|  |
| --- |
| Université du Québec à Rimouski |
| Travail Pratique I |
| INF11207 - Programmation orientée objet II  Professeur – Yacine Yaddaden, Ph. D. |

|  |
| --- |
| ⭢ Boutin, Frédérik  ⭢ Létourneau, Gabriel |

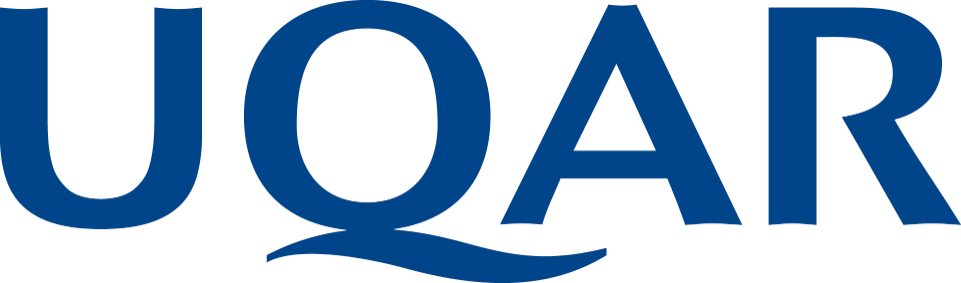


Table des matières

[Introduction 2](#_Toc96170807)

[La qualité du vin 2](#_Toc96170808)

[Algorithme du -NN et Résultats 2](#_Toc96170809)

[Utilisation du programme développé 2](#_Toc96170810)

[Problèmes et Difficultés rencontrées 2](#_Toc96170811)

[Conclusion 2](#_Toc96170812)

[Références 2](#_Toc96170813)

# Introduction

Le mandat de ce travail pratique 1 est la programmation d’un algorithme de mesure de la qualité des vins. Cet algorithme nous permettra de mettre en pratique les notions de programmation orienté objet, d’algorithmes de tri, d’interfaces et bien d’autres. Nous avons des fichiers ressources ainsi qu’un document de spécifications à notre disposition. Le document explicatif indique les notions nécessaires à la compréhension de l’algorithme. Les fichiers ressources se constituent, entre autres, de 30 fichiers csv *samples* qui représentent des vins individuels, le fichier *train.csv* qui contient les vins de références avec lesquels nous allons comparer les vins et le fichier *test.csv*. Ce dernier contient également plusieurs échantillons de vins à tester avec notre algorithme pour en voir l’efficacité.

# La qualité du vin

Il est possible de voir dans notre projet … que nous avons bien implémenté chaque classe interface du programme ressource, soit les classes IKNN et IWine. Nous avons pu respecter l’ordre d’appel des méthodes. Ainsi, les signatures des méthodes identifiées par *main methods* ont bien été appelé à partir du *main* et celles identifiées *utils* ont été appelées au sein de la classe de référence. Notre programme obtient des résultats de qualité de vins qui sont précis 64% du temps. Ce résultat est le même que celui obtenu dans le document explicatif. Ce faible résultat vient du fait que notre algorithme compare uniquement une distance entre le vin à tester et les autres échantillons pour lesquels nous avons déjà la qualité. De plus, en fonction de paramètres de départ, l’algorithme choisi la qualité équivalente du référentiel ayant la plus petite distance avec le vin test et le plus grand nombre de qualité. Ainsi, il résulte davantage de comparaison que d’une règle scientifique, d’où le faible résultat.

# Algorithme du -NN et Résultats

Le principe de notre k-nn est le suivant, nous traitons 6 cas de 3 compteurs de qualité (3, 6 et 9) avec des conditions. Dans l’ordre, le premier cas est lorsque nous avons le compteur 3 qui est supérieur à ces homologues 6 et 9, c’est un résultat qualité 3. Le deuxième et troisième cas reprennent le même principe pour les compteurs 9 et 6. Le quatrième cas indique que lorsque les compteurs 3 et 6 sont à égalité et dont on sait que le compteur 9 n’est pas celui ayant la plus grande valeur (par la position de la condition dans la série de *if*, *if else*), 3 est retourné comme résultat qualité. De même, le cinquième cas lorsque le compteur 3 et 9 sont à égalité, le retour est une qualité de 3. Finalement, dans le sixième cas avec les compteurs 6 et 9 à égalité, c’est le résultat de qualité 6 qui est retourné. Nous avons également testé d’autres types d’arrondi de qualité, mais ce n’est qu’avec le principe d’écrit ci-haut que nous avons obtenu une précision d’algorithme équivalente à celle du document explicatif.

# Utilisation du programme développé

L’utilisateur se verra d’abord offrir un choix d’entré de 6 lettres, soit **-e** pour spécifier le fichier csv contenant une liste de vins à faire évaluer par le programme, **-p** pour spécifier un fichier csv d’un seul vin à faire évaluer, **-t** pour spécifier la liste de vins de comparaison du programme qualifié par un expert, **-k** pour indiquer le nombre de proches parents à considérer dans le k-nn du programme(1 par défaut), **-s** pour choisir l’algorithme de tri qui sera utilisé (tri par sélection par défaut) et finalement **-h** pour faire afficher de l’aide au besoin.

Afin de mieux illustrer notre programme, deux scénarios serons illustrés. Le premier scénario sera pour un choix de (p), donc lorsque le programme devra comparer un seul échantillon de vin avec des paramètres par défaut (k=1, tri par sélection). Le deuxième scénario sera pour un choix de (e) avec une liste de vins csv, un (k) de 4 et pour un (s) avec le tri shell.

Scénario 1 : Une fois les paramètres sélectionnés, le programme va localiser le bon échantillon de vin, puis un objet vin1 va appeler la méthode *ImportOneSample* avec le fichier csv du vin à tester. Cette méthode va transposer les informations csv du fichier dans un objet vin. Puis, la méthode *Predict* sera appelée avec l’objet vin, elle va créer les listes de distances euclidiennes et de résultats qualité pour ensuite les utiliser dans l’appel de la méthode *EuclideanDistance* ainsi qu’avec le vin à tester. Le retour de la méthode *EuclideanDistance* sera une liste complète de chaque distance entre l’échantillon du vin par rapport à chaque vin de référence, ces deux listes seront envoyées, à leur tour, à la méthode *SelectionSort.* C’est cette méthode qui va trier la liste des distances euclidiennes de la plus petite à la plus grande et faire suivre les positions dans la liste des qualités (par rapport à la liste des distances). De retour à la méthode *Predict* avec les listes en ordre, la dernière méthode appelée sera *Vote* avec la liste des qualités et le k=1. Cette dernière va choisir le résultat de qualité en fonction de compteurs afin de choisir la qualité la plus présente. Puisque k=1, il n’y aura qu’un seul des 3 compteurs qui sera incrémenté et qui retournera le résultat. La méthode *Predict* peut maintenant retourner à son tour le résultat de qualité à l’utilisateur.

Scénario 2 : Afin d’éviter les répétitions, nous allons uniquement mentionner les particularités qui se distinguent par rapport au scénario 1 et non toutes les étapes du scénario 2. Une fois les paramètres entrés et le fichier csv localisé par le logiciel, les vins seront affichés dans la console avec la méthode *PrintInfo*, puis la méthode *Evaluate* sera appelée avec cette liste de vin. L’avantage de la méthode *Evaluate* vient du fait qu’elle va s’adapter à la particularité que nous comparons une liste de vins plutôt qu’un seul échantillon. En effet, des méthodes équivalentes au scénario 1 seront appelés, mais qui travaillent sur des listes. Ainsi, *Evaluate* va appeler *ImportAllSample* qui va retourner une liste d’objets Wine. *Evaluate* fera ensuite appel à *Predict*, pour comparer chaque vin de la liste avec ceux de la liste des comparatifs et les étapes du scénario 1 pourront se répéter. Cependant, dans notre cas, avec le tri shell et pour 4 proches voisins. La liste de résultats qualités sera ensuite comparée en appelant la méthode *ConfusionMatrix* qui va comparer les résultats qualités de l’algorithme avec ceux d’un expert et afficher la matrice ainsi que le pourcentage d’efficacité du logiciel.

# Problèmes et Difficultés rencontrées

# Conclusion

En conclusion, malgré les … difficultés rencontrées au cours de ce travail pratique, nous avons été en mesure de programmer un algorithme répondant aux conditions du document explicatif et du fichier *visual studio* fournies. En effet, nous respectons les conditions d’appel des différentes méthodes pour les classes INN et IWine, tant par les paramètres d’entrées/sorties que par celui qui appel chaque méthode. De plus, le résultat de précision d’algorithme ainsi que la matrice de confusion nous auront permis de confirmer que notre algorithme arrive aux mêmes résultats que ceux présentés dans le document explicatif.

# Références