length");
        int idxKernelWidth = Array.IndexOf(headers, "kernel_width");
        int idxAsymmetry = Array.IndexOf(headers, "asymmetry_coefficient");
        int idxGroove = Array.IndexOf(headers, "groove_length");

        for (int i = 1; i < lignes.Length; i++)
        {
            string[] colonnes = lignes[i].Split(';');

            Grain g = new Grain(
                Enum.Parse<TypeDeGrain>(colonnes[idxVariety]),
                double.Parse(colonnes[idxArea], CultureInfo.InvariantCulture),
                double.Parse(colonnes[idxPerimeter], CultureInfo.InvariantCulture),
                double.Parse(colonnes[idxCompactness], CultureInfo.InvariantCulture),
                double.Parse(colonnes[idxKernelLength], CultureInfo.InvariantCulture),
                double.Parse(colonnes[idxKernelWidth], CultureInfo.InvariantCulture),
                double.Parse(colonnes[idxAsymmetry], CultureInfo.InvariantCulture),
                double.Parse(colonnes[idxGroove], CultureInfo.InvariantCulture));
            grains.Add(g);
        }
        return grains;
    }

    0 références
    public static void Afficher(List<Grain> grains)
    {
        foreach (Grain g in grains)
        {
            Console.WriteLine($"{g.TypeDeGrain} | {g.Area} | {g.Perimeter} | {g.Compactness} | " + $"{g.LongueurNoyau} | {g.LargeurNoyau} | {g.AsymetryCoefficient} | {g.GrooveLength}");
        }
    }
}

```

```

2 références
public static void ConstruireTableauDeGrain(List<Grain> grains)
{
    var table = new Table();
    table.AddColumn("type");
    table.AddColumn("area");
    table.AddColumn("perimeter");
    table.AddColumn("compactness");
    table.AddColumn("kernel_length");
    table.AddColumn("kernel_width");
    table.AddColumn("asymmetry");
    table.AddColumn("groove_length");

    foreach (var g in grains)
    {
        table.AddRow(
            g.TypeDeGrain.ToString(),
            g.Area.ToString(),
            g.Perimeter.ToString(),
            g.Compactness.ToString(),
            g.LongueurNoyau.ToString(),
            g.LargeurNoyau.ToString(),
            g.AsymetryCoefficient.ToString(),
            g.GrooveLength.ToString()
        );
    }
    AnsiConsole.Write(table);
}

3 références
public static void saveEchantillon(List<Grain> grains, EnsembleDonnees gainsTraining)
{
    List<Echantillon> echantillon = new List<Echantillon>();
    foreach (Grain g in grains)
    {
        double[] carac = new double[]
        {
            g.Area, g.Perimeter, g.Compactness,
            g.LongueurNoyau, g.LargeurNoyau,
            g.AsymetryCoefficient, g.GrooveLength
        };
        gainsTraining.AjouterUnEchantillon(new Echantillon(carac, g.TypeDeGrain));
    }
}

```

FIGURE 12 – Classe Convert

## 6 Tests et validation

### 6.1 Jeu de données

Les données proviennent d'un fichier CSV contenant les mesures de plusieurs grains des trois types.

### 6.2 Résultats de classification

Les tests effectués montrent une bonne précision de classification avec k=5 et la distance euclidienne.

Réel \ Prédit	Kama	Rosa	Canadian
Kama	19	2	3
Rosa	1	22	0
Canadian	0	0	24

Accuracy : 91.55 %  
Fichier JSON mis à jour ici : C:\Users\Fidele Atta\Desktop\UQAR HIVER 2026\P002\Travail pratique 1\TP1\_P002\_BON\TP\_P002\ClassificationGrainsDeBle\bin\Debug\net8.0\resultats.json  
Résultats sauvegardés dans resultats.json  
Fichier JSON créé ici : C:\Users\Fidele Atta\Desktop\UQAR HIVER 2026\P002\Travail pratique 1\TP1\_P002\_BON\TP\_P002\ClassificationGrainsDeBle\bin\Debug\net8.0\resultat.json  
Appuie sur une touche pour continuer...

FIGURE 13 – Sortie écran avec k = 5 et Distance Euclidienne

```

{
  "results.json": {
    "3": {
      "ParametresExecution": {
        "k": 4,
        "distance": {},
        "date": "2026-02-28 00:33:32"
      },
      "JeuxDeDonnees": {
        "taille_train": 139,
        "taille_test": 71
      },
      "Evaluation": {
        "accuracy": 0.9577464788732394,
        "matrice_confusion": [
          [0, 0, 0],
          [1, 0, 0],
          [2, 24, 0]
        ]
      }
    },
    "4": {
      "ParametresExecution": {
        "k": 5,
        "distance": {},
        "date": "2026-02-28 02:18:31"
      },
      "JeuxDeDonnees": {
        "taille_train": 139,
        "taille_test": 71
      },
      "Evaluation": {
        "accuracy": 0.9154929577464789,
        "matrice_confusion": [
          [19, 2, 3],
          [1, 22, 0],
          [0, 0, 24]
        ]
      }
    }
  }
}

```

FIGURE 14 – Le fichier resultats.json avec l'historique

Les résultats sont sauvegardé dans le fichier resultats.json avec l'historique.

## 7 Program.cs

### 7.1 Menu principal

C'est le menu de notre application. Il propose les options suivantes :

- **1 - Importer données**
- **2 - Choisir k**
- **3 - Choisir distance**
- **4 - Entraîner modèle**
- **5 - Tester modèle**
- **6 - Quitter**

**Importer données** Cette option importe le fichier CSV d'entraînement et génère les échantillons.

**Choisir distance** Cette option permet de sélectionner la distance à utiliser :

- distance euclidienne,
- distance de Manhattan.

**Entraîner modèle** Cette option appelle la méthode knn.Entraîner() pour entraîner le modèle.

**Tester modèle** Cette option :

- Charge le fichier test.csv,
- Prédit la classe de chaque grain,
- Remplit la matrice de confusion,
- Calcule l'exactitude,
- Affiche un tableau avec Spectre.Console,
- Sauvegarde les résultats dans un fichier JSON.

**ClasseToIndex()** La fonction ClasseToIndex() convertit les classes en indices 0, 1, 2 pour l'utilisation dans la matrice de confusion.

```

namespace ClassificationGrainsDeBlé
{
    // ...
    internal class Program
    {
        // ...
        static void Main(string[] args)
        {
            string fichierTrain = "seeds_dataset_training.csv";
            string fichierTest = "seeds_dataset_test.csv";

            EnsembleDonnees training = new EnsembleDonnees();
            EnsembleDonnees testing = new EnsembleDonnees();

            IDistance distanceChoisie = new DistanceEuclidienne();
            int k = 3;

            ClassifieurKnn knn = new ClassifieurKnn(k, distanceChoisie);
            bool continuer = true;

            while (continuer)
            {
                AnsiConsole.Clear();
                AnsiConsole.WriteLine("MENU PRINCIPAL [/]");

                string choix = AnsiConsole.Prompt(
                    new SelectionPrompt<string>()
                        .Title("Veuillez choisir les options dans l'ordre : [/]")
                        .AddChoices(new[]
                        {
                            "#1 - Importer données",
                            "#2 - Choisir k",
                            "#3 - Entrer distance",
                            "#4 - Entrainer modèle",
                            "#5 - Tester modèle",
                            "#6 - Quitter"
                        }));
            }

            switch (choix)
            {
                case "#1 - Importer données":
                    //Importation du Train
                    List<Grain> grainsTrain = Convert.conversion.Liste(fichierTrain);
                    Convert.saveEchantillon(grainsTrain, training);
                    AnsiConsole.Markup($"[green]Les données en grain du fichier train: [/]\n");
                    Convert.ConstireTableauDeGrain(grainsTrain);

                    //Importation du test
                    List<Grain> grainsTest = Convert.conversion.Liste(fichierTest);
                    AnsiConsole.Markup($"[green]Les données en grain du fichier test: [/]\n");
                    Convert.ConstireTableauDeGrain(grainsTest);
                    Convert.saveEchantillon(grainsTest, testing);
                    testing = new EnsembleDonnees();
                    Convert.saveEchantillon(grainsTest, testing);
                    AnsiConsole.Markup($"[green]Données importées [/]\n");
                    break;

                case "#2 - Choisir k":
                    k = AnsiConsole.Ask<int>($"[yellow]Entrez la valeur de k : [/]");
                    knn = new ClassifieurKnn(k, distanceChoisie);
                    AnsiConsole.Markup($"[green]k mis à jour : {k} [/]\n");
                    break;

                //Choix du méthode de calcul de distance
                case "#3 - Choisir distance":
                    string choixDistance = AnsiConsole.Prompt(
                        new SelectionPrompt<string>()
                            .Title("Veuillez choisir distance : [/]")
                            .AddChoices("Euclidienne", "Manhattan"));
                    break;
            }

            if (choixDistance == "Euclidienne")
                distanceChoisie = new DistanceEuclidienne();
            else
                distanceChoisie = new DistanceManhattan();

            knn = new ClassifieurKnn(k, distanceChoisie);
            AnsiConsole.Markup($"[green]Distance mise à jour [/]\n");
            break;
        }

        //entraîner le modèle
        case "#4 - Entrainer modèle":
            if (training.Taille() == 0)
            {
                AnsiConsole.Markup($"[red]Importer les données d'abord : [/]\n");
            }
            else
            {
                knn.Entraîner(training);
                AnsiConsole.Markup($"[green]Modèle entraîné [/]\n");
            }
            break;

        //tester
        case "#5 - Tester modèle":
            if (training.Taille() == 0)
            {
                AnsiConsole.Markup($"[red]Modèle non entraîné : [/]\n");
            }
            else
            {
                EvaluationPerformance eval =
                    new EvaluationPerformance();
                eval.Evaluer(k, distanceChoisie, training, testing);
                eval.AfficherTableSpectre();
                eval.SauvegarderJSONGlobal("resultats.json", k, distanceChoisie, training, testing);
            }
            break;

        case "#6 - Quitter":
            continuer = false;
            break;
        }

        if (continuer)
        {
            AnsiConsole.Markup(
                $"\n[blue]Résultats sauvegardés dans resultats.json [/]\n";
                Console.WriteLine($"Fichier JSON créé ici : {Path.GetFullPath("resultat.json")}\")");
            break;
        }
    }
}

```

FIGURE 15 – Programme principal

## 8 Conclusion

Ce projet nous a permis de :

- Maîtriser l'implémentation de l'algorithme KNN en C#.
- Appliquer les principes POO (encapsulation, héritage, polymorphisme).
- Utiliser des outils professionnels (Git, GitHub, Trello).
- Structurer un projet de A à Z.

## 9 Références

### 9.1 Documentation KNN

- Visual Studio Magazine – Implémentation du k-NN en C# (Partie 1) <https://visualstudiomagazine.com/articles/2024/10/01/implementing-k-nn-classification-us.aspx?Page=1>
- Visual Studio Magazine – Implémentation du k-NN en C# (Partie 2) <https://visualstudiomagazine.com/articles/2024/10/01/implementing-k-nn-classification-us.aspx?Page=2>
- MSDN Magazine – Comprendre la classification k-NN en C# <https://learn.microsoft.com/fr-fr/archive/msdn-magazine/2017/december/test-run-understanding-k-nn-in-csharp>
- Medium – Understanding K-Nearest Neighbors Algorithm (C# Example) <https://medium.com/@kdcodechronicles/understanding-k-nearest-neighbors-algorithm-knn-c-e>

### 9.2 Matrice de confusion

- Documentation ML.NET – ConfusionMatrix <https://learn.microsoft.com/en-us/dotnet/api/microsoft.ml.data.confusionmatrix?view=ml-dotnet-preview>

### 9.3 JSON : Sérialisation et écriture

- Microsoft – Travailler avec System.Text.Json <https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/how-to>
- Medium – Working with JSON in C# using System.Text.Json <https://ruper-anjaria.medium.com/working-with-json-in-c-using-system-text-json-9b61f95b551e>
- Newtonsoft.Json – Documentation officielle <https://www.newtonsoft.com/json>
- Newtonsoft.Json – Guide de sérialisation <https://www.newtonsoft.com/json/help/html/SerializingJSON.htm>

### 9.4 Spectre.Console

- Documentation officielle Spectre.Console <https://spectreconsole.net/console>
- Tutoriel – Création de tableaux avec Spectre.Console <https://www.luisllamas.es/en/csharp-spectre-console/>