



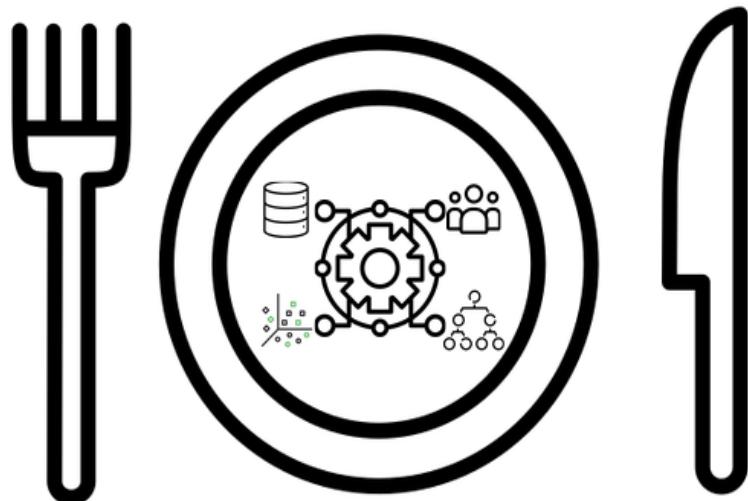
Master 1 Informatique et Ingénierie des Systèmes Complexes (IISC)

CY Cergy Paris Université

Projet de synthèse

Big Cooking Data

Rédigé par : Arthur MIMOUNI



Equipes :

Arthur MIMOUNI
Mamadou Bella DIALLO
Marouane RACHIDY
Imane CHBIRA
Matthieu SAUVAGEOT

Rapporteuse :
Mme. TZOMPANAKI Katerina

Tuteur technique :
Mr. VODISLAV Dan
Encadrant de gestion de projet :
Mr. LIU Tianxiao

Rendu le
10 juin 2022

Table des matières

1	Introduction	8
1.1	Contexte du projet	8
1.2	Mise en scénario	8
1.3	Objectif du projet	9
1.4	Organisation du rapport	10
2	Présentation et spécification du projet	11
2.1	Fonctionnalités attendues	11
2.2	Conception globale du projet	12
2.2.1	Vue pour l'utilisateur	12
2.2.2	Architecture technique	12
2.3	Problématiques identifiées et solutions envisagées	12
2.4	Environnement de travail	13
3	Collecte, nettoyage et insertion des données	15
3.1	Analyse de la problématique	15
3.2	Etat de l'art : études des solutions existantes	16
3.2.1	Différences entre Data Crawling et Data Scraping	16
3.2.2	Création d'un jeu de données avec Scrapy	17
3.3	Solution proposée et sa mise en œuvre	18
3.3.1	Architecture de la solution	18
3.3.2	Extraction des données	19
3.3.3	Nettoyage, formatage et insertion des données	20
3.4	Tests et certifications de la solution	22
4	Partitionnement des données	24
4.1	Analyse de la problématique	24
4.2	Etat de l'art : études des solutions existantes	25
4.2.1	K-Means Clustering	25
4.2.2	Principal Component Analysis (PCA)	26
4.3	Solution proposée et sa mise en œuvre	27
4.3.1	Transformation des recettes en vecteurs binaires	27
4.3.2	Réduction de dimension de nos vecteurs	28
4.3.3	Recherche du nombre optimal de clusters	29
4.4	Tests et certifications de la solution	30
5	Classification des données	32
5.1	Analyse de la problématique	32
5.2	Etat de l'art : études des solutions existantes	32
5.2.1	Arbre de décision	32
5.2.2	Classification naïve Bayésienne	34
5.2.3	Validation croisée K-Fold	35
5.3	Solution proposée et sa mise en œuvre	36
5.3.1	Choix de l'algorithme d'apprentissage	36

5.3.2	Création des règles de décision	36
5.3.3	Évaluation de notre arbre de décision	37
5.3.4	Désavantages de l'arbre	39
5.3.5	Comparaison de performance entre l'arbre de décision et la classification naïve Bayésienne	39
5.4	Tests et certifications de la solution	40
5.5	Ajout de nouvelles recettes dans la base de données	42
5.5.1	Prétraitement textuel des caractéristiques de la recette	42
5.5.2	Recherche du cluster et des recettes voisines pour la nouvelle recette	43
6	Algorithme de recommandation	45
6.1	Analyse de la problématique	45
6.2	Etat de l'art : études des solutions existantes	46
6.2.1	Filtrage basé sur le contenu	46
6.2.2	Filtrage collaboratif	47
6.3	Solution proposée et sa mise en œuvre	48
6.3.1	Calcul des recettes proches	48
6.3.2	Diagramme de classes de l'algorithme de recommandation	48
6.3.3	Post-traitement textuel des ingrédients	49
6.3.4	Récupération des données de l'utilisateur	50
6.3.5	Calcul vectoriel et matriciel	52
6.4	Tests et certifications de la solution	52
6.5	Solution alternative implémentée : Algorithme de filtrage collaboratif User-User	54
6.5.1	Création des clients	55
6.5.2	Création de la matrice d'utilité	55
6.5.3	Calcul des prédictions des notes	57
6.5.4	Tests et certifications	58
7	Rendu final	61
7.1	Interface utilisateur finale	61
7.2	Tests utilisateur et certification	65
7.2.1	Recherche de recettes par ingrédients	65
7.2.2	Ajout de recette	66
7.2.3	Suggestion de recettes	68
8	Gestion de projet	72
8.1	Méthode de gestion	72
8.1.1	Phase d'étude	72
8.1.2	Phase de planification	72
8.1.3	Phase d'exécution	74
8.1.4	Phase de clôture	74
8.2	Répartition de tâches	75
8.2.1	Répartition de tâches prévisionnelles	75
8.2.2	Répartition de tâches réalisé	76
8.3	Utilisation des outils de gestion de projet au sein de l'équipe	77
8.3.1	Trello	77
8.3.2	YouTrack	77
8.3.3	Discord	78
8.3.4	GitHub	78
8.4	Gestion des réunions et des conflits de l'équipe	79

9 Conclusion et perspectives	80
9.1 Conclusion	80
9.2 Perspectives	80
9.2.1 Planificateur de repas	80
9.2.2 Partitionnement des données avec les quantités des ingrédients	81
9.2.3 Inférence en temps réel de l'arbre de décision	82

Table des figures

1.1	Mise en scénario du projet	9
2.1	Diagramme de cas d'utilisation	11
2.2	Composants vus par l'utilisateur	12
2.3	Architecture globale	13
3.1	Exemple d'une recette du site de cuisine Marmiton	16
3.2	Nom des champs et description des contenus d'une recette structurée	16
3.3	Architecture du Web Crawling	17
3.4	Architecture du Web Scraping	17
3.5	Architecture de Scrapy	18
3.6	Architecture hiéarchique des traitements de nos données	19
3.7	Exemple de la structure hiérarchique des catégories pour une recette	19
3.8	Règle d'extraction des boutons de pagination	20
3.9	Règle d'extraction des liens URLs	20
3.10	Schéma Merise de notre base de données	21
3.11	Schéma logique de notre base de données	22
3.12	Extrait du fichier JSON pour les recettes collectées	23
3.13	Exemple de recette dans la base de données locale	23
4.1	Relation des recettes selon leurs ingrédients	24
4.2	Calcul des centroides	25
4.3	Calcul des distances entre les centroides et les points	25
4.4	Algorithme du K-Means Clustering	26
4.5	Exemple de fonctionnement du K-Means Clustering	26
4.6	Projection d'un cube 3D sur un espace 2D	27
4.7	Vectorisation des recettes	28
4.8	Réduction de dimensions via PCA	29
4.9	Recherche du nombre de clusters optimale à l'aide de la méthode Elbow	30
4.10	Calcul de la silhouette de notre partitionnement de données	31
4.11	Plan 2D et 3D de notre partitionnement de données	31
5.1	Fonctionnement de l'arbre de dimension 2D	33
5.2	Pureté d'un arbre de décision	34
5.3	34
5.4	35
5.5	35
5.6	35
5.7	Architecture de la validation croisée K-Fold	36
5.8	Algorithme de la création des règles de décision	37
5.9	Schéma de la construction des règles de décision	37
5.10	Algorithme de l'évaluation de l'arbre de décision	38
5.11	Précision d'un arbre avec différentes profondeurs	39
5.12	Comparaison de performance des classificateurs	40
5.13	Exemple d'un arbre de décision avec une profondeur de 3	40

5.14	Extrait de l'arbre de décision avec une profondeur de 12	41
5.15	Nombre de recettes par cluster selon les ingrédients "crème fraîche" , "oignons" et "sole"	41
5.16	Résultat de la prédiction	41
5.17	Temps d'exécution des appels à l'arbre de décision	42
5.18	Recherche des ingrédients similaires - 1	42
5.19	Recherche des ingrédients similaires - 2	43
5.20	Recherche du meilleur cluster	44
6.1	Architecture globale de l'algorithme de recommandation	45
6.2	Exemple de cas d'utilisation du filtrage basé sur le contenu	46
6.3	Exemple de cas d'utilisation du filtrage collaboratif	47
6.4	Diagramme des classes de l'algorithme de recommandation	49
6.5	Post-traitement des ingrédients et appel de l'arbre de décision	50
6.6	Récupération des données nécessaires à l'algorithme de recommandation	50
6.7	Création de la matrice des recettes semi-pertinentes	51
6.8	Création du vecteur de l'utilisateur	51
6.9	Formule du cosinus similarité	52
6.10	Calcul des recettes pertinentes	52
6.11	Extrait des recettes à suggérer pour un utilisateur	53
6.12	Récupération des ingrédients de préférence de l'utilisateur	54
6.13	Comparaison des algorithmes de recommandation	54
6.14	Création de la matrice d'utilité	55
6.15	Cosinus similarité	55
6.16	Calcul du cosinus similarité entre (U1,U2) et (U1,U3)	56
6.17	Création de la matrice d'utilité centrée sur la moyenne	56
6.18	Calcul du cosinus similarité centré sur la moyenne entre (U1,U2) et (U1,U3)	57
6.19	57
6.20	Matrice d'utilité	57
6.21	Résultat de la prédiction	58
6.22	Extrait des recettes à suggérer pour un utilisateur	58
6.23	Temps d'exécution pour l'algorithme basé sur le contenu	59
6.24	Temps d'exécution pour l'algorithme de filtrage collaboratif	60
6.25	Comparaison des matrices d'utilité	60
7.1	Page d'accueil de notre application web	62
7.2	Page de la recette	63
7.3	Page du profil de l'utilisateur	64
7.4	Page de recherche	65
7.5	Recherche de recettes par ingrédients	66
7.6	Ajout des caractéristiques de la nouvelle recette	67
7.7	Visualisation de la nouvelle recette	68
7.8	Création du compte utilisateur	69
7.9	Suggestion de recettes basées sur les ingrédients de préférence	69
7.10	1ère recette évaluée par l'utilisateur	70
7.11	2ème recette évaluée par l'utilisateur	70
7.12	Pop-in pour évaluer une recette	71
7.13	Affichage des recettes à suggérer	71
8.1	Répartition des rôles de l'équipe	72
8.2	Planification prévisionnelle	74
8.3	Planification réalisée	75
8.4	Répartition des tâches prévisionnelles	76
8.5	Répartition des tâches réalisée	76
8.6	Exemple de notre tableau Trello avant la phase d'exécution	77

8.7	Exemple de notre dashboard YouTrack	77
8.8	Exemple d'un ticket YouTrack	78
8.9	Exemple de l'état de notre répertoire GitHub	79
9.1	Estimation du nombre de calories	81
9.2	Partitionnement des données avec les quantités des ingrédients	82

Remerciements

Les auteurs du projet voudraient remercier Monsieur Dan VODISLAV de la confiance qu'il nous a accordé tout au long de l'élaboration du projet ainsi qu'aux différents conseils qu'il nous a apporté pour améliorer la conception de notre projet. Nous remercions de même Monsieur TIANXIAO Liu pour ses conseils apportés au niveau de la structure de notre rapport de projet.

Chapitre 1

Introduction

1.1 Contexte du projet

Avec la popularité croissante des applications web au cours des dernières années, Internet est devenu une source accessible pour de grandes quantités d'informations et a ouvert les portes à un partage collaboratif à grande échelle notamment en ce qui concerne les recettes de cuisine. Ces recettes sont proposées aux utilisateurs à l'aide d'algorithmes de recommandations de contenu qui sont devenus indispensables sur la plupart des applications web. Leur but est de proposer une meilleure approche pour organiser le contenu de façon pertinente aussi bien pour l'utilisateur que pour la plateforme.

Ces recommandations personnalisées sont possibles seulement par une analyse pointue de données massives qui font l'objet d'un enjeu de filtrage, de tri et de prédiction à l'aide d'algorithmes et de modèles de Machine Learning. Le machine learning est l'un des sous-domaines les plus populaires de l'intelligence artificielle et donne un moyen aux machines d'apprendre et de s'améliorer automatiquement à partir de l'expérience.

Le succès d'une application web se mesure par la satisfaction du client et à la qualité des suggestions. Cela implique par conséquent la maîtrise des outils de machine learning visant à structurer les données utilisées par les algorithmes de recommandations.

1.2 Mise en scénario

La mise en scénario que nous avons imaginé se déroule en plusieurs étapes. Pour ce faire, l'utilisateur aura accès à une interface graphique depuis laquelle il pourra sélectionner les différentes fonctionnalités de notre application.

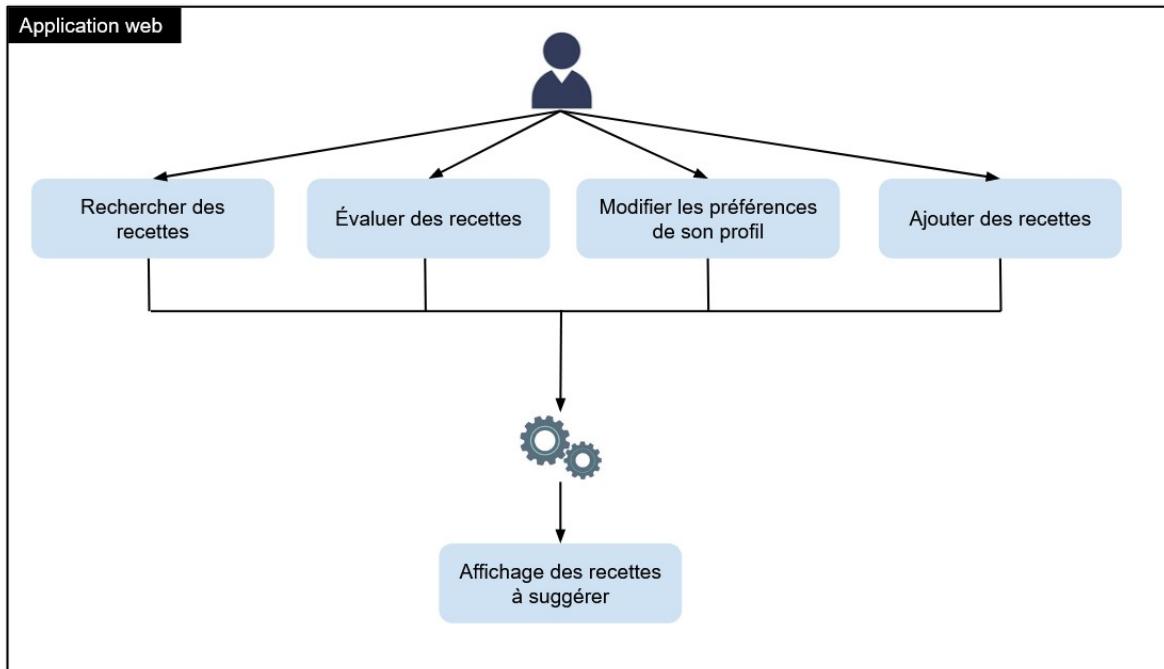


FIGURE 1.1 – Mise en scénario du projet

Tout d'abord, l'utilisateur à la possibilité d'évaluer des recettes et de choisir des ingrédients de préférences sur son profil. Ces deux fonctionnalités sont indispensables car elles permettront de générer des recommandations selon les données de l'utilisateur. Lorsque l'utilisateur sera sur l'accueil de l'application web, nous exécuterons un processus algorithmique visant à suggérer les recettes les plus pertinentes. Ces recettes seront affichées et pourront être visualisées en détail en cliquant dessus. L'utilisateur peut aussi effectuer une recherche de recettes basée sur des ingrédients ou sur des mots-clefs. De plus, l'utilisateur à la possibilité d'ajouter des nouvelles recettes dans la base de données.

1.3 Objectif du projet

L'objectif de ce projet est de créer une application web proposant une multitude de recettes aux utilisateurs. L'intérêt principal de l'application est de pouvoir suggérer automatiquement des recettes selon les préférences des utilisateurs.

La première étape que nous devons franchir est évidemment d'obtenir les données nécessaires au fonctionnement de cet algorithme de suggestion. Ce jeu de données devra être collecté, nettoyé et structuré de manière uniforme afin qu'une analyse puisse être effectuée. Cette collecte de recette sera faite sur le site web **Marmiton**.

Pour pouvoir regrouper les recettes similaires entre elles, nous utiliserons une méthode de partitionnement de données afin de catégoriser les recettes en rapprochant celles dont les ingrédients sont les plus similaires. Suite à cela, nous effectuerons une classification de nos recettes afin de réduire le temps d'exécution de certaines requêtes dans la base de données telle que la recherche de recette par ingrédient. Cette classification nous permettra aussi d'ajouter de nouvelles recettes dans un groupe existant.

Pour l'algorithme de recommandation, nous devrons utiliser les résultats de nos modèles de machine learning précédemment décrit afin de récupérer les données nécessaires à son fonctionnement. Ensuite, nous devrons effectuer les calculs vectoriels / matriciels permettant de trouver les recettes à suggérer.

1.4 Organisation du rapport

Nous nous consacrerons tout d'abord aux spécifications du projet, c'est-à-dire les fonctionnalités attendues ainsi qu'une description de la conception globale du projet. Cette dernière rendra compte des différentes vues (utilisateur et technique) du projet et pourra mettre en évidence les différentes problématiques qui seront traitées au cours des chapitres techniques. Enfin, une dernière section concernera l'environnement de travail au cours de laquelle sera dressé une liste exhaustive du matériel ainsi que des logiciels et outils utilisés.

Les problématiques soulevées précédemment dans le cahier des charges seront respectivement traitées dans les sections techniques. Ces sections, qui constituent les chapitres techniques auront pour sujet principaux : la collecte des recettes sur le site web **Marmiton**; l'apprentissage non supervisé (K-mean Clustering) et supervisé (Arbre de décision) de notre jeu de données et enfin l'élaboration de notre algorithme de recommandation.

Un chapitre concernant le rendu final figurera également sur ce rapport. Il aura pour finalité de présenter le résultat attendu du projet ainsi que des différents outils que l'utilisateur sera amené à manipuler pour faire fonctionner l'application web. Plusieurs tests seront mis en place pour témoigner du bon fonctionnement des différentes fonctionnalités.

Enfin, un chapitre sera consacré à la gestion du projet dans laquelle figureront les méthodes de travail employées ainsi que la répartition des tâches au cours du projet. Nous conclurons ce rapport par une explication du déroulement du projet, du travail réalisé et des améliorations possibles à effectuer.

Chapitre 2

Présentation et spécification du projet

Ce chapitre présente le cahier des charges du projet et les spécifications techniques, il contient les principaux éléments nécessaires pour comprendre la conception technique et l'architecture du projet.

2.1 Fonctionnalités attendues

Notre application web permet à l'utilisateur d'effectuer des recherches de recettes de deux manières différentes :

- Recherche par mots-clefs.
- Recherche par ingrédients (avec / sans ingrédients).

L'utilisateur peut aussi évaluer et commenter une recette. La note donnée à une recette est comprise entre 1 et 5 étoiles et le commentaire d'une recette ne peut être fait seulement si l'utilisateur a évalué la recette. Les recettes pourront être affichées en détail afin de visualiser les caractéristiques de celles-ci (temps de cuisson, étapes de préparations etc...). De plus, l'utilisateur peut ajouter une recette dans la base de données en détaillant les caractéristiques de celle-ci.

Lorsque l'utilisateur est sur l'accueil, le système générera un algorithme de recommandation permettant de proposer à l'utilisateur des recettes selon ce qu'il a visualisé et évalué.

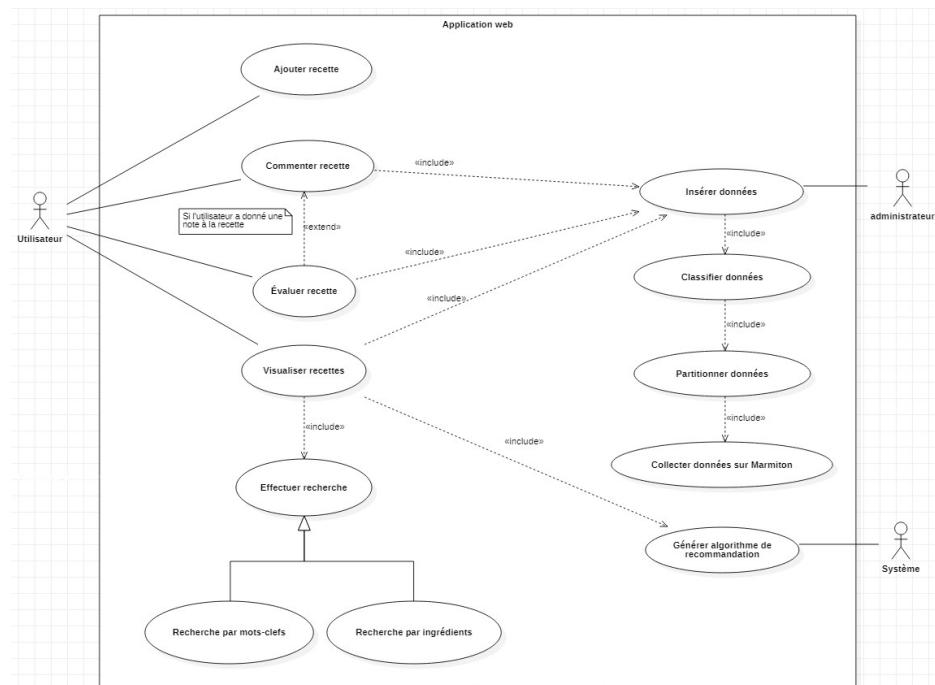


FIGURE 2.1 – Diagramme de cas d'utilisation

2.2 Conception globale du projet

2.2.1 Vue pour l'utilisateur

Notre application intelligente étant une application web, les composants vus par l'utilisateur correspondent à nos différentes pages web. Lorsque l'utilisateur arrive sur l'application, il est redirigé vers la page d'accueil dans laquelle seront affichées les recettes à suggérer. Sur cette page, l'utilisateur peut choisir d'être redirigé vers d'autres pages :

- la page de recherche où l'utilisateur pourra effectuer des recherches précises de recettes selon ses goûts et ses envies
- la page de détail dans laquelle l'utilisateur pourra visualiser une recette en détail et effectuer une évaluation de celle-ci.
- la page de profil qui sera accessible après s'être inscrit sur le site web et qui contiendra les données personnelles de l'utilisateur.

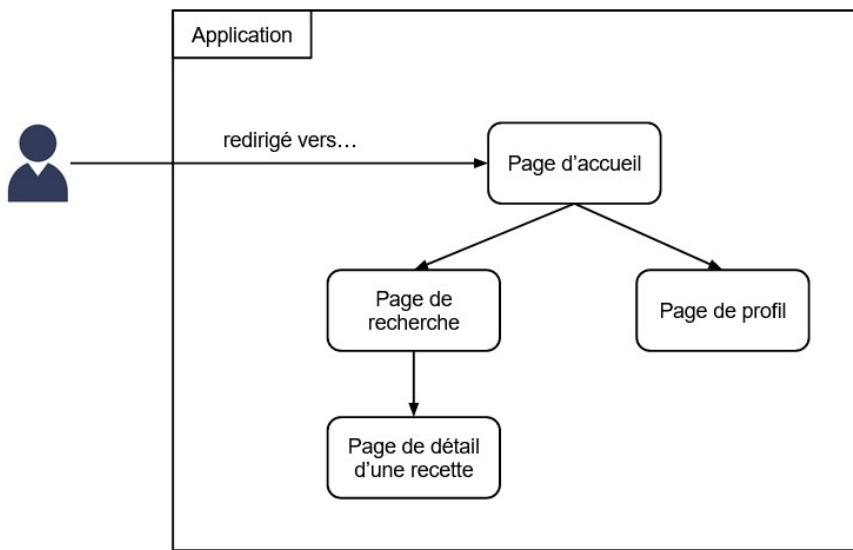


FIGURE 2.2 – Composants vus par l'utilisateur

2.2.2 Architecture technique

L'architecture globale de notre projet est une architecture hiérarchique. Premièrement, nous effectuons le processus de data scraping en collectant les recettes sur le site web **Marmiton**. Le nettoyage et l'organisation des données vont nous permettre de procéder au partitionnement de celles-ci grâce à un K-Means Clustering qui va regrouper les recettes dans différents clusters. Ce partitionnement va ainsi permettre la création et l'entraînement de notre arbre de décision pour classifier nos données. Enfin, notre algorithme de recommandation utilisera nos données clustérées et classifiées pour son processus de filtrage de recettes.

2.3 Problématiques identifiées et solutions envisagées

Plusieurs problèmes majeurs se posent d'un point de vue technique du projet. Dans cette section, nous décrirons ces problèmes et les solutions envisagées pour celles-ci.

La récupération des recettes sur le site web **Marmiton** est l'une des étapes les plus importantes dans l'élaboration de notre projet puisque ce sont ces données qui seront utilisées dans l'ensemble des processus suivants. La structure HTML des pages du site web doit donc être étudiée afin de connaître la position de l'ensemble des éléments à récupérer. Ces éléments devront être nettoyés avec précaution pour éviter la récupération de données faussées ou corrompues. Nous utiliserons le framework **Scrapy**

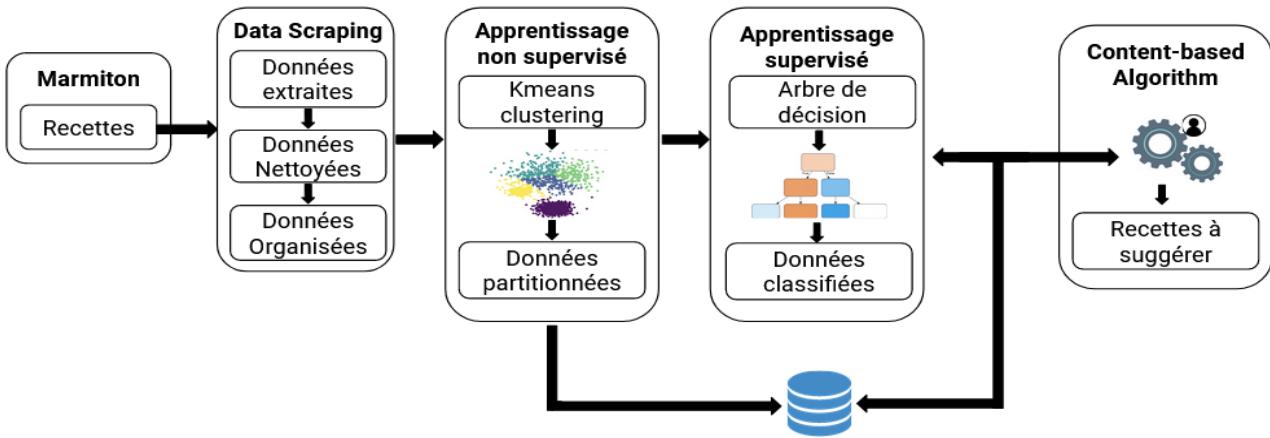


FIGURE 2.3 – Architecture globale

qui facilitera grandement l'extraction des recettes du code HTML.

Pour pouvoir effectuer le partitionnement de nos données, il nous faut effectuer des calculs de distance euclidienne entre nos recettes afin de regrouper celles-ci dans des clusters. Par conséquent, la principale problématique de cette étape est de trouver un moyen de convertir nos recettes en points dans un plan à K dimensions. Pour ce faire, nous utiliserons l'algorithme **K-Means Clustering** ainsi qu'un schéma de réduction de dimensionnalité (**PCA**) afin de réduire drastiquement la dimension de nos vecteurs.

Concernant la classification de nos données, nous utiliserons un **Arbre de décision**. Nous devrons générer nos règles de décisions permettant l'entraînement de l'arbre. Plusieurs évaluations vont être nécessaires pour trouver la profondeur de l'arbre pour laquelle la précision est la plus élevée. La construction de ces analyses se fera grâce à une **validation croisée K-Fold** qui aide à l'évaluation des modèles d'apprentissage automatique. De plus, un service web devra être mis en place pour pouvoir utiliser l'arbre de décision dans notre application web.

Enfin, deux problématiques peuvent être abordées en ce qui concerne l'algorithme de recommandation. La première est la récupération des recettes semi-pertinentes qui sont les recettes proches de celles des données de l'utilisateur, et la deuxième est le calcul des recettes pertinentes .Pour résoudre ces problématiques, nous utiliserons les coordonnées des recettes (récupérés grâce au K-Means Clustering) pour stocker dans la base de données l'ensemble des recettes proches pour chacune d'elles. Pour la recherche des recettes pertinentes, nous utiliserons le **cosinus similarité** qui permet de calculer l'angle entre deux vecteurs et qui représentera notre score de similarité.

2.4 Environnement de travail

Plusieurs outils sont utilisés pour développer les différents composants de notre projet. Le code de l'application web est implémenté avec l'IDE **PHPStorm** qui facilite grandement la programmation en permettant l'auto-complétion du code, la recherche rapide de classes ou encore la génération automatique de la documentation.

Pour la création de notre application web, nous utilisons la plupart des technologies web majeurs sur le marché :

- **HTML, CSS, Javascript et Bootstrap** pour la structure des différentes pages web.
- **PHP Orienté Objet** pour le back-end et la création de l'algorithme de recommandation.

Pour notre collecte de données, notre apprentissage supervisé (K-Means Clustering) et non supervisé (Arbre de décision), nous utilisons le langage **Python** avec les IDEs **PyCharm** et **VisualStudio**. Python est un très bon langage lorsque l'on veut effectuer du machine learning et de la collecte de données, c'est donc un outil indispensable pour notre projet. Plusieurs librairies externes python sont utilisées :

- **Scrapy** pour le data scraping et data crawling.
- **Matplotlib** pour la visualisation de nos données sous forme graphique.
- **Pandas** et **Numpy** pour le traitement des données
- **Sklearn** pour l'élaboration du K-Means Clustering et de l'arbre de décision.
- **Ladon** pour la création du web service SOAP.

En ce qui concerne la plateforme de développement web et le système de gestion de base de données, nous avons choisi respectivement **WAMP** et **phpMyAdmin** (mySQL).

Chapitre 3

Collecte, nettoyage et insertion des données

Dans ce chapitre, nous parlerons des différents aspects liés à la récupération et au traitement des données. Après une présentation des problématiques, nous effectuerons un état de l'art des solutions existantes puis nous détaillerons les solutions mises en place. Nous compléterons ce chapitre par une série de tests pour certifier nos solutions.

3.1 Analyse de la problématique

Étant donné que la plupart des sites web de cuisine affichent leurs recettes sous forme de texte non structuré, il est primordial d'analyser les pages web afin d'extraire toute information possible qui pourrait ensuite être nécessaire pour notre analyse. Des exemples simples d'informations utiles sont les ingrédients, les préparations ou encore les noms et images des recettes. Puisque nous ne connaissons pas à l'avance toutes les informations que nous aurions besoin pour chaque recette, nous devons d'abord télécharger l'intégralité du code HTML de chaque page web pour ensuite analyser le code et extraire les informations recherchées. Dans ce processus, la structure de chaque site web doit être prise en considération et l'arborescence DOM de chaque page web doit être parcourue en conséquence.

Le Document Object Model ou DOM (pour modèle objet de document) est une interface de programmation pour les documents HTML, XML et SVG. Il fournit une représentation structurée du document sous forme d'un arbre et définit la façon dont la structure peut être manipulée par les programmes, en termes de style et de contenu. En d'autres termes, le DOM permet de fournir en détails la structure d'une page web et est accessible très facilement dans la plupart des moteurs de navigation (Mozilla firefox, Google Chrome, Opera etc...).

La figure 3.1 montre un exemple de l'apparence d'une recette sur le site web **Marmiton**, tandis que la figure 3.2 montre le contenu d'une recette structurée.

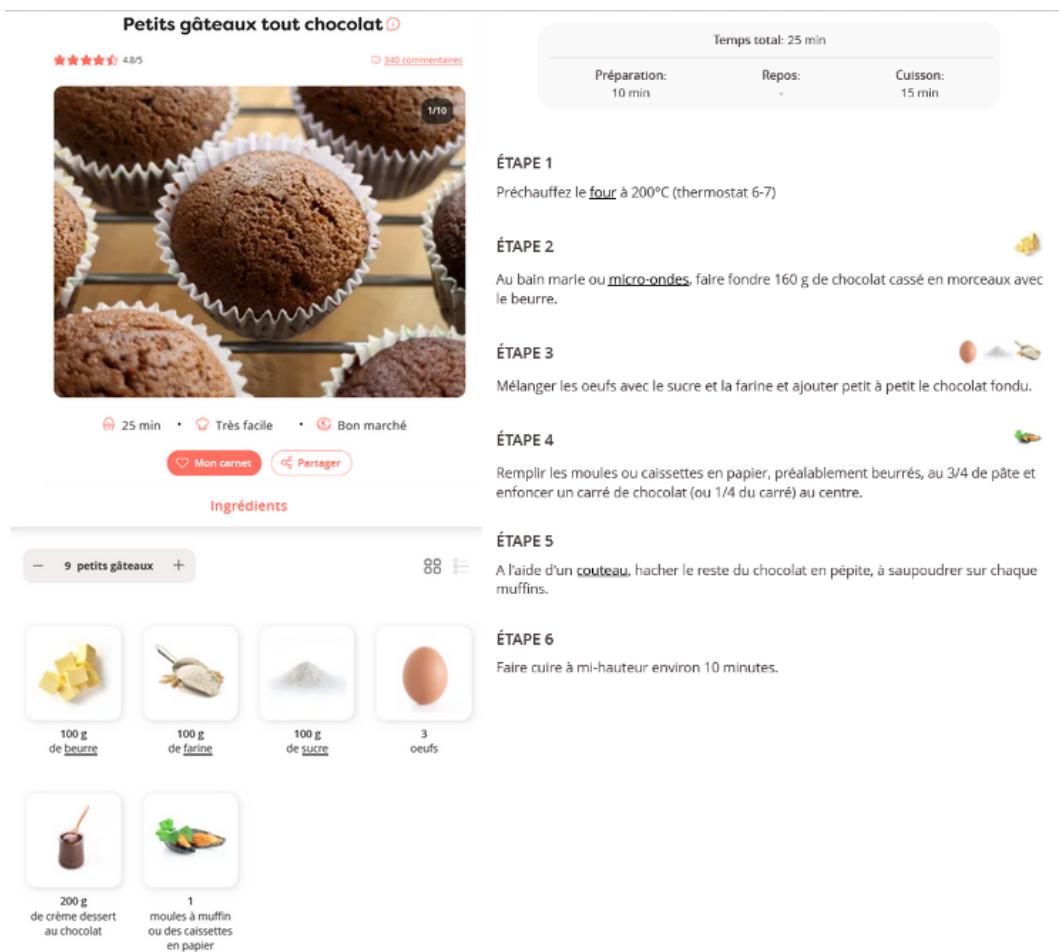


FIGURE 3.1 – Exemple d'une recette du site de cuisine Marmiton

Champ	Contenu
Nom	Nom de la recette
Ingrédients	Liste des ingrédients avec leur quantité
Préparation	Temps de préparation
Repos	Temps de cuisson
Cuisson	Liste des étapes
Étapes	Image URL de la recette
Catégorie	Catégorie de la recette
Difficulté	Difficulté de la recette
Budget	Budget de la recette

FIGURE 3.2 – Nom des champs et description des contenus d'une recette structurée

3.2 Etat de l'art : études des solutions existantes

3.2.1 Différences entre Data Crawling et Data Scraping

Il existe de nombreuses façons de recueillir des informations sur le web mais le Data Scraping et le Data Crawling sont deux des méthodes les plus populaires. Même si la plupart des personnes utilisent ces termes de manière interchangeable, ils n'ont pas la même définition et ont des objectifs différents.

Un robot d'indexation, également connu sous le nom de Web Crawler, passe par diverses cibles et recherche deux types d'éléments : les données et davantage de cibles à explorer. Tout d'abord, il visite une liste spécifique d'URLs (aussi appelé "graines"). Lors de la visite, le robot localise le contenu des pages et index celui-ci. Après avoir fait l'indexation, il identifie d'autres liens trouvés sur les pages web initiales et les ajoute à la frontière. Puis, le robot réitère les mêmes étapes avec de nouveaux liens jusqu'à ce que la frontière soit vide. En d'autres termes, c'est ce que les moteurs de recherche tels que Bing, Google ou Yahoo effectuent ; ils utilisent des robots pour parcourir les sites web, découvrir les données qu'ils incluent et ainsi créer des indexés.

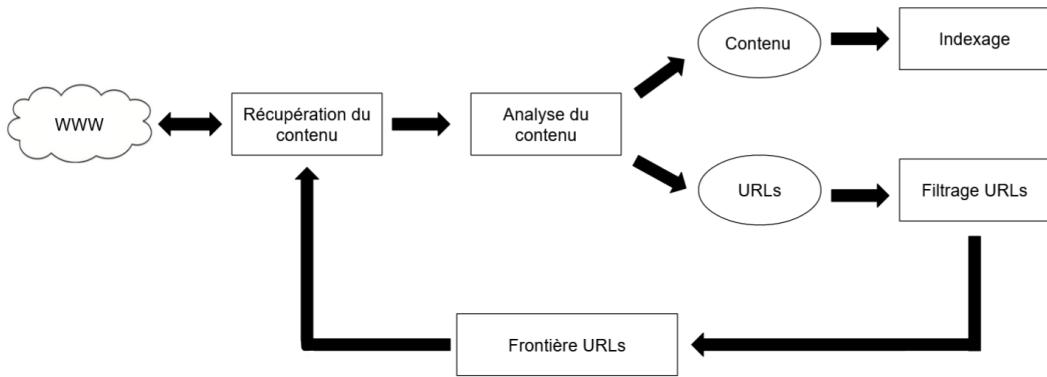


FIGURE 3.3 – Architecture du Web Crawling

Le scraping est une technologie qui permet d'extraire de manière automatisée des données et informations présentent sur un site web et de les transformer en d'autres formats plus exploitables (CSV, JSON ou XML). Un Web Scraping utilise une liste d'URLs, charge tout le code HTML et rassemble toutes les données (ou des données prédéfinies). Enfin, il télécharge et enregistre dans un format spécifique ces données. Ce processus peut être comparé à un copier-coller automatique.

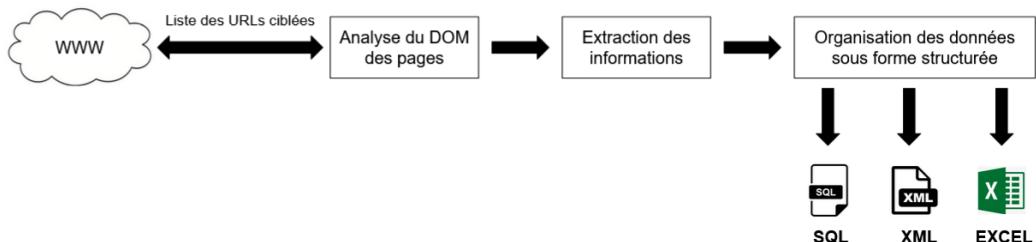


FIGURE 3.4 – Architecture du Web Scraping

3.2.2 Crédit d'un jeu de données avec Scrapy

Pour pouvoir effectuer la collecte de recettes, nous utilisons le framework Open Source **Scrapy** qui permet à la fois la création de robots d'indexation afin d'explorer les données mais aussi le téléchargement de celles-ci à l'aide d'ensemble de classes qui se nomment **Spiders**. Développé en Python, il dispose d'une forte communauté et offre de nombreux modules supplémentaires permettant de grandement faciliter la collecte de données. Ci-dessous, nous allons expliquer l'architecture et le fonctionnement global du framework. Nous affichons ci-dessous, l'architecture proposée par **Scrapy** (voir la source [Dev22] dans la bibliographie).

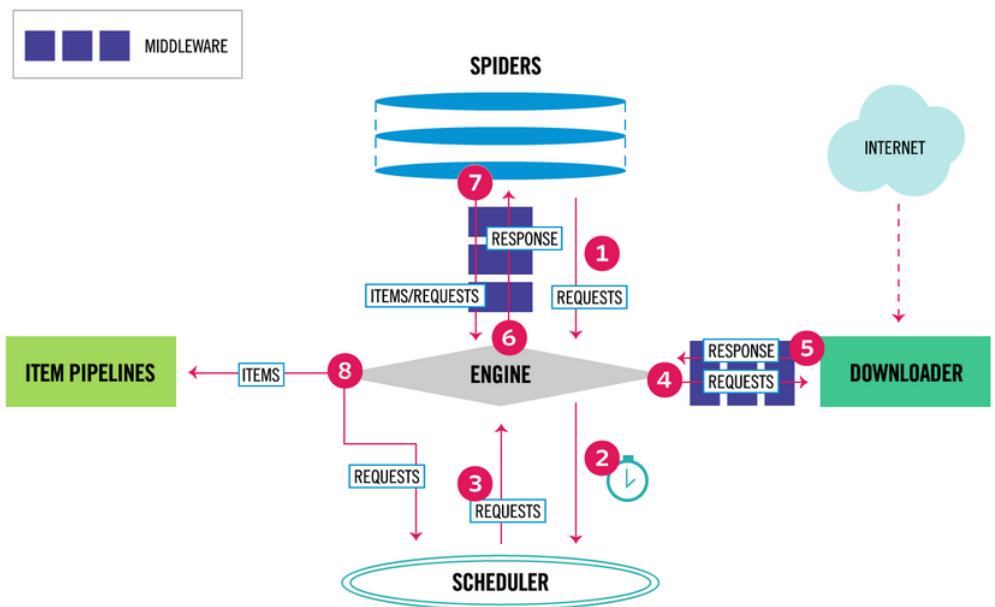


FIGURE 3.5 – Architecture de Scrapy

Avant de décrire les différentes étapes qu'effectue le framework Scrapy, il est important de donner une courte définition des composants de cette architecture :

- **Engine** : Il s'agit du moteur de **scrapy** qui va essentiellement faire l'orchestration de toutes les demandes et réponses, et les transmettre à différentes parties du framework.
- **Scheduler** : Le planificateur reçoit les demandes du moteur et les met en file d'attente pour les renvoyer plus tard au moteur lorsque celui-ci les demande.
- **Downloader** : Comme son nom l'indique, le téléchargeur récupère l'URL d'une demande, télécharge la page web et la transmet au moteur.
- **Spiders** : Les Spiders sont des classes personnalisées écrites par les utilisateurs de **Scrapy** pour analyser les réponses et en extraire des éléments. Ce sont les classes gérant la partie Scraping de la collecte de données.
- **Item pipelines** : Le pipeline d'articles est responsable du traitement des articles qui ont été extraits par les Spiders. Elles sont généralement utilisées pour nettoyer, valider et persister les articles.

Dès que nous faisons une requête initiale, le moteur reçoit les différentes requêtes à explorer et les transmet au planificateur. Celui-ci stocke ces requêtes dans une file d'attente (la frontière) pour les alimenter plus tard lorsque le moteur les demandera. Le planificateur va ensuite retourner au moteur la requête suivante et celui-ci va à son tour, l'envoyer au téléchargeur. Une fois le téléchargement de la page terminé, le téléchargeur génère une réponse (avec cette page) et l'envoie au moteur. Lorsque le moteur reçoit la réponse, il la remet aux classes Spiders qui vont traiter celle-ci et renvoyer les éléments récupérés. Enfin, le moteur envoie les éléments traités aux pipelines d'éléments, puis envoie les demandes traitées au planificateur et demande les éventuelles prochaines requêtes à explorer. L'ensemble de ces processus est répété jusqu'à qu'il n'y est plus de requêtes dans la file d'attente du planificateur.

3.3 Solution proposée et sa mise en œuvre

3.3.1 Architecture de la solution

Avant de détailler la solution de notre problématique, visualisons tout d'abord l'architecture hiérarchique de notre traitement de données. La première étape est bien entendue l'extraction des recettes sur le site web **Marmiton** à l'aide du framework **Scrapy**. Puis nous effectuons le nettoyage des données et

le formatage de celles-ci. L'ensemble des recettes structuré sera ensuite utilisé pour le partitionnement via un K-Means Clustering.

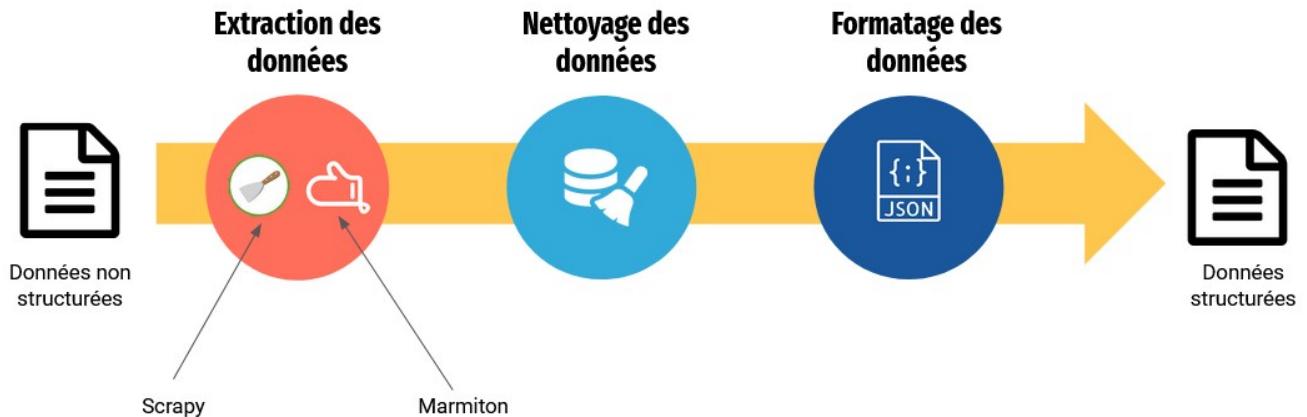


FIGURE 3.6 – Architecture hiérarchique des traitements de nos données

3.3.2 Extraction des données

Pour effectuer l'extraction de nos données, nous devons tout d'abord définir une liste d'URLs d'entrées où le Crawler de **Scrapy** effectuera son processus de crawling. Dans notre cas, cette liste d'URLs concerne les catégories principales des recettes qui sont : "Entrées", "Plat principal", "Apéritifs", "desserts", "Sauces" et "boissons". Chaque recette du site web **Marmiton** contient plusieurs catégories réparties hiérarchiquement. Sur la figure 3.7, nous pouvons voir que la recette contient la catégorie principal "Plat principal" et plusieurs sous-catégories qui donnent une définition plus précise de celle-ci.

Remarque : L'ensemble des processus décrit ci-dessous a été implémenté à l'aide de la solution proposée dans le site web **ledatascientist.com** (voir la source [Oli21] dans la bibliographie).



FIGURE 3.7 – Exemple de la structure hiérarchique des catégories pour une recette

Pour permettre au crawleur de naviguer entre les différentes pages, nous créons des règles afin de définir un type de comportement pour l'exploration du site web. Dans ces règles, nous utilisons des liens d'extraction qui définissent comment les liens URLs seront extraits de chaque page crawlée. Le premier objectif de notre crawleur est d'aller d'une page à une autre automatiquement. Pour ce faire, nous créons une règle dans laquelle nous définissons les liens (ou régions) qui doivent être extraites à l'intérieur d'une page. Dans notre cas, ces liens sont représentés par les boutons de pagination d'une page. Pour fournir les régions au crawleur, nous utilisons le langage de requête XPath qui permet de faire des requêtes sur des documents XML et HTML. Cette règle nous permet donc de visiter l'ensemble des pages de chacune des catégories de recettes.

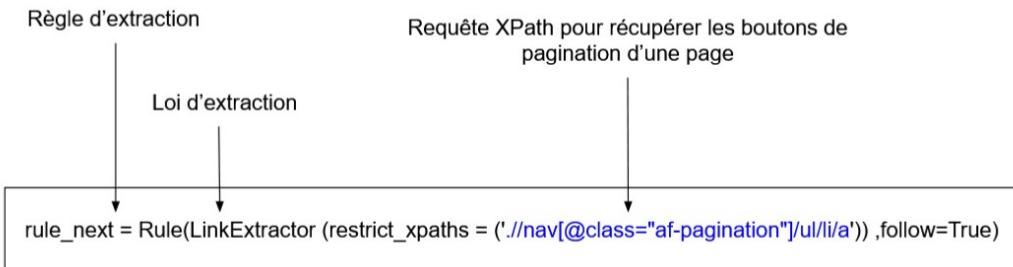


FIGURE 3.8 – Règle d'extraction des boutons de pagination

La deuxième règle que nous devons créer est celle permettant au crawleur de rentrer dans la page de détail d'une recette afin d'analyser les données de celle-ci. Nous allons donc créer une règle afin d'extraire les liens URLs contenant le mot "recipe". Chaque lien produit sera utilisé pour générer un objet *Request*, qui contiendra le texte du lien dans son méta-dictionnaire. Dans la loi d'extraction de la règle, nous insérons l'URL "https://www.marmiton.org/recettes/". Par conséquent, l'ensemble des URLs n'ayant pas cette syntaxe ne sera pas traité. Une fois le processus d'extraction terminé, nous appelons une fonction de rappel qui va analyser le contenu extrait.

Remarque : Exemple d'URL extraite :

"https://www.marmiton.org/recettes/recette_tacos-mexicains_34389.aspx"

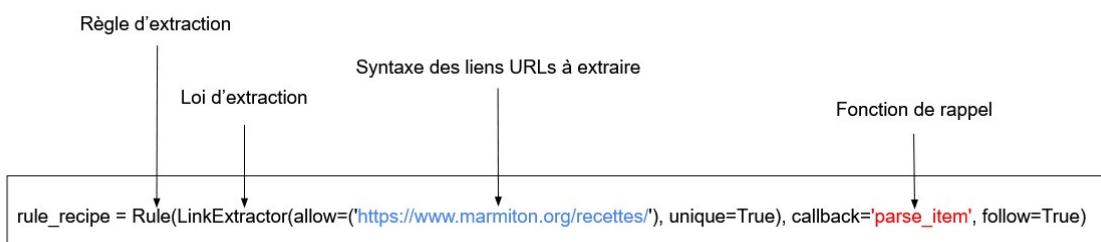


FIGURE 3.9 – Règle d'extraction des liens URLs

La fonction d'appel est appelée pour chaque réponse produite pour les URLs. Pour chaque réponse, nous recevons une page contenant une recette avec l'ensemble des informations de celle-ci en format HTML. Avec l'utilisation du langage XPath, nous sommes capables d'effectuer des requêtes afin d'extraire les données de la recette telles que son nom, son titre, ses ingrédients ou encore son image. L'ensemble de ces caractéristiques est stocké dans un dictionnaire de données.

3.3.3 Nettoyage, formatage et insertion des données

Après avoir collecté nos données, nous devons les nettoyer et les normaliser :

- les images URLs des recettes vont être standardisées pour permettre un rendu homogène de celles-ci lors de leurs affichages sur l'application web.
- les catégories des recettes vont être standardisées (séparation des catégories d'une recette, etc...).
- les recettes en doublons vont être supprimées.
- certaines informations vont être formatées afin de régler les problèmes d'encodage à l'écriture.

Suite à ce nettoyage, nous formatons le dictionnaire de données sous format JSON puis nous insérons l'ensemble des données dans notre base de données locale qui possède 6 tables :

- Recipe : contient les informations d'une recette.
- Client : contient les données personnelles d'un client.

- Ingredient : stocke les ingrédients distincts de toutes les recettes
- Contain_recipe_ingredient : effectue la liaison entre les recettes et leurs ingrédients (avec leurs quantités).
- Assess : contient les évaluations des recettes par les clients.
- Have_preferences_ingredient : possède les ingrédients de préférence des clients.

Remarque : Seules les tables Recipe, Ingredient, Contain_recipe_ingredient et Have_preferences_ingredient sont impactées par l'insertion des données. De plus, certains attributs comme "id_cluster" ou "coordinates" de la table Recipe sont vides puisqu'ils seront remplis lors du partitionnement de nos données.

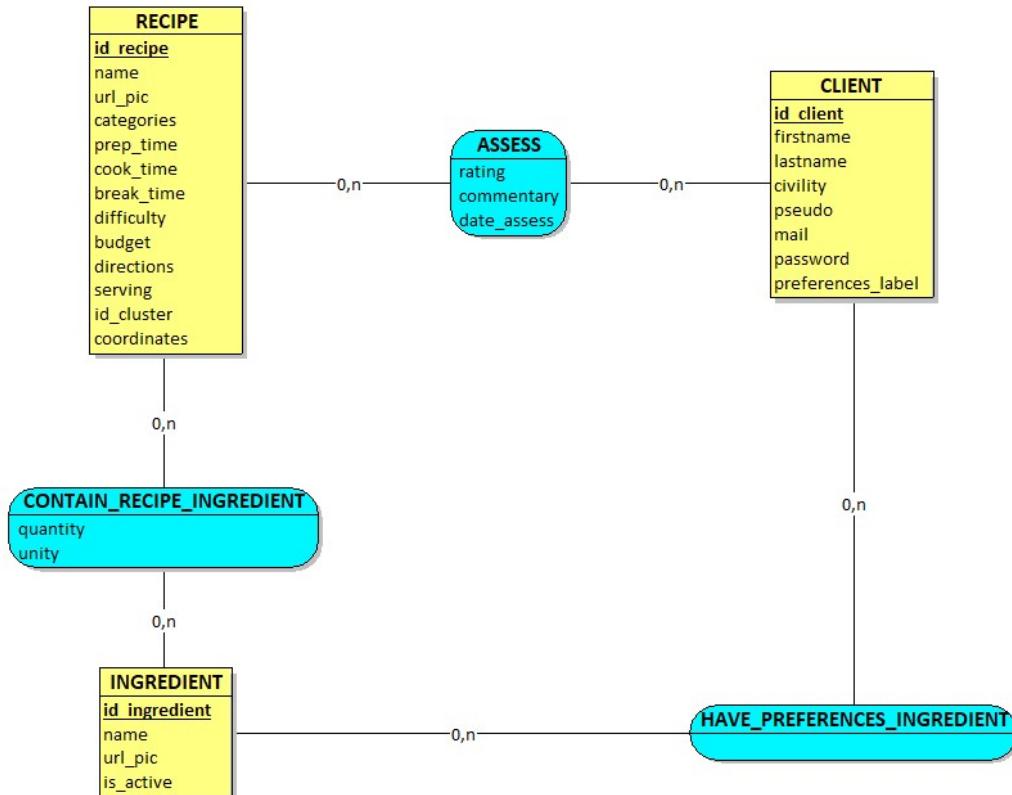


FIGURE 3.10 – Schéma Merise de notre base de données

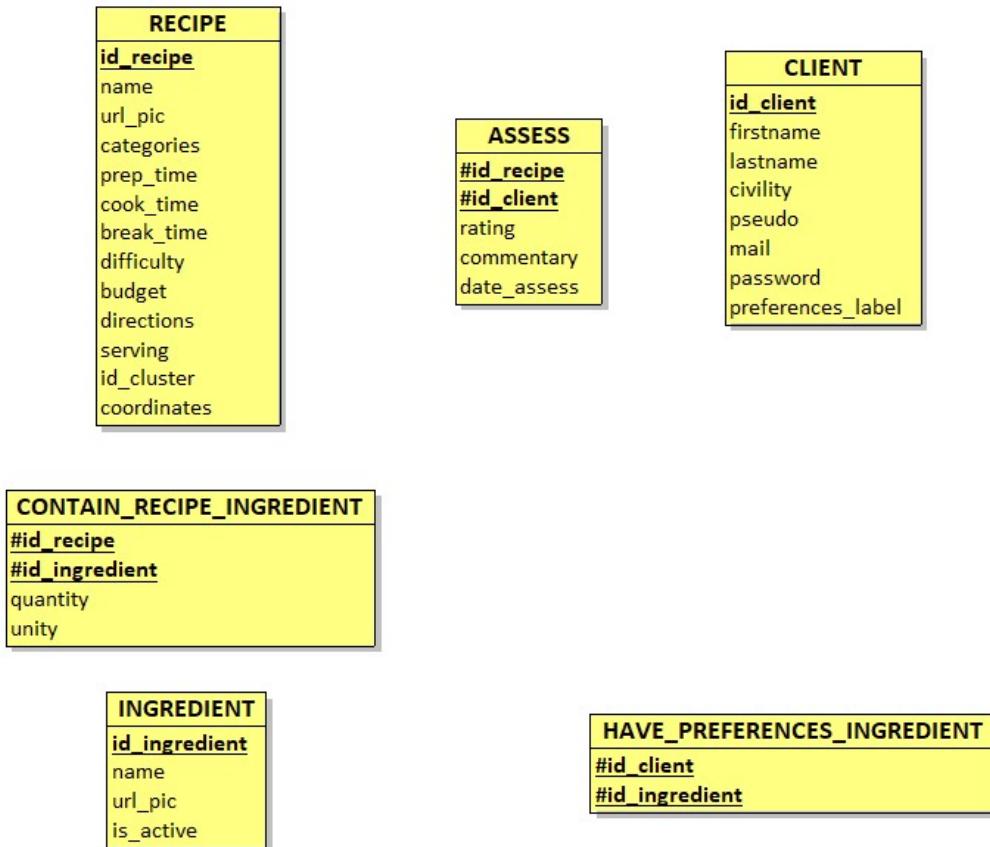


FIGURE 3.11 – Schéma logique de notre base de données

3.4 Tests et certifications de la solution

L'ensemble des processus précédemment expliqués nous ont permis d'insérer plus de 2426 recettes dans notre base de données. Sur la figure 3.12, vous pouvez voir un extrait de nos données collectées sous format JSON. Certains attributs comme celles des "étapes" contiennent des problèmes d'encodage. Cependant, ceux-ci ont été corrigés lors de l'ajout de la recette dans la base de données.

```

{
  "id": 1625,
  "nom": "Mont d'Or au four",
  "img_url": "https://assets.afcdn.com/recipe/20220307/129750_w1000h1000.jpg",
  "category": ["Plat principal", "Plats au fromage", "Mont d'or"],
  "time_prep": "10\u00a0min",
  "time_repo": "-",
  "time_cuisson": "20\u00a0min",
  "difficulty": "tr\u00e0u00e8s facile",
  "budget": "moyen",
  "etapes": [
    "Le Mont d'Or est un fromage du Haut-Jura, vendu \u00e0 la coupe ou dans une bo\u00e0eete en \u00e9pic\u00e9e.",
    "Prenez votre Mont d'Or, faites un trou au milieu dans lequel vous verserez un peu de vin blanc.",
    "Placez le fromage au four \u00e0 180\u00b0C (thermostat 6) jusqu'\u00e0 ce qu'il soit fondu.",
    "Accompagnez de pommes de terre vapeur, de charcuterie, et d'un vin blanc."
  ],
  "ingredients": [
    {
      "id_ingredient": 1232,
      "nom_ingredient": "vin blanc",
      "quantity": ["5", "\u00a0cl"],
      "image_ingredient": "https://assets.afcdn.com/recipe/20170607/67771_w96h96c1cx350cy350.jpg"
    },
    {
      "id_ingredient": 399,
      "nom_ingredient": "mont d'or",
      "quantity": ["1"],
      "image_ingredient": "https://assets.afcdn.com/recipe/20171229/76410_w96h96c1cx2500cy1681cxb5000cyb3362.jpg"
    }
  ]
}



Problème d'encodage


```

FIGURE 3.12 – Extrait du fichier JSON pour les recettes collectées

Recette :	id_recipe	name	categories	url_pic	directions	prep_time	cook_time	break_time	difficulty	budget
	1625	Mont d'Or au four	Plat principal;Plats au fromage;Mont d'or	https://assets.afcdn.com/recipe/20220307/129750_w1...	Le Mont d'Or est un fromage du Haut-Jura, vendu à ...	10 min	20 min	-	très facile	moyen

IngREDIENTS de la recette :	name	id_ingredient	name	quantity	unity	url_pic
	Mont d'Or au four	399	mont d'or	1		https://assets.afcdn.com/recipe/20171229/76410_w96...
	Mont d'Or au four	1232	vin blanc	5	cl	https://assets.afcdn.com/recipe/20170607/67771_w96...

FIGURE 3.13 – Exemple de recette dans la base de données locale

Chapitre 4

Partitionnement des données

Dans ce chapitre, nous étudierons les moyens mis en place pour effectuer le partitionnement de nos données. Nous effectuerons un état de l'art des différentes méthodes existantes puis nous expliquerons les solutions mises en place. Enfin, nous analyserons les résultats obtenus à l'aide de graphes.

4.1 Analyse de la problématique

Les recettes extraites à l'aide du Data Scraping contiennent des catégories prédéfinies, mais celles-ci ne reflètent pas entièrement ce qu'est la recette. Par conséquent, la similarité entre deux recettes peut être faible même si elles sont dans la même catégorie. Ci-dessous, sur la figure 4.1, vous pouvez constater que la recette "Cake au Chocolat" est beaucoup plus proche de la recette "Mousse au Chocolat" que celle du "Cake au jambon" alors qu'elles ne se situent pas dans la même catégorie. Il est donc préférable d'effectuer une catégorisation personnalisée qui comparera les ingrédients des recettes et non leurs catégories.

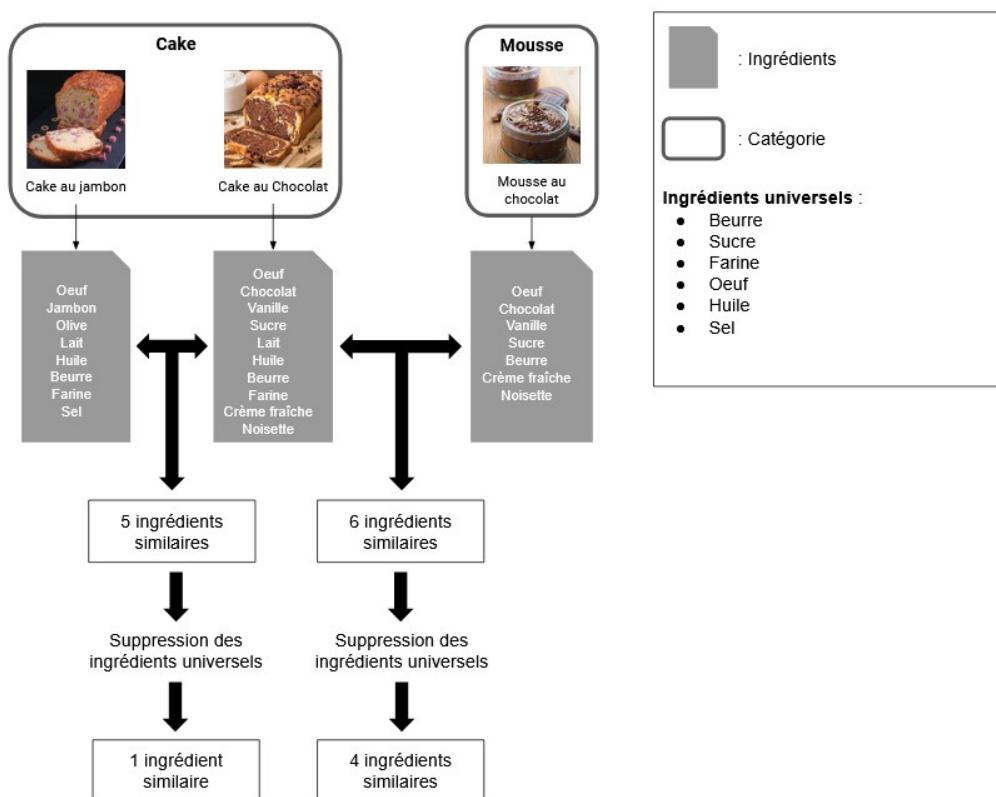


FIGURE 4.1 – Relation des recettes selon leurs ingrédients

Pour permettre la catégorisation de nos recettes, il nous faudra partitionner celles-ci afin de trouver des groupes distincts (ou "clusters") au sein d'un ensemble de données. De nombreux outils de clustering et Machine Learning existent et permettent de créer des groupes où les éléments d'un groupe similaire auront, en général, des caractéristiques similaires les uns aux autres. Dans notre cas, ces caractéristiques seront nos ingrédients.

Nous utiliserons l'algorithme K-Means Clustering de la librairie **Scikit-Learn** qui est un algorithme non supervisé très connu en matière de Clustering. Notre principale difficulté dans ce processus est donc de transformer nos recettes en données géométriques pour pouvoir les stocker dans un plan multidimensionnel et effectuer l'apprentissage non supervisé de notre outil de clustering.

Les algorithmes de clustering tels que le K-Means Clustering ont du mal à regrouper avec précision des données de haute dimensionnalité. Lorsque notre algorithme de clustering a trop de dimensions, les paires de points commencent à avoir des distances très similaires et nous ne serions pas en mesure d'obtenir des clusters significatifs. Par conséquent, une réduction de dimensionnalité (via un **PCA**) de nos données devra être effectuée pour avoir le meilleur partitionnement possible.

4.2 Etat de l'art : études des solutions existantes

4.2.1 K-Means Clustering

Le but des algorithmes de clustering est de partitionner un ensemble de points de données en groupe appelés clusters, de sorte que les dissemblances entre ceux affectés au même cluster tendent à être plus petites que celles des points de données situés dans différents clusters.

Le K-Means Clustering est l'un de ces algorithmes. Le nombre de clusters doit être connu a priori et la similarité entre les points de données est exprimée à l'aide d'une certaine forme de métrique de distance. Un tel exemple est la distance euclidienne au carré. On note la distance entre les points a et b par $d(a, b)$. Chaque cluster C est alors représenté par la moyenne μ_C , qui est définie par la formule suivante où $|C|$ représente le nombre de points dans le cluster C .

$$\mu_C = \frac{1}{|C|} \sum_{x \in C} x$$

FIGURE 4.2 – Calcul des centroides

La moyenne représente le point "milieu" du cluster, elle est également appelée centroïde. L'objectif est de trouver les centroïdes qui minimisent la somme de toutes les distances au carré de toutes les observations dans chaque groupe. Autrement dit, pour chaque cluster, nous voulons trouver une moyenne μ telle que :

$$\arg \min_{\mu} \sum_{x \in C} d(x, \mu)$$

FIGURE 4.3 – Calcul des distances entre les centroides et les points

Comme nous pouvons le constater sur la figure 4.2, la première étape consiste à sélectionner aléatoirement un centroïde pour chaque cluster. L'étape suivante attribue chaque point de données au centroïde le plus proche. Ensuite, tous les centroïdes sont mis à jour en calculant la moyenne de tous

les points de données observés qui leur sont attribués à l'aide de la formule de la figure 4.3.

La différence entre les nouvelles et anciennes valeurs des centroïdes est ensuite calculée et comparée à un seuil pour déterminer si l'algorithme doit continuer ou s'il peut être arrêté car les centroïdes n'ont pas bougé de manière significative. Si ce seuil n'est pas atteint, nous attribuons à nouveau des points de données aux centroïdes de cluster nouvellement définis.

Algorithme 1 : Kmeans Clustering

1. Les centroïdes sont initialisés avec des points aléatoires du jeu de données.
 2. Tous les autres points du jeu de données sont affectés à leur centroïde le plus proche en fonction de la mesure de distance choisie
 3. Pour chaque cluster, la moyenne de tous les points attribués est calculée et devient le nouveau centroïde
 4. Si le nombre d'itérations dépasse le nombre de boucles maximum ou si un critère de convergence donné est satisfait, terminez l'algorithme, sinon exécutez l'étape 2
-

FIGURE 4.4 – Algorithme du K-Means Clustering

Sur la figure 4.5 ci-dessous, nous pouvons voir un exemple de fonctionnement de l'algorithme K-Means Clustering en utilisant deux clusters sur un ensemble de points de données bidimensionnel. Tout d'abord, deux points sont choisis au hasard comme centroïdes initiaux (croix noires) et les points sont attribués au centroïde le plus proche. Ensuite, la moyenne intra-cluster est calculée et utilisée pour mettre à jour les centroïdes. Enfin, les points sont réaffectés au nouveau centroïde le plus proche.

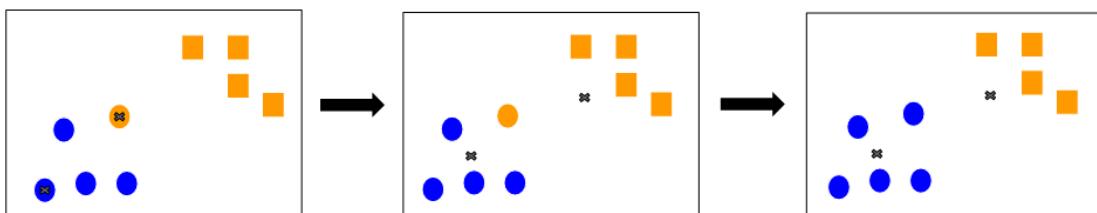


FIGURE 4.5 – Exemple de fonctionnement du K-Means Clustering

4.2.2 Principal Component Analysis (PCA)

Le PCA est un algorithme de réduction dimensionnelle non supervisé capable d'identifier les corrélations et patterns dans un jeu de données et de les transformer en un ensemble de données avec un nombre réduit de variable en minimisant la perte d'information. En d'autres termes, c'est une méthode classique que nous pouvons utiliser pour réduire les données de grandes dimensions à un espace de faible dimension.

Ce procédé est réalisé par la projection de notre jeu de données initiale dans un espace réduit, en utilisant les vecteurs propres. La projection est la fonction qui permet de représenter des points dans un espace plus petit impliquant une perte d'informations. Pour minimiser la perte, il nous faut :

- Maximiser la variance "expliquée" de nos projections.
- Minimiser la distance entre nos données et nos projections graphique.

Dans l'image ci-dessous (figure 4.6), nous avons un objet 3D qu'on projette sur un espace 2D. Une des pertes d'informations ici, se situe au niveau de la profondeur du cube.

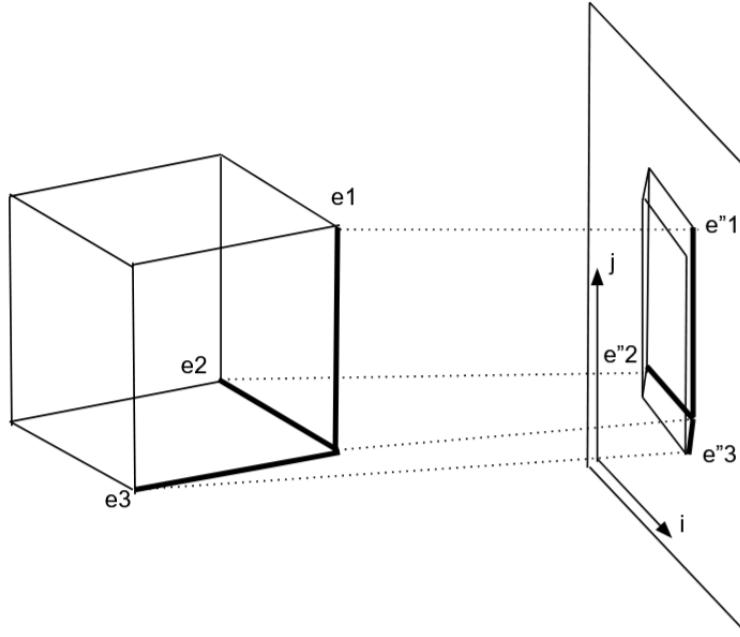


FIGURE 4.6 – Projection d'un cube 3D sur un espace 2D

4.3 Solution proposée et sa mise en œuvre

La solution que nous avons implémentée repose sur trois aspects qui sont, la transformation de nos recettes en vecteurs binaires, la réduction de dimension de nos vecteurs et la recherche du nombre de clusters optimale pour notre partitionnement de données.

Remarque : Les solutions proposées ci-dessous ont été implémenté à l'aide de trois sources distincts. Pour le redimensionnement de nos vecteurs via PCA (sous-section 4.3.2), nous avons utilisé la solution proposée par la source [Kan19]. Pour le calcul du nombre optimal de clusters (sous-section 4.3.3), nous nous sommes basé sur la source [Ban21]. Enfin, pour le calcul de la silhouette, la source [Kum20] a été étudiée(section 4.4).

4.3.1 Transformation des recettes en vecteurs binaires

La méthode de partitionnement du K-Means Clustering repose sur le calcul de distance entre plusieurs points, par conséquent, nous devons transformer nos recettes pour qu'elles puissent être intégrées dans un plan dimensionnel. Pour ce faire, nous définissons chaque recette comme un point dans un espace euclidien de dimension K où chaque dimension représente un ingrédient (K étant le nombre total d'ingrédient). Nous ne considérons que les ingrédients fréquents de notre ensemble de données au lieu de tous les ingrédients. Plus précisément, nous supprimons les ingrédients dits "universels" qui apparaissent de façon très fréquents dans l'ensemble des recettes (par exemple : "sel", "eau", "huile", "poivre", etc...). Chaque recette est alors représentée sous forme de vecteur, où la valeur à chaque index correspond à un ingrédient et est :

- mise à 0 si l'ingrédient n'est pas présent.
- mise à 1 si l'ingrédient est présent

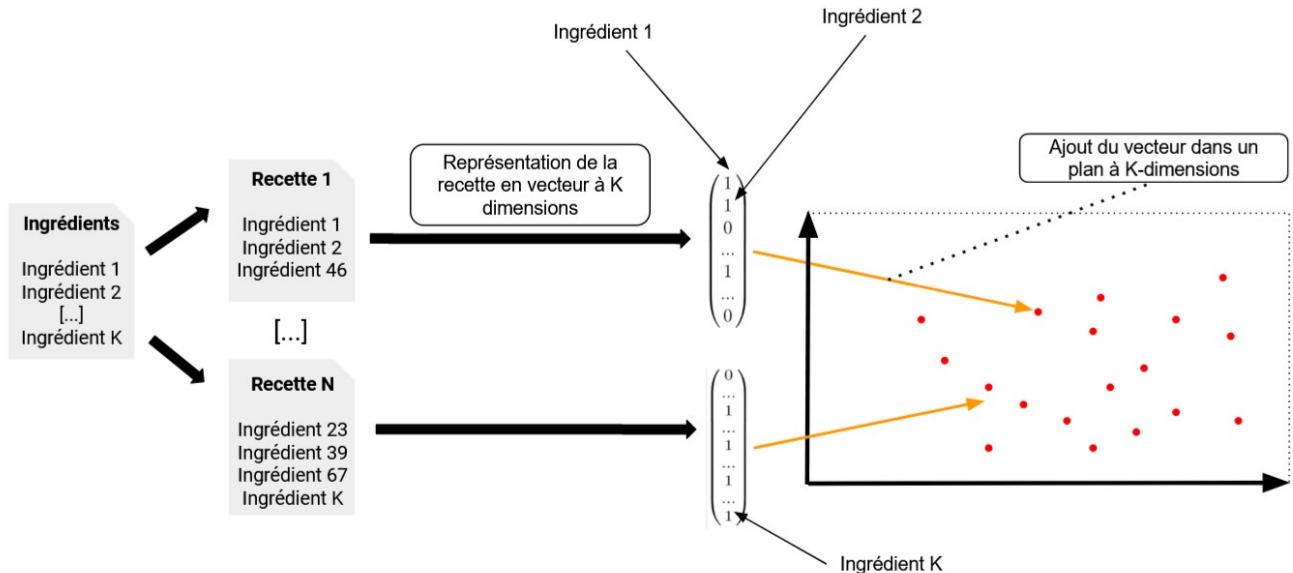


FIGURE 4.7 – Vectorisation des recettes

4.3.2 Réduction de dimension de nos vecteurs

Nos vecteurs étant construit par rapport aux ingrédients, leurs tailles sont beaucoup trop élevées (plus nous avons d'ingrédients distincts, plus la taille des vecteurs augmente). Nous utilisons donc le principal component of analysis (PCA) afin de réduire drastiquement la dimension de nos vecteurs sans avoir un réel impact négatif sur leurs précisions. De plus, cette réduction nous permet de grandement faciliter la visualisation et l'analyse de nos différents clusters.

Pour pouvoir décider de la meilleure dimension pour nos vecteurs, nous instancions un objet PCA de la librairie **Scikit-Learn** et nous lui passons en paramètre notre matrice contenant l'ensemble de nos vecteurs binaires afin d'ajuster nos données. Ensuite, nous calculons la variance "expliquée" de notre PCA. La variance "expliquée" est utilisée pour mesurer l'écart entre un modèle et les données réelles. En d'autres termes, c'est la partie de la variance totale du modèle qui s'explique par des facteurs réellement présents et qui n'est pas dû à la variance d'erreur.

Le graphique sur la figure 4.8 montre la quantité de la variance "expliquée" (sur l'axe des y) en fonction du nombre de composants (c'est-à-dire le nombre de dimensions) que nous incluons (l'axe des x). Une des règles empiriques consiste à conserver au minimum 70 % de la variance pour ne pas avoir un impact négatif sur la précision de nos vecteurs. Donc, dans notre cas, nous sommes passé d'un vecteur initiale à plus de 700 composants à un vecteur d'environ 130 composants.

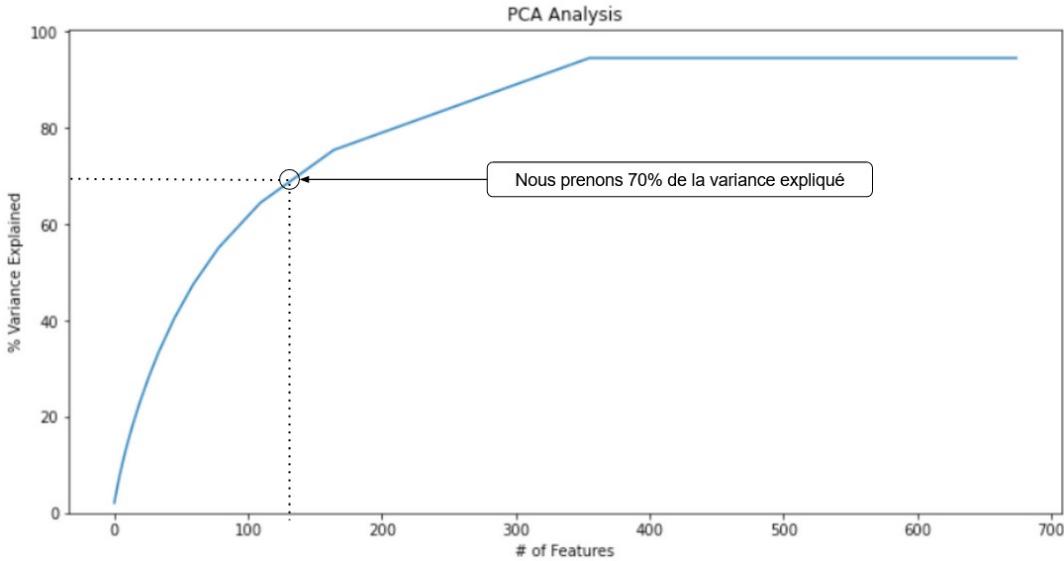


FIGURE 4.8 – Réduction de dimensions via PCA

4.3.3 Recherche du nombre optimal de clusters

La dernière étape avant d’entrainer notre K-Means Clustering est de trouver le nombre optimal de clusters à l’aide de l’inertie qui permet de donner une indication de la cohérence des différents clusters. L’inertie mesure à quel point un ensemble de données a été regroupé par un K-Means Clustering. Elle est calculée en mesurant la distance entre chaque point de données et son centre de gravité, en élevant cette distance au carré et en additionnant ces carrés sur un cluster.

Notre objectif est donc de minimiser l’inertie à l’intérieur des clusters et maximiser celle entre les différents clusters afin de bien les distinguer. Pour ce faire, nous utiliserons la méthode Elbow qui est une méthode empirique, et qui consiste à calculer la variance des différentes tailles de clusters envisagées puis à placer les variances obtenues sur un graphique. Lorsque le nombre de clusters k est à 1, la somme intra-cluster du carré sera élevée. Au fur et à mesure que la valeur de k augmente, la somme des valeurs au carré au sein du cluster diminue. L’objectif est de visualiser le graphique et de trouver le point où la décroissance de l’inertie commence à ralentir brusquement.

Sur la figure 4.9, nous utilisons la méthode Elbow pour un nombre de clusters allant de 1 à 14. Nous constatons que la ligne du graphe diminue brusquement pour un nombre de clusters égale à 7. C’est donc ce nombre que nous utiliserons pour entraîner notre K-Means Clustering.

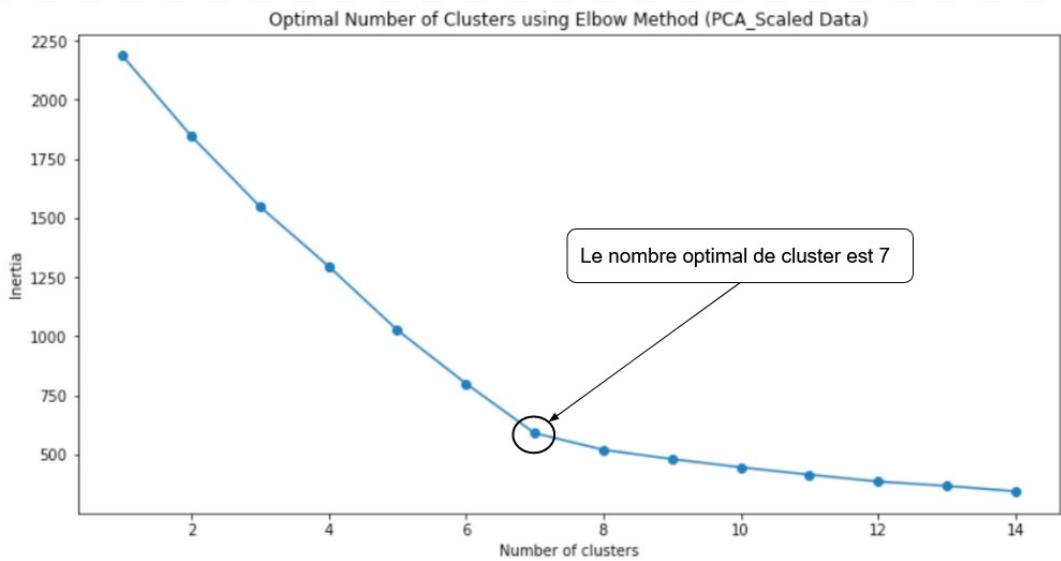


FIGURE 4.9 – Recherche du nombre de clusters optimale à l'aide de la méthode Elbow

Après avoir effectué le partitionnement de nos données avec le K-Means Clustering, nous insérons dans la base de données, l'ensemble des coordonnées des recettes (c'est-à-dire les points de notre partitionnement). Ces coordonnées vont nous permettre ensuite de trouver la liste des recettes proches de chacune des recettes de notre base. De plus, nous insérons dans la base, les numéros de cluster pour chaque recette.

4.4 Tests et certifications de la solution

Afin de tester et valider notre solution, nous calculons la silhouette de notre partitionnement en passant en paramètre le résultat de notre K-Means Clustering. L'analyse de la silhouette est utilisé pour étudier la distance de séparation entre les différents clusters. Le tracé de la silhouette affiche une mesure de la proximité de chaque point d'un cluster avec les points des clusters voisins et fournit ainsi un moyen d'évaluer visuellement notre partitionnement. La valeur du score de la silhouette varie de -1 à 1, si le score est de 1, le cluster est dense et bien séparé des autres. Une valeur proche de 0 représente des clusters qui se chevauchent avec des échantillons très proches de la frontière de décision des clusters voisins. Un score négatif [-1, 0] indique que les échantillons ont peut-être été affectés aux mauvais clusters.

Sur la figure 4.10, se trouve le résultat de cette évaluation avant d'effectuer la réduction de dimension via PCA et après utilisation du PCA. Nous constatons que la réduction de dimension de nos vecteurs a permis une amélioration significante de la silhouette de notre partitionnement puisque nous sommes passé d'une valeur d'environ 8% à 58% de la silhouette (soit environ 0.48).

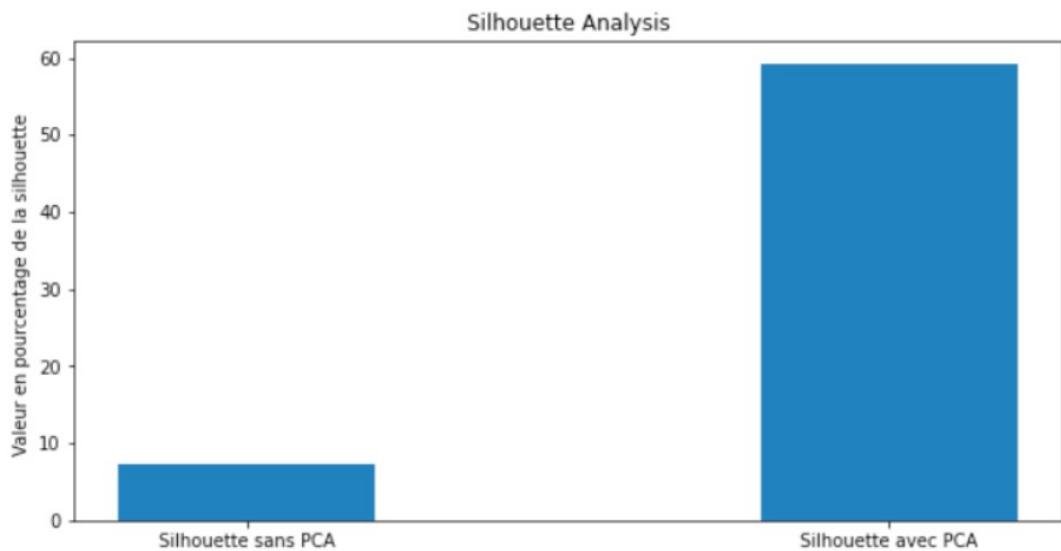


FIGURE 4.10 – Calcul de la silhouette de notre partitionnement de données

En utilisant le PCA pour réduire l'ensemble de données à 110 composants principaux, nous pouvons tracer les clusters du K-Means clustering en visuels 2D et 3D. Les visualisations PCA ont tendance à agréger les clusters autour d'un point central, ce qui rend l'interprétation difficile dans un espace 2D. Cependant, lorsque nous utilisons un espace 3D, nous pouvons clairement distinguer tous les clusters.

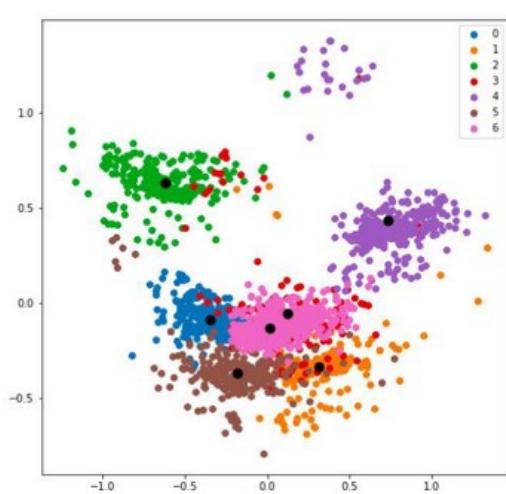


Schéma: Plan 2D - Kmeans Clustering

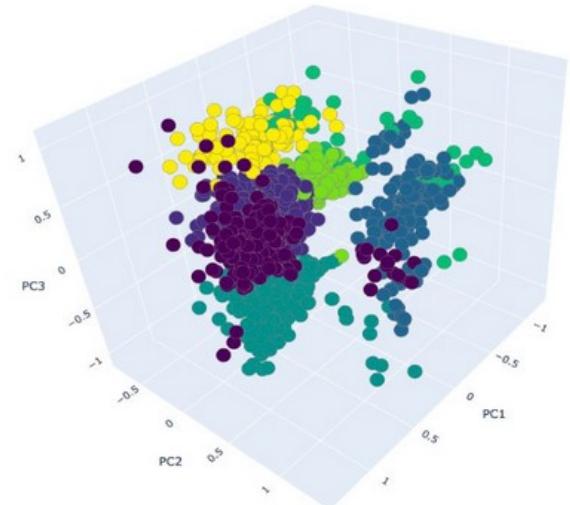


Schéma: Plan 3D - Kmeans Clustering

FIGURE 4.11 – Plan 2D et 3D de notre partitionnement de données

Chapitre 5

Classification des données

Ce chapitre sera consacré à la classification de nos données. Nous expliquerons l'intérêt de l'utilisation d'un arbre de décision, puis nous détaillerons l'ensemble des processus mis en place pour élaborer cet arbre. Enfin, nous analyserons les résultats obtenus de notre solution.

5.1 Analyse de la problématique

Quand bien même le partitionnement de nos données a permis de catégoriser nos recettes et de les regrouper dans des clusters, un problème persiste. Une des fonctionnalités de l'utilisateur sur notre application est de rechercher des recettes selon des ingrédients demandés. La problématique est donc de savoir comment récupérer les recettes qui ont ces ingrédients ? En ayant une vision orientée "Big Data", il est inconcevable de vérifier dans l'ensemble des données de notre base de données, les recettes qui possèdent ces ingrédients car le temps d'exécution pour parcourir celles-ci serait trop long. De plus, cela peut faire transiter sur le réseau une grande masse de données inutilement.

Il nous faut donc trouver un système permettant de réduire considérablement ce temps d'exécution. Pour ce faire, nous utiliserons un arbre de décision de la librairie python **Scikit-Learn** et qui sera formé à l'aide des données partitionnées par notre Kmeans Clustering.

L'objectif de cet arbre est de pouvoir prédire le cluster possédant les recettes ayant le plus les ingrédients demandés. En d'autres termes, cet arbre nous permettra d'effectuer une recherche de recette dans un cluster (et donc dans une zone de recherche plus petite) sans avoir un impact négatif trop important au niveau de la précision de notre requête.

D'autre part, l'utilisateur a aussi la possibilité d'ajouter de nouvelles recettes dans la base de données. Une des questions que l'on pourrait se poser est de savoir comment assigner cette nouvelle recette dans un cluster ? Une possible réponse à cette question est d'utiliser un arbre de décision. Cet arbre prédira le meilleur cluster d'une recette selon ses ingrédients et nous permettra d'éviter un calcul beaucoup plus complexe de recherche (par exemple un calcul de distance euclidienne entre les coordonnées de recettes).

Notre application web étant codé en PHP, il nous faut sérialiser l'arbre de décision et implémenter un service web SOAP afin d'appeler l'arbre sur l'application. Ce service web sera élaboré avec la librairie python **Ladon** qui permet de générer automatiquement le WSDL de notre service SOAP.

5.2 Etat de l'art : études des solutions existantes

5.2.1 Arbre de décision

Les algorithmes d'apprentissage automatique sont divisés en différentes catégories, dont l'une est l'apprentissage supervisé. Dans l'apprentissage supervisé, la tâche consiste à déduire une fonction à partir d'un ensemble de données étiqueté. L'ensemble de données contient des règles sous forme de

tuples qui sont transmis à l'algorithme. L'algorithme utilise ces données pour déduire une fonction qui peut ensuite être utilisée pour prédire la sortie de valeurs. La classification est le sous-ensemble des techniques d'apprentissage supervisé qui considère des catégories distinctes comme résultat cible.

Les méthodes d'apprentissage automatique basées sur des arbres, partitionnent de manière récursive, l'espace des caractéristiques en un ensemble d'espaces, puis intègrent un modèle simple dans chacun d'eux. Le partitionnement est représenté à l'aide d'une structure arborescente composée de nœuds de décision internes et de feuilles terminales. Chaque nœud de décision implémente une fonction de test avec des résultats discrets étiquetant les branches. Pour prédire le résultat d'une entrée, l'arbre est parcouru à partir de la racine jusqu'à ce qu'une feuille soit atteinte, moment auquel la catégorie affectée à la feuille constitue la sortie. Au cours de ce processus, un test est appliqué à chaque nœud et, en fonction du résultat, l'une des branches est prise.

La figure 5.1 montre un exemple du fonctionnement de l'algorithme en deux dimensions. Les caractéristiques x et y doivent être utilisées pour distinguer les objets de la catégorie $C1$, des objets de la catégorie $C2$. Premièrement, l'espace est divisé en deux régions par une ligne verticale correspondant à la valeur $w1$ de la caractéristique x , et deuxièmement, l'espace est à nouveau divisé en deux en utilisant une ligne horizontale correspondant à la valeur $w2$ de la caractéristique y . Les régions résultantes contiennent chacune une classe d'objets, et nous pouvons donc construire un arbre de décision qui mappe les objets à une catégorie en comparant leur coordonnée (x_1, x_2) avec $w1$ et $w2$.

Par exemple, un objet avec une coordonnée x inférieure à $w1$ et y inférieur à $w2$ tombera à gauche de la ligne verticale et en bas de la ligne horizontale, il sera donc affecté à $C1$ car la condition dans le nœud racine n'est pas satisfaite.

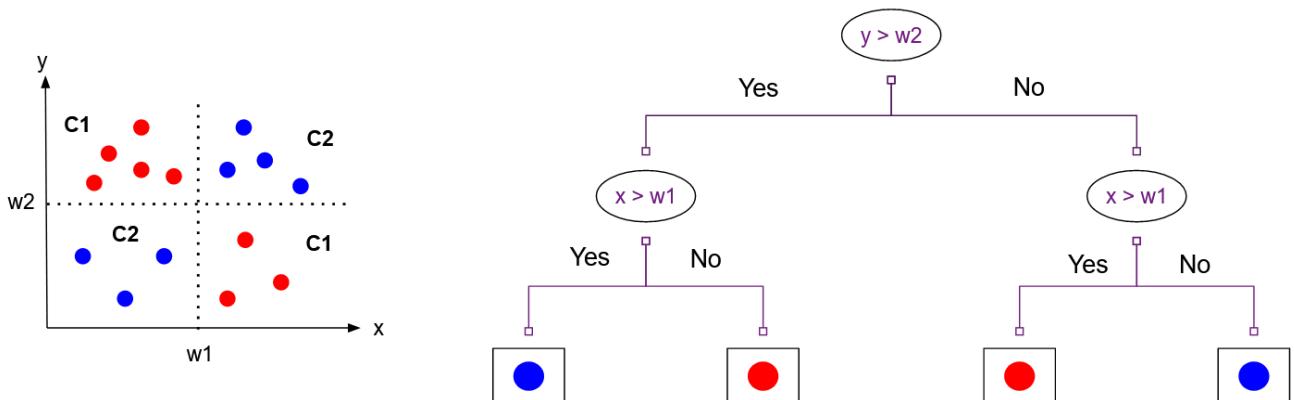


FIGURE 5.1 – Fonctionnement de l'arbre de dimension 2D

À des fins de classification, une mesure d'impureté est utilisée pour quantifier la qualité d'une division de région, déterminant ainsi la forme de l'arbre. Une scission est dite pure, si après la scission, pour toutes les branches, toutes les instances choisissant une branche appartiennent à la même classe. Dans un tel cas, il n'est pas nécessaire de diviser davantage, et un nœud feuille avec l'étiquette correspondante peut être ajouté à l'arbre. Si un nœud n'est pas pur, les instances doivent être déversées pour diminuer l'impureté jusqu'à ce qu'un critère d'arrêt soit atteint.

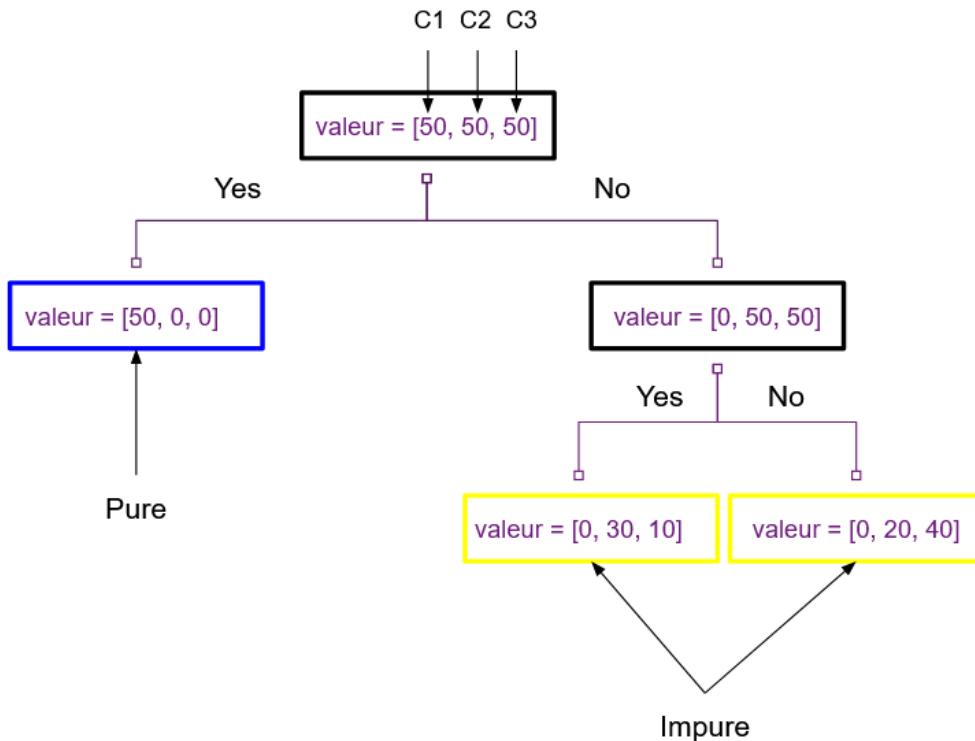


FIGURE 5.2 – Pureté d'un arbre de décision

Le but d'un arbre de décision est donc de créer les classes/catégories les plus pures possible pour résoudre au mieux notre problème et répartir de façon optimale nos caractéristiques. Dans l'implémentation de l'arbre de décision de la librairie python **scikit-learn**, cela se fait par le paramètre 'criterion' qui possède deux mesures : l'**Indice Gini** et l' **Entropy**.

5.2.2 Classification naïve Bayésienne

La classification naïve Bayésienne fait référence à un ensemble d'algorithmes de classification basé sur l'application du théorème de Bayes avec l'hypothèse "naïve" que toutes les variables sont indépendantes par paires. En utilisant différentes distributions, il est alors possible d'obtenir différents algorithmes.

Étant donné une variable de catégorie y et un certain nombre de caractéristiques x_1, \dots, x_n , le théorème de Bayes stipule que la probabilité conditionnelle que y soit une certaine valeur Y , connaissant la valeur de tous x_1, \dots, x_n , peut être décomposée comme :

$$P(y|x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n|y)}{P(x_1, \dots, x_n)},$$

FIGURE 5.3 –

Avec l'hypothèse naïve que toutes les caractéristiques x_1, \dots, x_n sont indépendantes deux à deux, nous pouvons appliquer à plusieurs reprises la règle d'indépendance $P(x_i, x_j|y) = P(x_i|y) * P(x_j|y)$ sur $P(x_1, \dots, x_n)$ et obtenir :

$$P(y|x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^n P(x_i|y)}{P(x_1, \dots, x_n)}.$$

FIGURE 5.4 –

Puisque $P(x_1, \dots, x_n)$ ne dépend pas de y , mais uniquement des valeurs d'entrée x_1, \dots, x_n , pour trouver la classe qui correspond avec la plus grande probabilité aux données d'entrée, nous pouvons utiliser la formule ci-dessous comme règle de classification :

$$y = \arg \max_y P(y) \prod_{i=1}^n P(x_i|y)$$

FIGURE 5.5 –

Avec la classification naïve Bayésienne, la fonction de distribution de probabilité P est :

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_i)^2}{2\sigma_y^2}\right)$$

FIGURE 5.6 –

5.2.3 Validation croisée K-Fold

La cross validation (ou validation croisée) est une méthode statistique qui permet d'évaluer les performances des modèles d'apprentissage automatique, et donc dans notre cas, d'analyser la précision de notre arbre de décision. Elle comprend les étapes suivantes :

- Les données d'origine sont divisées de manière aléatoire en k sous-échantillons qui deviennent les données d'apprentissage.
- les modèles sont estimés à l'aide de $k-1$ sous-échantillons pour chaque pli, le $k^{\text{ième}}$ sous-échantillon servant d'échantillon de validation.
- le processus se répète jusqu'à ce que chaque sous-échantillon ait servi de données de validation et que les résultats des modèles puissent être moyennés à travers les plis.

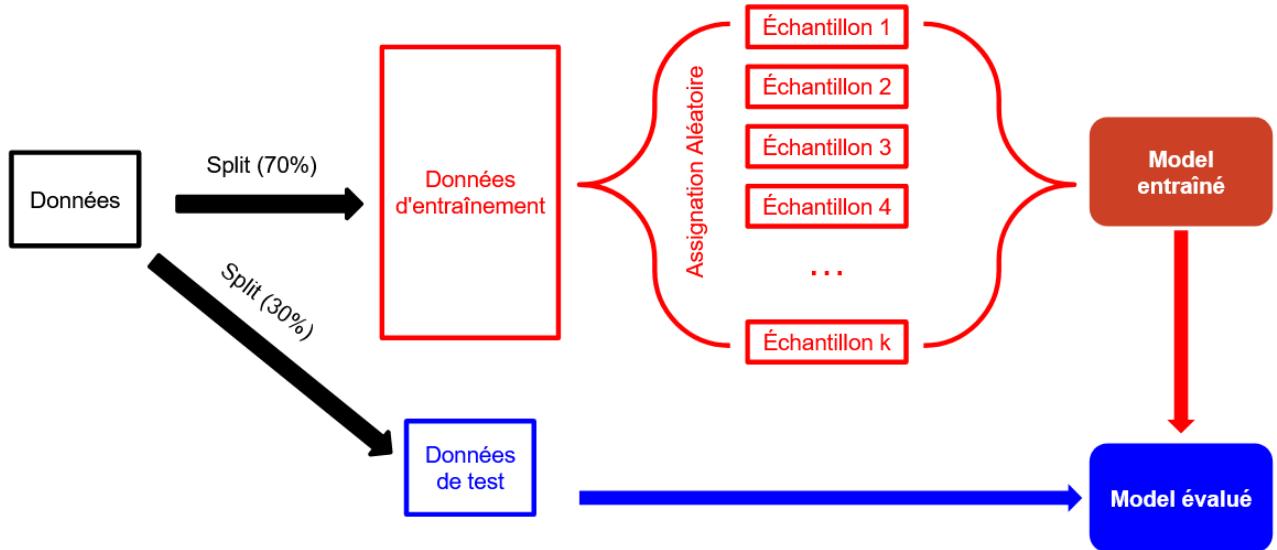


FIGURE 5.7 – Architecture de la validation croisée K-Fold

5.3 Solution proposée et sa mise en œuvre

5.3.1 Choix de l'algorithme d'apprentissage

Pour l'élaboration de notre solution, nous avons opté pour l'utilisation d'un arbre de décision. Même si les différences avec la classification naïve Bayésienne sont moindres, l'arbre possède de multiples fonctionnalités intéressantes qui prennent en charge divers problèmes tels que les valeurs manquantes et les valeurs aberrantes de nos règles de décisions. De plus, la classification naïve Bayésienne est principalement utilisée lorsqu'il y a un nombre conséquent de classes à prédire comme la classification de texte ou le filtrage de spam. Or, dans notre cas, les différentes classes à prédire ne sont pas élevées puisque ce sont les numéros de clusters.

Vous pouvez trouver dans la section 5.3.5, une comparaison des performances entre l'arbre de décision et la classification naïve Bayésienne.

5.3.2 Crédit des règles de décision

La première étape de l'élaboration de notre arbre est de définir nos règles de décision. La méthode est assez similaire à celle du K-Means Clustering puisque nous devons convertir nos recettes en vecteurs binaires à K dimensions, où K représente le nombre d'ingrédients distincts. Pour chaque recette partitionnée de notre base de données, nous vérifions pour chaque ingrédient distinct i si la recette le possède ou non. Si oui, dans ce cas, nous mettons la valeur 1 à l'index i du vecteur de la recette, sinon 0. Nous ajoutons à la fin de ces vecteurs un nouvel index qui contient le numéro de cluster de la recette. Ce numéro de cluster est le label de nos règles.

Algorithm 1: Création des règles de décision

Output : vectors : List[]
Local : db : mysql.connector
Begin

```
recipes ← db.getAllRecipes()
listIngredients ← db.getAllIngredients()
for recipe in recipes :
    vector ← [0] * len(listIngredients)
    for ingredient in listIngredients:
        if ingredient in recipe.getIngredients():
            vector[listIngredients.index(ingredient)] ← 1
    vector[len(listIngredients + 1) ← recipe.getCluster()]
    vectors.append(vector)
return vectors
```

End

FIGURE 5.8 – Algorithme de la création des règles de décision

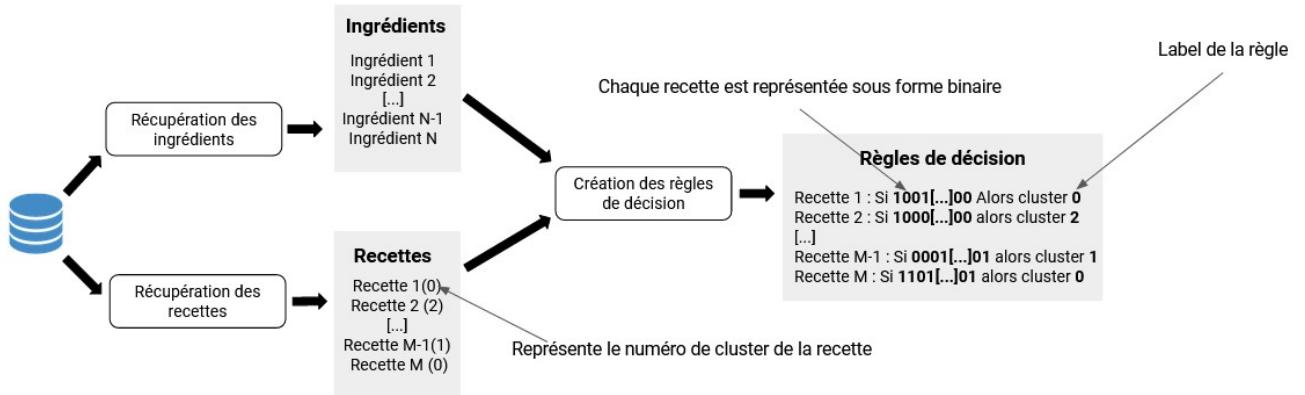


FIGURE 5.9 – Schéma de la construction des règles de décision

5.3.3 Évaluation de notre arbre de décision

Avant de générer notre arbre, il nous faut trouver la meilleure profondeur pour celui-ci. Pour cela nous utilisons une validation croisée K-Fold qui permet de trouver la précision moyenne de l'arbre pour une profondeur donnée.

Remarque : La solution que nous proposons a été influencée par celle proposée par le site web towardsdatascience.com (voir la source [Sim20] dans la bibliographie).

Nous allons décrire étape par étape l'algorithme d'évaluation ci-dessous (Algorithm 2). Tout d'abord, nous initialisons la liste *averageAccuracies* qui contiendra les précisions moyennes de l'arbre pour les différentes profondeurs. Ensuite, pour chaque profondeur comprise entre 3 et une profondeur maximale (*maxDepth*), nous allons :

- Initialiser l'arbre de décision avec la profondeur de la boucle. (*depth*)
- Initialiser le KFold avec le nombre de séparation demandé (*nbSplit*)
- Effectuer le processus de validation croisée K-Fold qui va mélanger le jeu de données au hasard et diviser celui-ci en *k* groupes.

- Pour chaque groupe, séparer notre jeu de données pour avoir d'un côté, des données pour entraîner notre arbre ($X_trainK, labels_trainK$) et de l'autre côté, des données pour tester l'arbre ($X_testK, labels_testK$). Puis, entraîner notre arbre avec chaque groupe et calculer la précision de la classification entre les étiquettes correctes ($labels_testK$) et les étiquettes prédictées ($predictedLabels$).
- Calculer la précision moyenne de l'arbre pour la profondeur donnée.

Algorithm 2: Évaluation de la profondeur de l'arbre

```

Input : maxDepth : int, X : List, labels : List, nbSplit : int, criterion : String, testSize : int
Output : averageAccuracies : List
Import sklearn : train_test_split, DecisionTreeClassifier, KFold, accuracy_score
Begin
    averageAccuracies ← [0] * maxDepth
    for depth between 3 and maxDepth :
        tree ← DecisionTreeClassifier(criterion=criterion, max_depth=depth, random_state=8)
        kfold ← KFold(nb_splits=nbSplit, shuffle=True, random_state=8)
        for train, test in kfold.split(X) :
            X_trainK, X_testK, labels_trainK, labels_testK ← X[train], X[test], labels[train], labels[test]
            tree.fit(X_trainK, labels_trainK)
            predictedLabels ← tree.predict(X_testK)
            accuracyScore ← accuracy_score(labels_testK, predictedLabels) * 100
            accuracy ← accuracy + ac_score
            averageAccuracy ← accuracy / nbSplit
            averageAccuracies.append(averageAccuracy)
    return averageAccuracies
End

```

FIGURE 5.10 – Algorithme de l'évaluation de l'arbre de décision

Pour effectuer cette évaluation, nous avons attribué comme valeur :

- $maxDepth = 19$ pour continuer l'évaluation jusqu'à cette profondeur maximale.
- $nbSplit = 10$ pour séparer en 10 groupes notre jeu de données à l'aide du KFold.
- $criterion = entropy$ pour mesurer la pureté des noeuds de notre arbre.

Sur la figure 5.11, se trouve le résultat de cette évaluation. La courbe bleue représente la précision moyenne de l'arbre. On constate que la précision de l'arbre augmente fortement entre la profondeur 3 et 11 et atteint une précision maximale d'environ 95%. Cependant, on constate une stagnation de la courbe à partir de la profondeur 11, c'est donc cette profondeur que l'on utilisera pour entraîner notre arbre. Suite à cela, nous sérialisons l'arbre pour pouvoir l'utiliser plus tard lors de l'appel d'une prédiction.

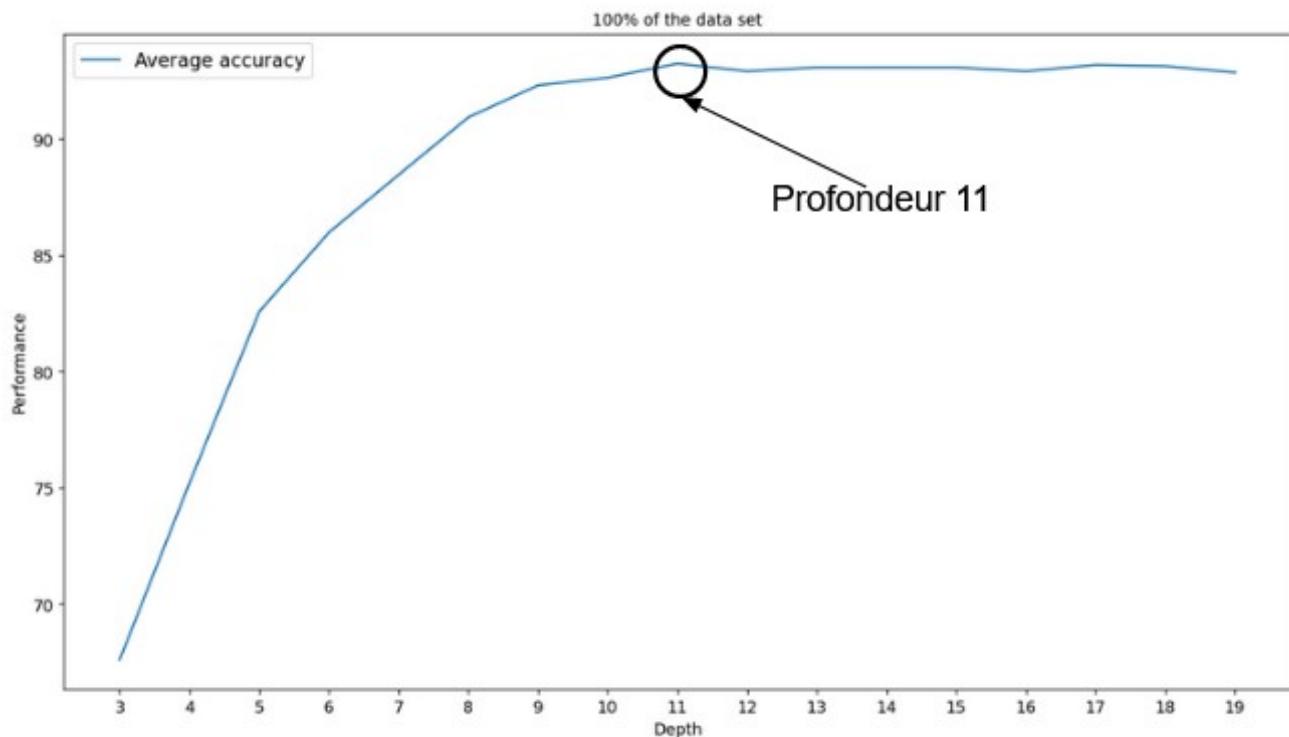


FIGURE 5.11 – Précision d'un arbre avec différentes profondeurs

5.3.4 Désavantages de l'arbre

Un des problèmes majeurs que nous avons avec la construction de l'arbre de décision concerne les ingrédients dits "universels" que l'on retrouve dans la plupart des recettes, quelle que soit leur catégorie, et donc, dans une grande partie de nos règles de décision. Ceci est problématique et peut impacter la classification de nos données.

Tout comme l'élaboration de notre K-Means Clustering, nous supprimons ces ingrédients avant d'en-trainer notre arbre. En effet, il n'y a peu d'intérêt à classifier nos données selon des assaisonnements tels que le sel et le poivre. Les indices de ces ingrédients seront donc mis à 0 pour les recettes les possédant. Cependant, la suppression de ces ingrédients est une décision d'expert qui ne relève plus de l'informatique. En effet, comment savoir qu'un ingrédient est intéressant ou non ? Par exemple, l'huile d'olive est un composant utilisé dans bon nombre de recettes, mais a-t-on le droit de le supprimer ? Certains utilisateurs préfèrent l'huile d'olive à d'autres huiles, et donc, supprimer celle-ci pourrait avoir un impact sur les requêtes des utilisateurs. Par conséquent, nous ne supprimons qu'une liste d'ingrédients "universelle" n'ayant peu de chances d'être demandée par un utilisateur. Cette décision a donc un impact sur la précision de notre arbre et sur les prédictions qu'il renvoie.

D'autre part, même si les prédictions de l'arbre sont très rapides et permettent d'améliorer le temps d'exécution d'une requête, elles diminuent la zone de recherche, et par conséquent, elles diminuent le nombre de recette envoyé. En effet, plusieurs recettes ne seront pas affichées à l'utilisateur puisqu'elles ne se situent pas dans le meilleur cluster.

5.3.5 Comparaison de performance entre l'arbre de décision et la classification naïve Bayésienne

Afin d'implémenter notre classification naïve Bayésienne, nous utilisons la classe **GaussianNB** de la librairie **sklearn**. Nous effectuons le même procédé que pour l'arbre de décision afin d'implémenter nos règles de décision. Pour évaluer le classificateur, nous utilisons une validation croisée K-Fold que nous réitérons 19 fois. Chaque K-Fold va calculer la précision moyenne du classificateur avec des jeux

de données aléatoires.

Sur la figure 5.12, nous constatons que la performance du classificateur naïf bayésien est très irrégulière et varie entre 38% et 44%. En revanche, pour l'arbre de décision, la précision moyenne est beaucoup plus élevée et tend vers une valeur proche de 95%. Ce faible seuil de performance peut être dû à l'hypothèse d'indépendance conditionnelle du classificateur naïf Bayésien. Cette hypothèse stipule que les caractéristiques sont indépendantes les unes des autres lorsqu'elles sont conditionnées par des étiquettes de classe. Or cette hypothèse est rarement exacte. En effet, les caractéristiques dépendent souvent les unes des autres, ce qui signifie qu'elles contiennent souvent des signaux similaires. Cependant, l'hypothèse d'indépendance conditionnelle du classificateur naïf Bayésien se traduit par son traitement des caractéristiques comme des signaux distincts.

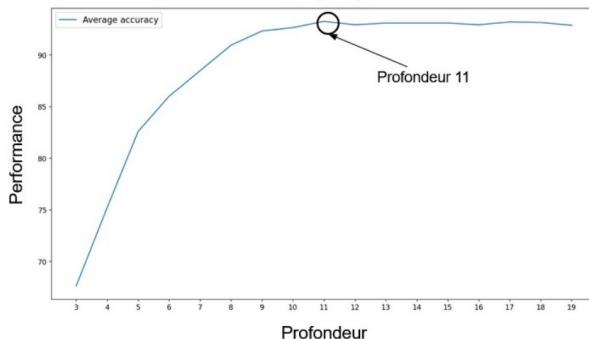


Schéma: Précision d'un arbre avec différentes profondeurs

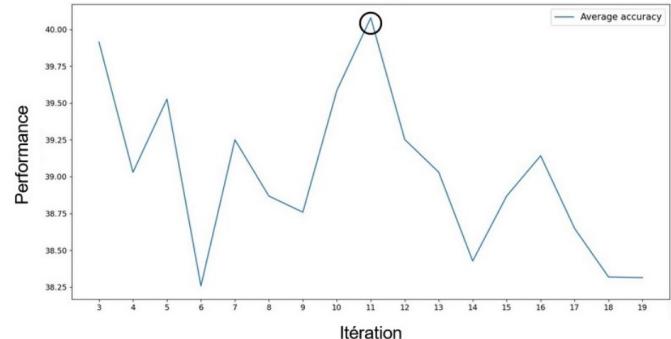


Schéma: Précision d'un Classificateur naïf bayésien avec plusieurs itérations

FIGURE 5.12 – Comparaison de performance des classificateurs

5.4 Tests et certifications de la solution

Ci-dessous (figure 5.13), se trouve un exemple d'arbre de décision avec une profondeur de 3. Le premier élément d'un noeud représente le nom de l'ingrédient, le deuxième représente la valeur de la pureté du noeud. On constate que pour l'ingrédient "crème fraîche", l'impureté du noeud est très élevé car cet ingrédient se retrouve dans toutes les classes/catégories. En revanche, pour l'ingrédient "oignons", l'impureté est beaucoup plus faible car l'ingrédient se trouve principalement dans le cluster 3. Le troisième élément correspond au nombre de recettes par cluster. Enfin, le dernier élément correspond au numéro de cluster possédant les recettes ayant le plus l'ingrédient. Par exemple, pour le "lait", il s'agit du cluster numéro 6.

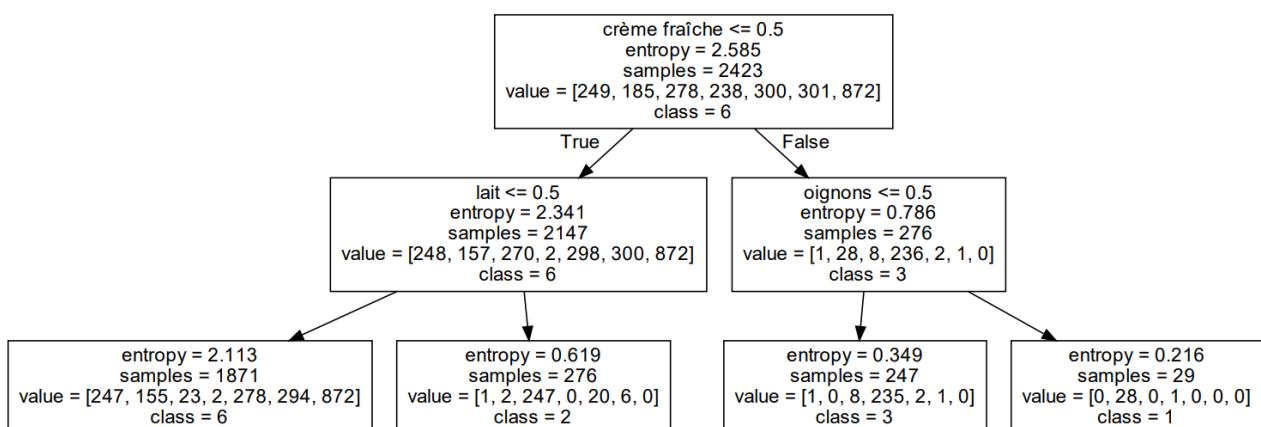


FIGURE 5.13 – Exemple d'un arbre de décision avec une profondeur de 3

Pour pouvoir utiliser notre arbre de décision, nous créons un service web SOAP Python avec le framework **Ladon**, ainsi qu'un client PHP qui appellera la méthode générant la prédiction de l'arbre. Lors de l'appel de la méthode, nous désérialisons l'arbre, nous effectuons la prédiction et nous renvoyons le résultat au client PHP.

Prenons comme exemple les ingrédient "crème fraîche", "oignons" et "sole" et appelons notre arbre de décision pour prédire le meilleur cluster contenant ces ingrédients. Selon le schéma de la figure 5.14, l'arbre doit nous renvoyer le cluster numéro 3. En vérifiant dans notre base de données le cluster possédant le plus ces ingrédients, on remarque qu'il s'agit bien du cluster numéro 3 (figure 5.15).

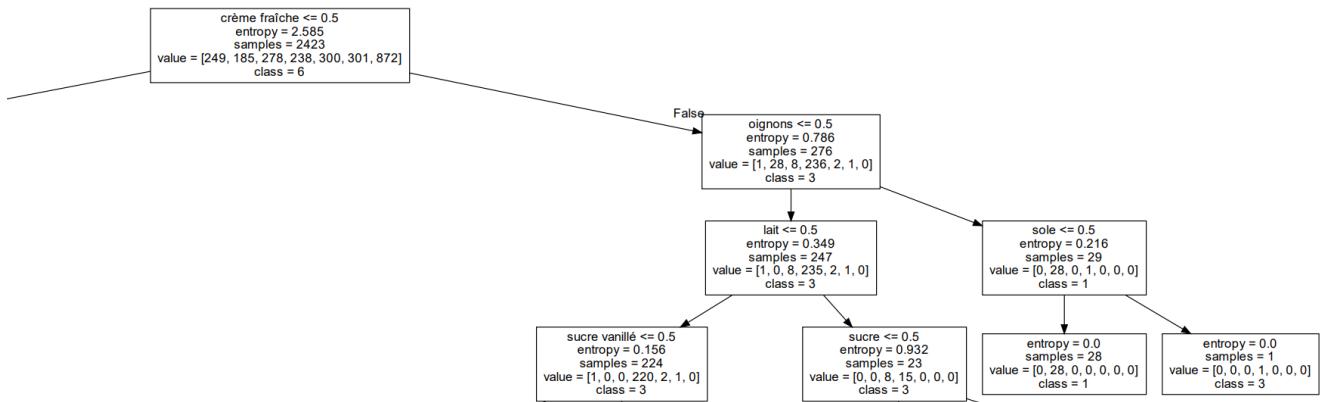


FIGURE 5.14 – Extrait de l'arbre de décision avec une profondeur de 12

numero_cluster	nombre_recette
0	23
1	225
2	21
3	241
4	21
5	13
6	70

FIGURE 5.15 – Nombre de recettes par cluster selon les ingrédients "crème fraîche" , "oignons" et "sole"

Nous avons bien comme résultat le numéro de cluster 3 pour les ingrédients demandés. On remarque sur la figure 5.17, que le temps d'exécution pour appeler la méthode SOAP et effectuer la prédiction ne dépasse pas les deux secondes.

Remarque : Si nous devions comparer le temps d'exécution de l'arbre avec le temps d'exécution de la requête SQL effectuant le même procédé, celle de la requête SQL serait beaucoup plus rapide. En effet, nous n'avons pas une grande quantité de données, et par conséquent, l'intérêt de notre arbre de décision se voit diminuer. Cependant, avec une quantité beaucoup plus conséquente de recettes, le temps d'exécution de la requête SQL augmentera fortement, à la différence de celui de l'arbre.

```
"Meilleur cluster pour les ingrédients 'crème fraîche', 'oignons' et 'sole' : 3 "
```

FIGURE 5.16 – Résultat de la prédiction

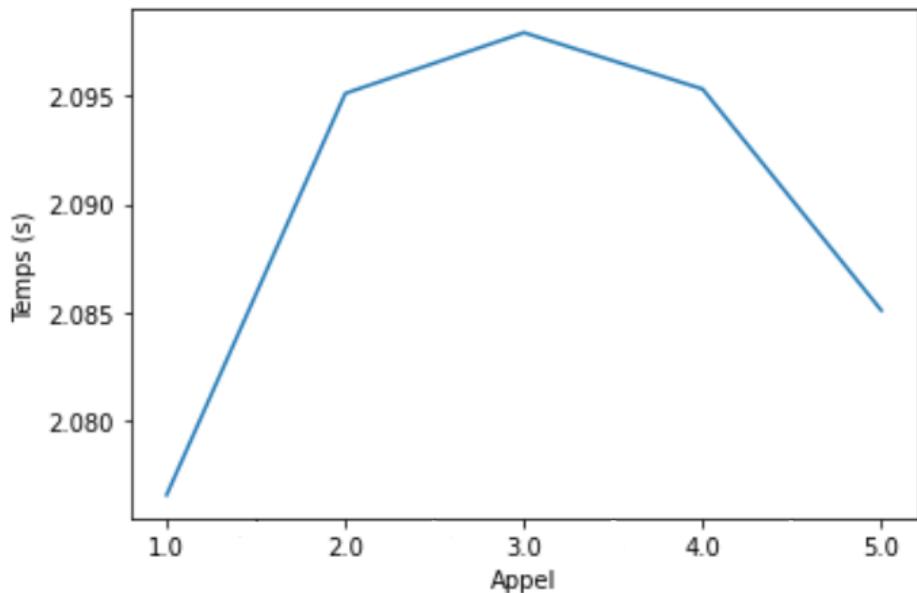


FIGURE 5.17 – Temps d'exécution des appels à l'arbre de décision

5.5 Ajout de nouvelles recettes dans la base de données

5.5.1 Prétraitement textuel des caractéristiques de la recette

Une des fonctionnalités supplémentaires proposées à l'utilisateur est de pouvoir ajouter une nouvelle recette dans notre base de données. Pour ce faire, l'utilisateur doit écrire les caractéristiques de cette recette et notamment les ingrédients de celle-ci. Or, ces ingrédients sont écrits à la main par l'utilisateur, nous devons donc effectuer un prétraitement textuel de ces ingrédients.

La première étape de ce processus est d'effectuer une tokenisation et une racinisation de ces ingrédients. Suite à cela, nous récupérons les ingrédients les plus similaires dans la base de données. Trois cas peuvent apparaître :

- (1) Plusieurs ingrédients similaires sont récupérés dans la base de données. Dans ce cas, nous prenons l'ingrédient dont le texte est le plus similaire à l'ingrédient recherché.
- (2) Un seul ingrédient similaire est récupéré dans la base de données. Nous récupérons donc cet ingrédient.
- (3) Aucun ingrédient similaire n'a été trouvé dans la base de données. Dans ce cas, l'ingrédient de la recette est un nouvel ingrédient et est stocké dans la base de données.

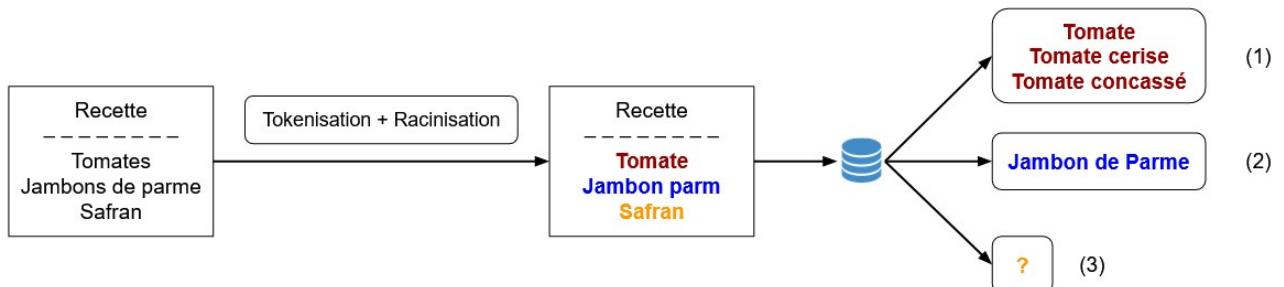


FIGURE 5.18 – Recherche des ingrédients similaires - 1

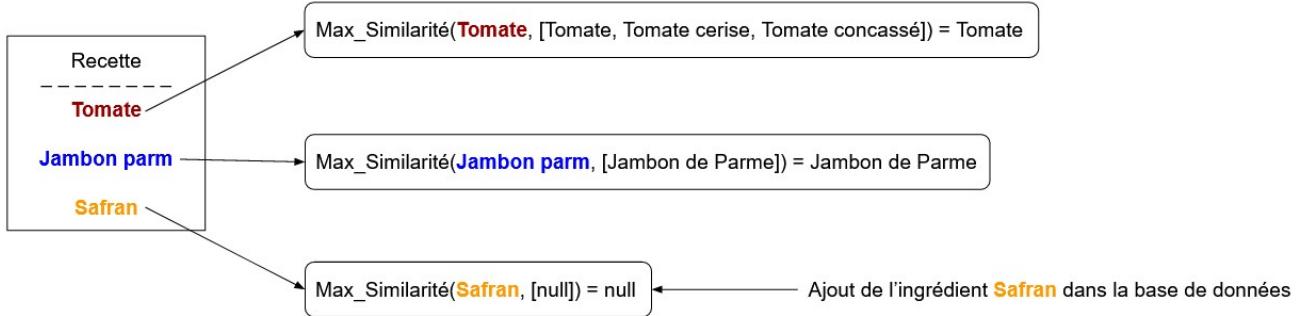


FIGURE 5.19 – Recherche des ingrédients similaires - 2

Une des problématiques de cette fonctionnalité est l'ajout de nouveaux ingrédients dans la base de données. En effet, notre partitionnement de données ainsi que notre classification de données sont basées sur les ingrédients de nos recettes. Par conséquent, l'ajout de nouveaux ingrédients pourrait avoir des conséquences négatives sur les prédictions de notre arbre de décision si celui-ci n'a pas été entraînée avec des règles de décisions possédant ces nouveaux ingrédients.

Une des solutions envisagées serait de reconstruire notre Kmeans Clustering et notre arbre de décision à chaque ajout de nouvelle recette. Cependant, avec un nombre conséquent d'utilisateurs sur notre application web, cette solution ne peut pas être envisageable (plusieurs ajouts de recettes en même temps peuvent être effectués).

La solution que nous avons donc choisie est de rendre "non actif" un nouvel ingrédient (valeur 0). Ces ingrédients non actifs ne seront pas pris en compte lors de l'appel de l'arbre de décision. Si la précision de notre arbre passe sous un certain seuil (environ 80%) en utilisant ces ingrédients non actifs, alors nous effectuerons une maintenance de l'application web qui nous permettra de reconstruire notre Kmeans Clustering et notre arbre de décision afin de rendre ces ingrédients actifs (valeur 1).

Une autre solution possible serait d'implémenter un système temps réel permettant d'appeler notre arbre de décision de manière automatisé afin de détecter quand les performances de l'arbre passent sous un certain seuil (pour plus de précision, voir la sous-section 9.2.3).

5.5.2 Recherche du cluster et des recettes voisines pour la nouvelle recette

Pour pouvoir stocker cette nouvelle recette dans notre base de données, nous devons lui attribuer un numéro de cluster. Nous appelons donc notre arbre de décision avec les ingrédients actifs de cette recette. L'arbre nous renverra ensuite le meilleur cluster où intégrer notre recette.

Remarque : Pour plus de précisions au niveau de l'utilisation de l'arbre de décision, veuillez vous référer à la sous-section 6.3.3.

Remarque : Si la recette à ajouter ne possède que des nouveaux ingrédients, alors lors de l'appel de l'arbre, aucun ingrédient ne sera envoyé et un cluster par défaut lui sera attribué par l'arbre.

Pour trouver les recettes voisines / proches de celle-ci, nous récupérons l'ensemble des recettes de ce cluster qui possèdent au moins un des ingrédients de la nouvelle recette.

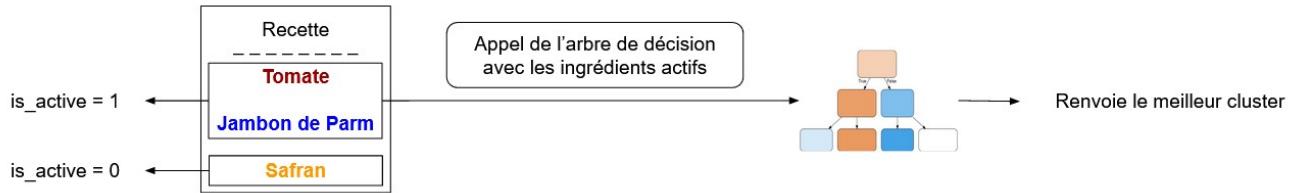


FIGURE 5.20 – Recherche du meilleur cluster

Chapitre 6

Algorithme de recommandation

Ce chapitre sera consacré à l'élaboration de notre algorithme de recommandation. Nous parlerons des différents algorithmes existants puis nous détaillerons l'ensemble des processus effectué pour construire celui-ci. Enfin, nous analyserons plusieurs résultats retournés par l'algorithme.

6.1 Analyse de la problématique

Maintenant que nous avons construit nos différents apprentissages, il nous faut procéder à la création de notre application web, et plus particulièrement, à notre algorithme de recommandation. Plusieurs systèmes de recommandation existent mais les deux modèles les plus traditionnels sont les algorithmes de filtrage collaboratif et les algorithmes de filtrage basés sur le contenu. Nous étudierons le pour et le contre entre ces deux algorithmes et nous choisirons celui qui se rapproche le plus de nos attentes, en adaptant le modèle pour qu'il soit conforme au contexte de notre projet. Le fonctionnement de notre système reposera sur les données des utilisateurs, les coordonnées de nos recettes (construit à l'aide du K-Means Clustering), mais aussi sur les prédictions de notre arbre de décision. Notre algorithme de suggestion utilisera ensuite ces données et effectuera un processus complexe pour construire les recettes à suggérer. Une conception orientée objet (PHP) sera utilisée pour construire notre modèle.

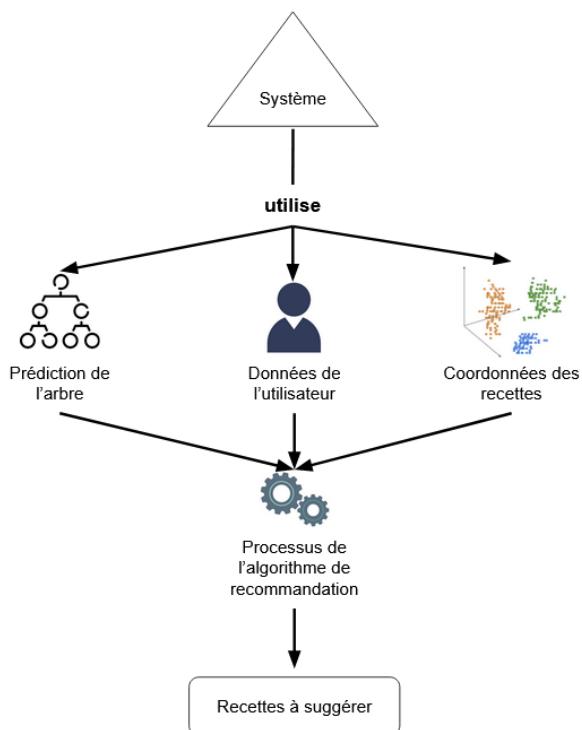


FIGURE 6.1 – Architecture globale de l'algorithme de recommandation

En plus de la suggestion de recettes, l'utilisateur à la possibilité de visualiser des recettes par rapport à des ingrédients mais aussi de choisir des ingrédients de préférence sur son profil. Etant donné que c'est l'utilisateur qui écrit les ingrédients, il se peut que l'on trouve des fautes d'orthographe. Il faut donc effectuer un prétraitement textuel de nos ingrédients avant de les envoyer à notre arbre de décision.

6.2 Etat de l'art : études des solutions existantes

Remarque : Les figures ci-dessous (figure 6.2 et 6.3) proviennent du site web **Google developers** (voir la source [Dev21] dans la bibliographie).

6.2.1 Filtrage basé sur le contenu

Il existe de nombreuses approches traditionnelles utilisées pour construire des systèmes de recommandation. Une approche courante est le filtrage basé sur le contenu. Il utilise les fonctionnalités des articles pour recommander d'autres articles similaires à ceux qu'un utilisateur aime en fonction de ses actions précédentes. Par exemple, nous illustrons ici quatre applications qui ont des fonctionnalités différentes. Chaque ligne représente une application et chaque colonne représente une fonctionnalité. Certaines applications sont éducatives ou scientifiques, d'autres concernent la santé. Lorsqu'un utilisateur installe une application de santé, nous pouvons lui recommander d'autres applications liées à la santé, car elles sont similaires aux applications installées.

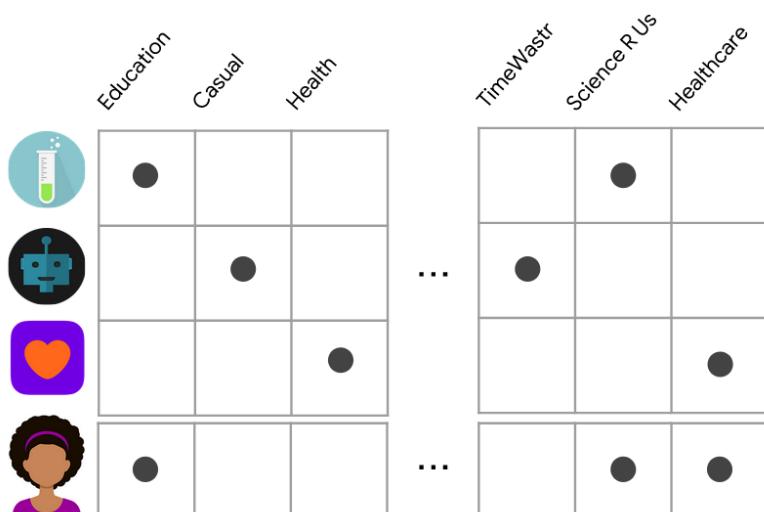


FIGURE 6.2 – Exemple de cas d'utilisation du filtrage basé sur le contenu

Nous pouvons utiliser une matrice binaire pour les méthodes basées sur le contenu. Cette matrice peut aider à indiquer la préférence de l'utilisateur pour certains éléments. Avec les données recueillies auprès de l'utilisateur, nous pouvons trouver une relation entre les éléments qui plaisent à l'utilisateur et ceux qui ne le sont pas. Nous attribuons une valeur particulière à chaque paire utilisateur-élément, cette valeur est connue sous le nom de degré de préférence.

L'approche basée sur le contenu présente de nombreux avantages :

- elle ne repose pas sur les informations des autres utilisateurs et prend en compte seulement ce qui est relatif au profil de l'utilisateur ciblé.
- elle souffre beaucoup moins du problème du démarrage à froid par rapport aux approches collaboratives, car les nouveaux éléments peuvent être décrits par leurs caractéristiques, c'est-à-dire leur contenu, et ainsi, des suggestions pertinentes peuvent être faites pour ces nouvelles entités.

- il est possible de recommander de nouveaux éléments ou même des éléments qui ne sont pas populaires ou qui viennent d'être ajouté.

Cependant, cette approche a aussi des inconvénients :

- les utilisateurs ayant visualisé un très grand nombre d'éléments posent un problème (trop d'informations dans le profil de l'utilisateur à faire coïncider avec les caractéristiques des éléments).
- l'approche peut également donner des suggestions très peu diversifiées car nous ne recommandons jamais d'article en dehors des préférences de l'utilisateur.
- les profils des utilisateurs restent difficiles à élaborer et, qui plus est, il faut prendre en compte l'évolution des intérêts de l'utilisateur.

6.2.2 Filtrage collaboratif

Une autre approche courante est le filtrage collaboratif. L'une des limites du filtrage basé sur le contenu est qu'il ne tire parti que des similitudes entre les éléments. L'intérêt du filtrage collaboratif est de pouvoir utiliser simultanément les similitudes entre les utilisateurs et les éléments pour fournir des recommandations. Cela permet de recommander un élément à l'utilisateur A en fonction des intérêts d'un utilisateur similaire B.

Nous illustrons ici une matrice de rétroaction de quatre utilisateurs et de cinq films. Chaque ligne représente un utilisateur et chaque colonne représente un film. La coche verte signifie qu'un utilisateur a regardé un film particulier. Nous considérons cela comme une rétroaction implicite. Si un utilisateur donne une note sur un film, ce serait un retour explicite. Comme vous pouvez le voir sur la figure 6.3, l'utilisateur de la première ligne a regardé les trois films "Harry Potter", "Shrek" et "The Dark Knight Rises". L'utilisateur dans la troisième rangée, a également regardé les films "Harry Potter" et "Shrek". Il peut donc être judicieux de lui recommander le film "The Dark Knight Rises" puisque le premier utilisateur a les mêmes préférences qu'elle.

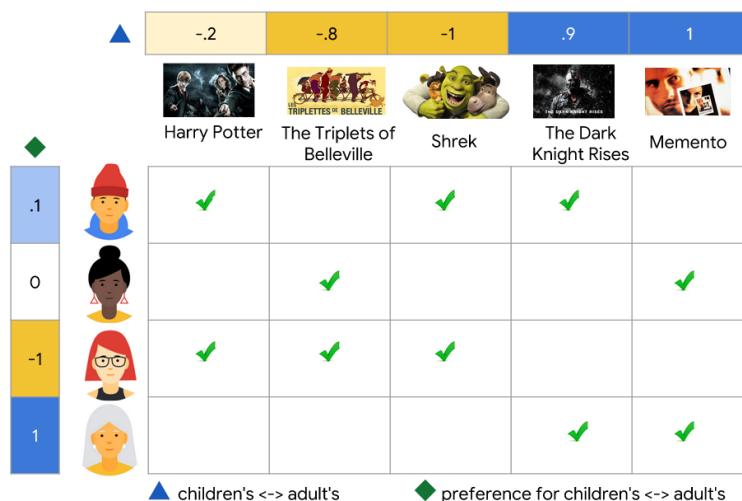


FIGURE 6.3 – Exemple de cas d'utilisation du filtrage collaboratif

Le filtrage collaboratif présente de nombreux avantages :

- il peut aider les utilisateurs à découvrir de nouveaux intérêts.
- plus il y a d'utilisateurs, plus il y a de scores, et meilleurs sont alors les résultats.

Cependant, tout comme le filtrage basé sur le contenu, il existe des inconvénients :

- il faut un nombre assez conséquent d'utilisateurs pour pouvoir recommander des éléments et trouver des groupes d'utilisateurs similaires.

- il existe aussi le problème du démarrage à froid : lorsqu'un nouvel élément est ajouté au catalogue, personne ne lui a attribué de score. Or, il est difficile de fournir de bonnes recommandations avant qu'un nombre suffisant d'utilisateurs aient saisi des avis.

6.3 Solution proposée et sa mise en œuvre

Après avoir étudié le fonctionnement du système de recommandation et les différences entre les modèles de filtrage collaboratif et de filtrage basé sur le contenu, nous avons décidé de choisir celui basé sur le contenu. Étant donné que nous devons créer une application web, un problème majeur fait surface : la création des utilisateurs. En effet, pour pouvoir utiliser l'algorithme de filtrage collaboratif, nous avons besoin d'utilisateurs, et plus précisément, d'évaluations de recettes. Cependant, la création de ces utilisateurs est un procédé complexe qui nécessite de générer des profils virtuels ayant des préférences aléatoires. Ceci est assez dangereux puisque la création de notes aléatoires pourrait avoir un réel impact négatif au niveau de notre algorithme. Le filtrage basé sur le contenu règle donc ce problème puisque nous ne prenons que ce qui est relatif à l'utilisateur ciblé et non l'ensemble des utilisateurs de notre application. Ce filtrage résout aussi le problème des évaluations de recettes.

6.3.1 Calcul des recettes proches

Rappel : Chacune des recettes de notre base de données possède des coordonnées qui ont été calculé à l'aide de notre partitionnement de données via un K-Means clustering. Pour plus de détail, voir section 4.3.3.

Avant de rentrer dans les détails de notre algorithme de recommandation, nous devons écrire un script PHP qui va calculer l'ensemble des recettes proches de chaque recette de notre base de données. La génération de ces recettes proches s'effectue en plusieurs étapes. Pour chaque recette de la base de données nous :

- calculons la distance euclidienne de la recette ciblée avec toutes les autres recettes à l'aide de leurs coordonnées.
- recherchons les recettes dont les distances avec la recette ciblée sont plus petites que la moyenne des distances.
- trions et insérons l'ensemble des recettes proches de la recette ciblée dans l'attribut *close_to* de notre table "Recipe"

Ce processus étant très couteux en temps d'exécution, nous avons décidé de l'effectuer avant la création de notre algorithme de recommandation.

6.3.2 Diagramme de classes de l'algorithme de recommandation

Pour une meilleure visualisation du diagramme de classes, la couche "front" et "business" ont été simplifiées. Nous ne présentons que les classes ayant un rapport avec l'algorithme de recommandation. De plus, par manque de clarté, l'ensemble des méthodes ne sont pas affichées dans les classes.

Notre programme est composé de plusieurs couches qui simplifie la lecture du code et sa maintenabilité :

- La couche model qui contient les données / modèles manipulées par la base données mais aussi par l'algorithme de recommandation.
- La couche process effectuant le processus de construction des recettes à suggérer.
- la couche facade qui effectue le lien entre la couche métier et le front de l'application web.
- la couche database qui effectue l'ensemble des requêtes à notre base de données
- la couche du web service SOAP qui appelle la méthode python effectuant la prédiction de l'arbre

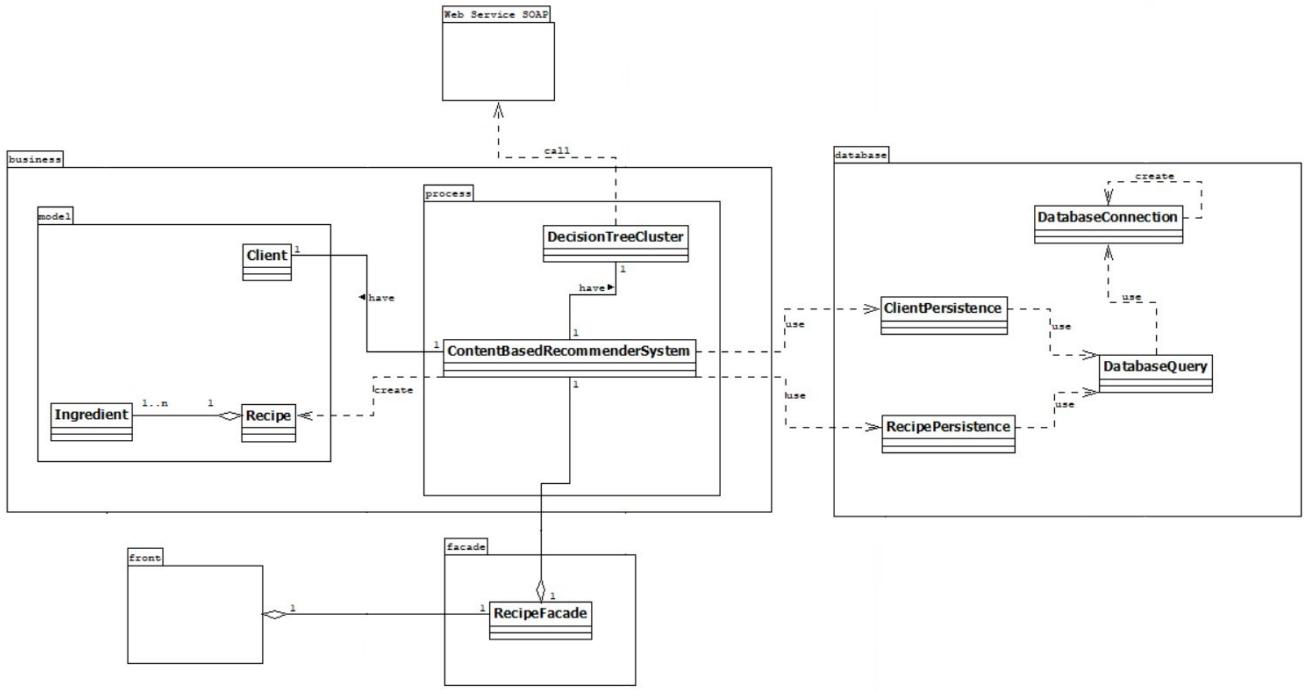


FIGURE 6.4 – Diagramme des classes de l’algorithme de recommandation

6.3.3 Post-traitement textuel des ingrédients

Pour mieux comprendre l'utilisation de notre arbre de décision dans notre algorithme de recommandation, nous allons présenter un exemple simple d'utilisation (figure 6.5).

Comme nous l'avons expliqué dans la section précédente, l'utilisateur peut choisir des ingrédients de préférence sur son profil. Cependant, avant d'appeler notre arbre de décision, il faut effectuer un traitement textuel des ingrédients et effectuer une récupération d'informations.

Imaginons que l'utilisateur aime l'ingrédient « filets de poulet ». Étant donné que c'est l'utilisateur qui écrit l'ingrédient, il se peut qu'il y ait des fautes d'orthographe. Donc la première étape va être de faire de la tokenization pour séparer les différents mots, puis de la suppression des mots vides (ou stopwords) (dans l'exemple ci-dessous, le « de » disparaît). Enfin, nous effectuons de la racinisation (ou stemming) pour transformer les mots aux pluriels aux singuliers, et corriger certains mots (le mot "pouletts" devient "poulet"). Suite à cela, on a donc les mots "filet" et "poulet" et l'on va récupérer dans la base de données, l'ensemble des ingrédients qui possède ces deux mots.

Maintenant que nous avons nos ingrédients normalisés, on va pouvoir appeler notre arbre de décision à l'aide du web service SOAP. L'arbre nous renverra ensuite le cluster possédant le plus de recettes ayant ces ingrédients, et il restera à récupérer ces recettes du cluster 0 dans la base de données.

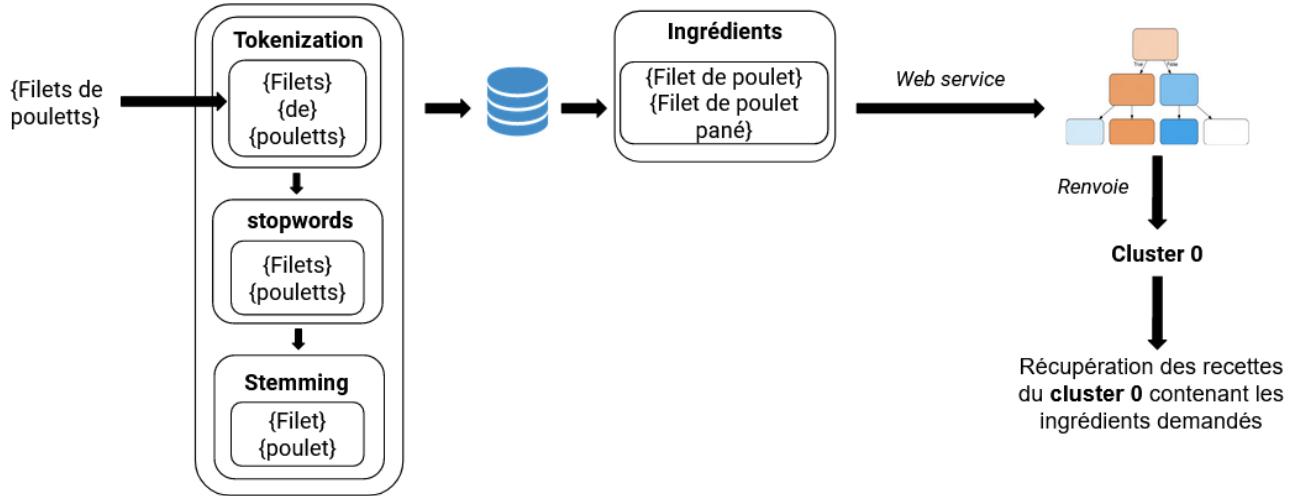


FIGURE 6.5 – Post-traitement des ingrédients et appel de l’arbre de décision

6.3.4 Récupération des données de l’utilisateur

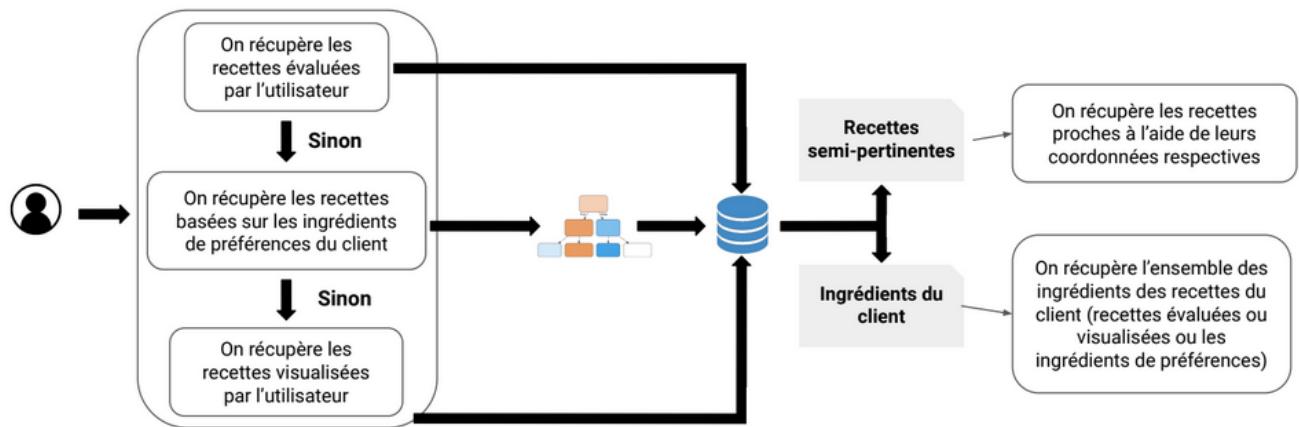


FIGURE 6.6 – Récupération des données nécessaires à l’algorithme de recommandation

Avant d'effectuer le processus de recommandation, il nous faut récupérer les données de l'utilisateur :

- Si l'utilisateur a évalué positivement des recettes alors on récupère celles-ci.
- Sinon, s'il n'y a aucune évaluation, on prend les ingrédients de préférence de l'utilisateur (c'est-à-dire les ingrédients qu'il a choisis sur son profil). On appelle notre arbre de décision pour avoir le meilleur cluster possédant ces ingrédients et on récupère les recettes contenant ces ingrédients dans ce cluster.
- Sinon, s'il n'y a aucun ingrédient de préférence, on récupère les recettes que l'utilisateur a visualisé durant sa session.
- Si aucune de ces demandes n'est possible, l'algorithme de recommandation ne peut être effectué, et donc, on génère des recettes aléatoires.

Remarque : Nous ne prenons que les 50 recettes les plus récentes évaluées par l'utilisateur. Le fait de définir un seuil permet de ne pas être surchargé de données et donc de ne pas régresser le temps d'exécution de notre algorithme de recommandation. De plus, nous sommes parti du principe que les goûts d'un utilisateur varient avec le temps, il n'est donc pas utile de récupérer toutes les recettes évaluées par celui-ci.

Grâce à ces données de l'utilisateur, on va pouvoir récupérer d'une part, les recettes dites "semi-

pertinentes", c'est-à-dire les recettes proches de celles de l'utilisateur. Grâce aux coordonnées de celles-ci et à l'attribut "close_to" de notre table RECIPE, il est possible de récupérer ces recettes proches très facilement et très rapidement. En effet, celles-ci ont été stockées dans notre base de données lors de notre apprentissage non supervisé via le K-Means Clustering.

Suite à cela, nous créons la matrice des recettes semi-pertinentes. Le procédé est le même que pour l'arbre de décision et le K-Means clustering puisqu'il s'agit d'une matrice binaire. Pour chaque ingrédient i des données de l'utilisateur, on vérifie si la recette semi-pertinente la possède, si oui on ajoute 1 à l'index i , sinon on ajoute 0.

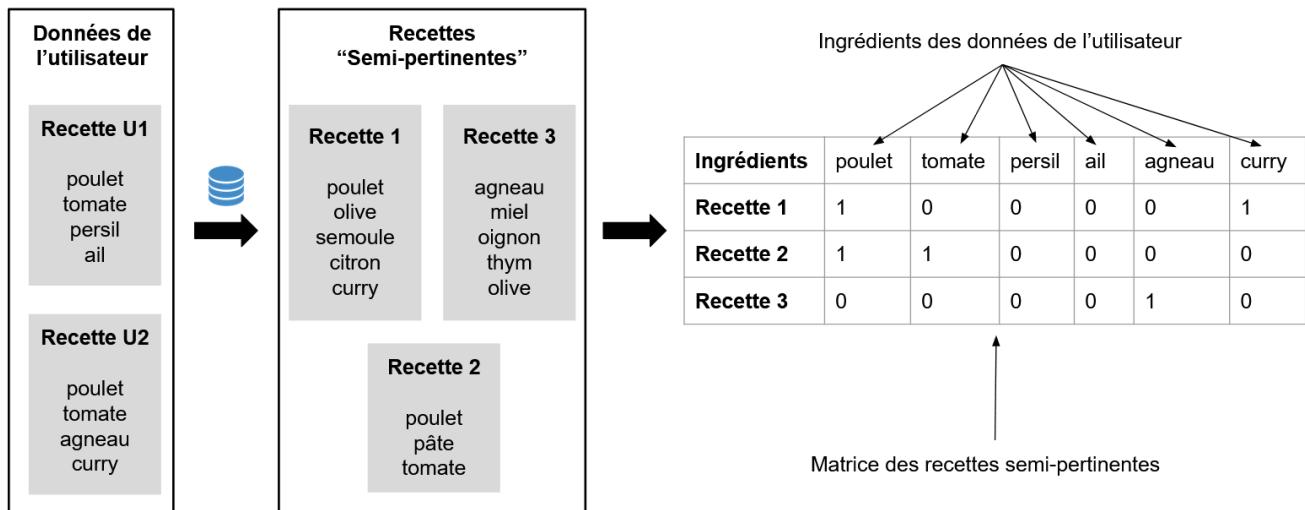


FIGURE 6.7 – Création de la matrice des recettes semi-pertinentes

D'autre part, nous récupérons l'ensemble des ingrédients des données de l'utilisateur et nous les indexons. Puis, nous construisons le vecteur de l'utilisateur avec le pourcentage d'apparition de ces ingrédients.

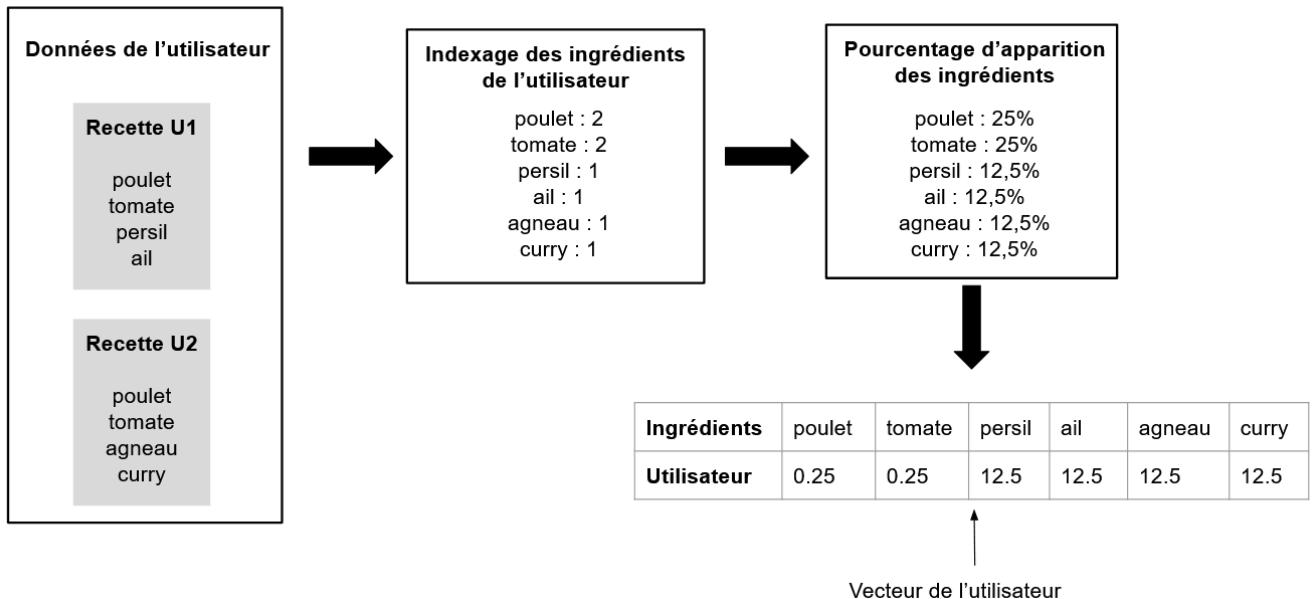


FIGURE 6.8 – Création du vecteur de l'utilisateur

6.3.5 Calcul vectoriel et matriciel

Maintenant que nous possédons les données nécessaires, nous pouvons commencer l'élaboration de notre algorithme de recommandation. La deuxième étape va être d'effectuer les calculs vectoriels entre le vecteur de l'utilisateur et la matrice des recettes semi-pertinentes. Nous allons donc calculer la similarité entre chaque vecteur de recette et le vecteur de l'utilisateur. Nous utilisons le **cosinus similarité** qui mesure la similarité en utilisant le cosinus de l'angle entre deux vecteurs dans un espace multidimensionnel et détermine s'ils pointent à peu près dans la même direction. Plus le score est proche de 1, plus il y a une grande similarité entre les deux vecteurs.

Remarque : La solution proposée ci-dessous est basée sur celle proposée par l'université de Stanford (voir la source [Uni21] dans la bibliographie).

$$u(\mathbf{x}, \mathbf{i}) = \cos(\mathbf{x}, \mathbf{i}) = \frac{\mathbf{x} \cdot \mathbf{i}}{||\mathbf{x}|| \cdot ||\mathbf{i}||}$$

FIGURE 6.9 – Formule du cosinus similarité

Nous effectuons le calcul de similarité pour chaque recette semi-pertinentes puis nous calculons la moyenne des scores. Les recettes qui ont un score plus grand ou égal à la moyenne des scores seront les recettes à suggérer à l'utilisateur.

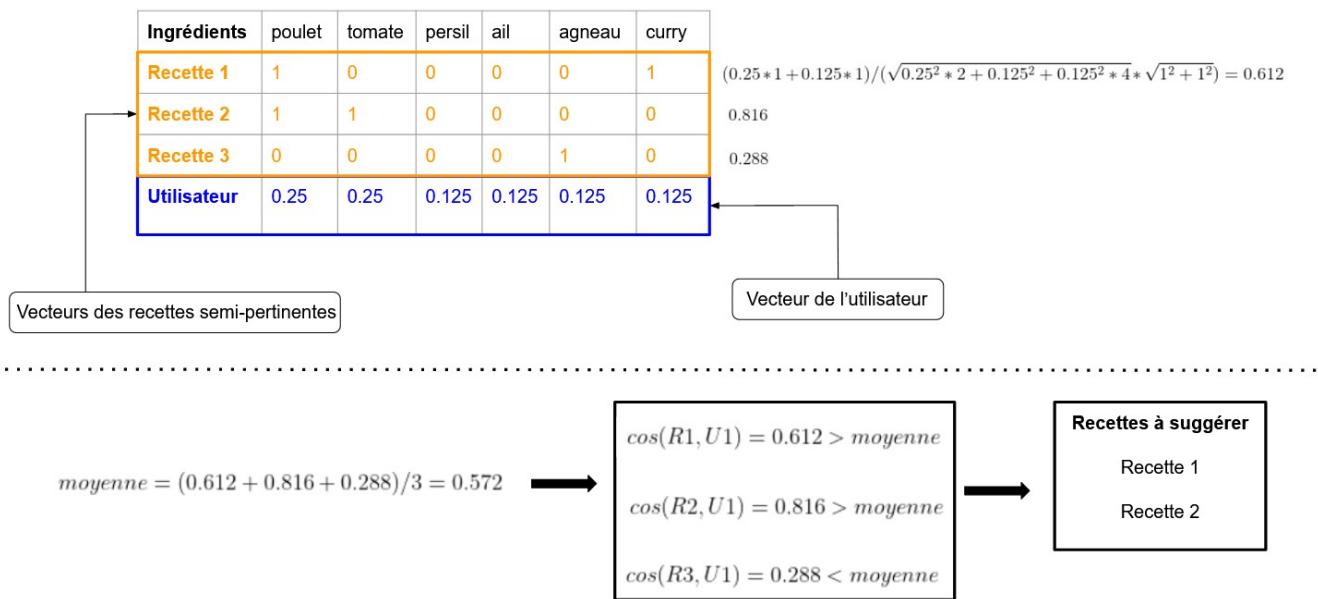


FIGURE 6.10 – Calcul des recettes pertinentes

6.4 Tests et certifications de la solution

Afin de tester notre algorithme de recommandation, nous créons un utilisateur auquel nous faisons évaluer positivement deux recettes aléatoires. Étant donné le nombre conséquent de recettes à suggérer, nous n'afficherons que les trois meilleures, c'est-à-dire celles qui ont les scores de similarité les plus élevés. Dans la figure 6.11, nous constatons que les recettes à suggérer ont une similarité plus ou moins importante avec celles évalué par l'utilisateur. Plus il y a d'ingrédients similaire à ceux des recettes de l'utilisateur, plus le score augmente. On peut voir que les deux premières recettes à suggérer ont un score équivalent puisqu'elles ont chacune le même nombre d'ingrédients similaires.

Nous ne prenons pas en compte les ingrédients "sucre en poudre", "sucre", "farine", "levure", "oeufs" et "beurre" car ce sont des ingrédients universels qui apparaissent dans de nombreuses recettes. De plus, il existe peu d'intérêt à suggérer des recettes selon ces ingrédients.



FIGURE 6.11 – Extrait des recettes à suggérer pour un utilisateur

Comme nous l'avons expliqué dans les sections précédentes, lorsque l'utilisateur n'a évalué aucune recette, nous prenons comme données d'entrée, ses ingrédients de préférence. Étant donné que c'est l'utilisateur lui-même qui écrit ses ingrédients préférés, nous devons effectuer un prétraitement textuel. Pour tester notre solution, nous donnons au client précédemment créé, les ingrédients de préférence : "patates", "tomates" et "haricots". Nous effectuons donc la tokenization, la suppression des mots vides (ou stopwords), la racinisation (ou stemming) et la récupération des ingrédients dans la base de données. On constate sur la figure 6.12, qu'un nombre assez conséquent d'ingrédients a été récupéré suite au post-traitement. Malheureusement, nous n'en trouvons aucun faisant référence à "haricots". En effet, le processus de réduction (stemming) utilisé dans notre programme n'est pas complètement opérationnel et ne permet pas de corriger les fautes d'orthographe importantes.

id_ingredient	name	id_client
222	patates douces	1
224	double concentré de tomates	1
229	purée de tomate en bouteille	1
274	fondu de tomate	1
453	tomates ananas	1
478	patate douce	1
533	tomates séchées	1
590	tomate séchée à l'huile d'olive	1
666	pulpe de tomate	1
979	tomates concassées	1
980	sauce tomate	1
981	purée de tomate	1
993	tomates pelées	1
995	tomate séchée	1
1015	coulis de tomate	1
1043	concentré de tomates	1
1080	tomates cerise	1
1168	tomates confites à l'huile	1
1215	tomate	1

FIGURE 6.12 – Récupération des ingrédients de préférence de l'utilisateur

6.5 Solution alternative implémentée : Algorithme de filtrage collaboratif User-User

Pour pouvoir effectuer une analyse plus approfondie du temps d'exécution de notre algorithme de recommandation, nous avons construit un algorithme de filtrage collaboratif User-User et avons comparé leurs temps d'exécution. Cet algorithme ne suggère plus des recettes par rapport aux préférences de notre utilisateur ciblé mais par rapport aux préférences similaires de l'ensemble des utilisateurs avec cet utilisateur ciblé. Au lieu d'utiliser le contenu des recettes pour déterminer ce qu'il faut recommander, nous recherchons des utilisateurs similaires et recommandons des recettes qu'ils aiment.

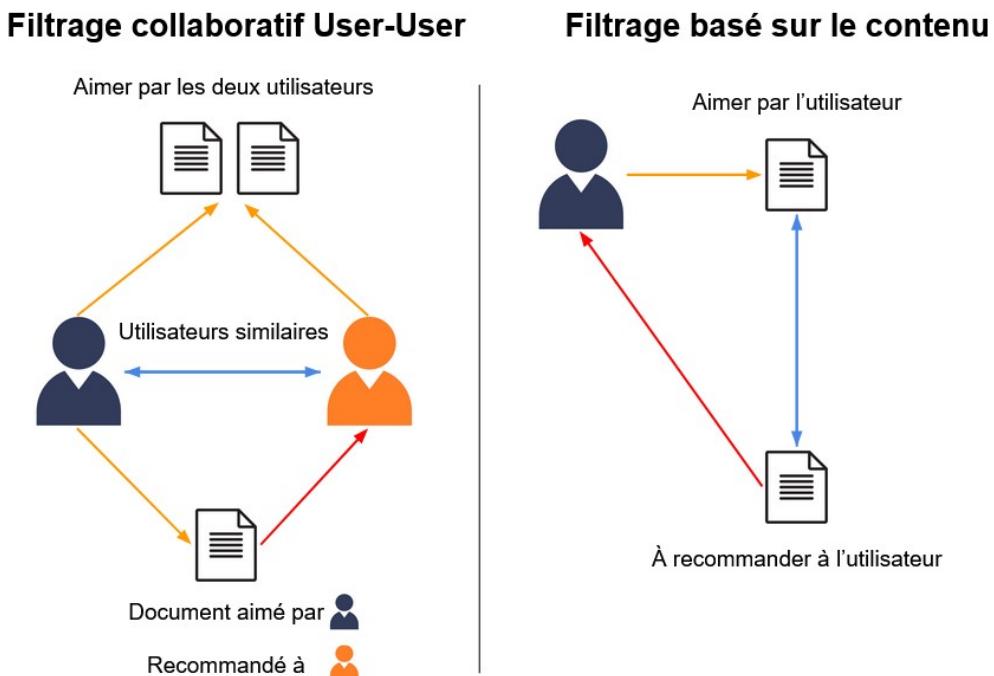


FIGURE 6.13 – Comparaison des algorithmes de recommandation

6.5.1 Crédation des clients

L'algorithme de filtrage collaboratif n'étant plus basé sur les contenus des recettes mais sur les évaluations des utilisateurs, nous devons générer aléatoirement des clients avec des préférences préconçues, et leur faire effectuer des évaluations de recettes. Pour ce faire, chaque client générera se verra attribuer aléatoirement 200 ingrédients préférés et 100 ingrédients détestés. Ensuite, nous prenons une recette aléatoire et, selon les ingrédients de celle-ci, nous lui attribuons une note. Si la recette contient un ingrédient préféré, nous incrémentons le score de l'évaluation. Au contraire, si la recette contient un ingrédient détesté, nous décrémentons la note. Nous générerons plus de 1000 clients et nous effectuons ce processus d'évaluations 50 fois par client.

6.5.2 Crédation de la matrice d'utilité

Les données utilisées dans notre système de recommandation sont divisées en deux catégories : les utilisateurs et les recettes. Chaque utilisateur aime certaines recettes, et la valeur d'évaluation r_{ij} (de 1 à 5) est la donnée associée à chaque utilisateur i et recette j et représente à quel point l'utilisateur apprécie la recette. Ces valeurs de notation sont collectées dans une matrice, appelée matrice d'utilité R, dans laquelle chaque ligne i représente la liste des recettes notées pour l'utilisateur i tandis que chaque colonne j répertorie tous les utilisateurs qui ont noté la recette j .

Remarque : La solution proposée ci-dessous est basée sur celle proposée par l'université de Stanford (voir la source [Uni21] dans la bibliographie).

Pour créer cette matrice, nous allons récupérer l'ensemble des utilisateurs "semi-similaires" qui ont, au minimum, évalué une recette que l'utilisateur ciblé à évalué. Dans les colonnes de cette matrice, nous aurons donc l'ensemble des recettes évaluées par ces utilisateurs semi-similaires.

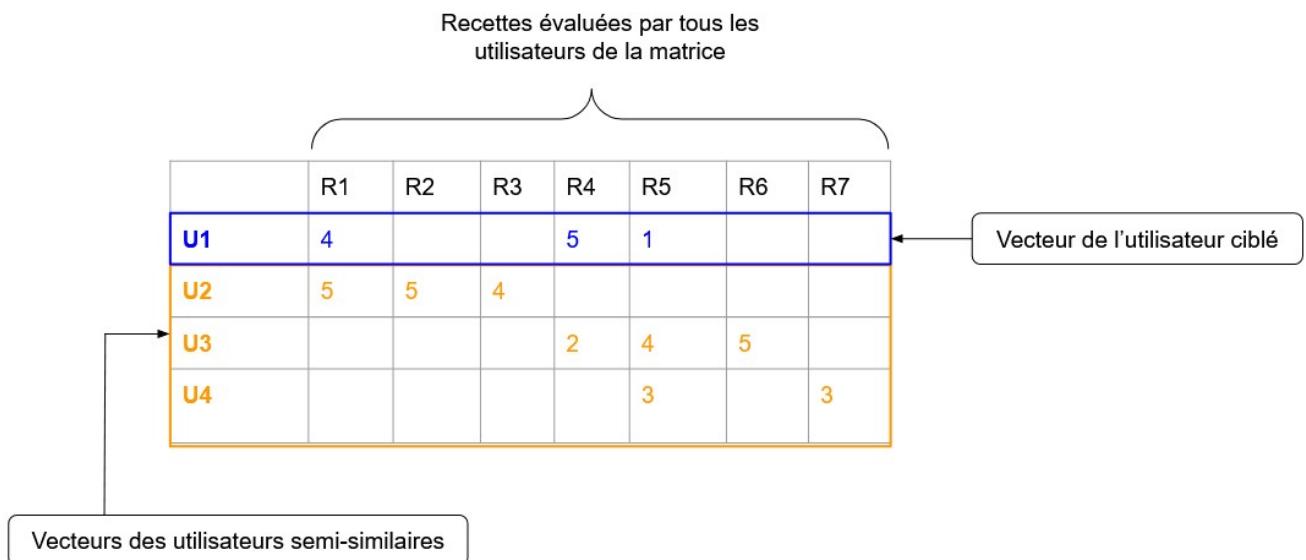


FIGURE 6.14 – Crédation de la matrice d'utilité

Rappel : Soit r_x , le vecteur des notes de l'utilisateur x et r_y le vecteur des notes de l'utilisateur y . Alors :

$$sim(x, y) = \cos(r_x, r_y) = \frac{r_x \cdot r_y}{\|r_x\| \|r_y\|}$$

FIGURE 6.15 – Cosinus similarité

En considérant la matrice d'utilité de la figure 6.14, nous aimerions intuitivement que U_1 ressemble beaucoup plus à U_2 et soit très différent de U_3 car U_1 et U_3 ont des opinions exactement opposées. Une des solutions possible pour traiter cela est le cosinus similarité. Malheureusement, le cosinus similarité ne capture pas vraiment à quel point U_1 est opposé à U_3 . De plus, les opinions opposées ne sont pas considérées comme des notes négatives et le cosinus ne traite pas correctement un manque de note comme non informative. Nous aimerions donc que la similitude de (U_1, U_2) soit beaucoup plus grande que la similitude de (U_1, U_3) . Cependant, comme vous pouvez le constater sur la figure 6.16, les similitudes sont très similaires.

$$\cos(U_1, U_2) = \frac{4 * 5}{\sqrt{4^2 + 5^2 + 1^2} * \sqrt{5^2 + 5^2 + 4^2}} = 0.380$$

$$\cos(U_1, U_3) = \frac{5 * 2 + 1 * 4}{\sqrt{4^2 + 5^2 + 1^2} * \sqrt{2^2 + 4^2 + 5^2}} = 0.322$$

FIGURE 6.16 – Calcul du cosinus similarité entre (U_1, U_2) et (U_1, U_3)

Une des solutions pour résoudre cette problématique est de transformer notre matrice d'utilité en matrice d'utilité centrée sur la moyenne. Cela consiste à centrer les utilisateurs sur leur moyenne puis à soustraire les moyennes de chaque ligne. En d'autres termes, pour chaque ligne, nous calculons sa moyenne, puis nous la soustrayons à chaque élément de l'utilisateur. Cependant, nous ne soustrayons pas les valeurs NULL et nous les laissons à zéro à la fin.



	R1	R2	R3	R4	R5	R6	R7
U1	4			5	1		
U2	5	5	4				
U3				2	4	5	
U4					3		3

$\leftarrow \text{mean}(U_1) = \frac{4+5+1}{3} = \frac{10}{3}$
 $\leftarrow \text{mean}(U_2) = \frac{11}{3}$
 $\leftarrow \text{mean}(U_3) = \frac{11}{3}$
 $\leftarrow \text{mean}(U_4) = \frac{6}{2}$

	R1	R2	R3	R4	R5	R6	R7
U1	$2/3$ (*)			$5/3$	$-7/3$		
U2	$1/3$	$1/3$	$-2/3$				
U3				$-5/3$	$1/3$	$4/3$	
U4					0		0

$(*) (U_1, R1) = 4 - \text{mean}(U_1) = 4 - \frac{10}{3} = \frac{2}{3}$

FIGURE 6.17 – Création de la matrice d'utilité centrée sur la moyenne

En effectuant ce processus, on constate maintenant, qu'un zéro signifie aucune information. Ces zéros peuvent se produire lorsque nous n'avons pas du tout d'évaluation ou parce que l'évaluateur n'est pas informatif (c'est-à-dire lorsque l'utilisateur évalue avec la même valeur toutes les recettes).

Maintenant, si nous calculons le cosinus dans la matrice d'utilité centrée sur la moyenne, nous remarquons que $U1$ et $U2$ ont des valeurs opposées et que leurs vecteurs sont très différents.

$$\cos(U1, U2) = \frac{2/3 * 1/3}{\sqrt{(2/3^2) + (5/3)^2 + (-7/3)^2} * \sqrt{(1/3)^2 + (1/3)^2 + (-2/3)^2}} = 0.092$$

$$\cos(U1, U3) = \frac{5/3 * (-5/3) + (-7/3) * 1/3}{\sqrt{(2/3^2) + (5/3)^2 + (-7/3)^2} * \sqrt{(-5/3)^2 + (1/3)^2 + (4/3)^2}} = -0.559$$

FIGURE 6.18 – Calcul du cosinus similarité centré sur la moyenne entre ($U1, U2$) et ($U1, U3$)

Pour pouvoir trouver les utilisateurs les plus proches de l'utilisateur ciblé $U1$, nous effectuons le calcul de similarité entre $U1$ et chacun des autres utilisateurs semi-similaires. Suite à cela, nous récupérons les utilisateurs dont le score est plus grand ou égal à la moyenne des similarités.

6.5.3 Calcul des prédictions des notes

Maintenant que nous avons notre matrice d'utilité et nos utilisateurs similaires, nous pouvons voir comment faire des recommandations pour une recette non notée i de l'utilisateur ciblé.

Soit :

- r_x le vecteur des notes de l'utilisateur x
- N l'ensemble des k utilisateurs les plus similaires à x
- S_{xy} le score de similarité entre l'utilisateur ciblé x et l'utilisateur similaire y

La prédiction pour la recette i de l'utilisateur ciblé x se traduit par la formule suivante :

$$r_{xi} = \frac{\sum_{y \in N} S_{xy} * r_{yi}}{\sum_{y \in N} S_{xy}}$$

FIGURE 6.19 –

Supposons que nous avons la matrice d'utilité suivante (figure 6.20) dans laquelle l'utilisateur ciblé $U1$ est similaire aux utilisateurs $U2$ et $U6$. Nous voulons trouver l'évaluation la plus probable de l'utilisateur $U1$ pour la recette $R2$. Supposons que le score de similarité entre $(U1, U2)$ est égal à 0.092 et celui entre $(U2, U6)$ est égal à 0.96. On constate sur la figure 6.21, que la prédiction de l'évaluation pour la recette $R2$ par l'utilisateur $U1$ est d'environ 2.8.

	R1	R2	R3	R4	R5
U1	4	?	?	5	1
U2	5	5	4		
U6		3		4	2

FIGURE 6.20 – Matrice d'utilité

$$r_{U1,R2} = \frac{(0.092 * 5) + (0.82 * 3)}{0.092 + 0.96} = 2.77$$

FIGURE 6.21 – Résultat de la prédiction

Nous effectuons ce processus pour l'ensemble des recettes de la matrice d'utilité qui ne sont pas évaluées par l'utilisateur. Puis, nous récupérons les recettes dont la prédiction de score est plus grande ou égale à 3. Ensuite, nous trions ces recettes par score et nous les suggérons à l'utilisateur.

6.5.4 Tests et certifications

Afin de tester notre algorithme de recommandation, nous créons un utilisateur auquel nous faisons évaluer positivement les deux mêmes recettes que celles avec l'algorithme de recommandation basé sur le contenu. Étant donné le nombre conséquent de recettes à suggérer, nous n'afficherons que les quatre meilleures, c'est-à-dire celles qui ont les scores de prédictions les plus élevés. Dans la figure 6.22, nous constatons que les recettes à suggérer ont une similarité beaucoup moins importante que celles avec l'algorithme basé sur le contenu. Les recettes à suggérer ont au maximum deux ingrédients similaires. De plus, on constate que la recette 4 ("Soufflé citron et amande") a peu de similarité avec les recettes évaluées. Cependant, l'ensemble des recettes suggérées sont des desserts comme celles évaluées.

Remarque : Nous ne prenons pas en compte les ingrédients "sucre en poudre", "sucre", "farine", "levure", "oeufs" et "beurre" car ce sont des ingrédients universels qui apparaissent dans de nombreuses recettes. De plus, il existe peu d'intérêt à suggérer des recettes selon ces ingrédients.



FIGURE 6.22 – Extrait des recettes à suggérer pour un utilisateur

Maintenant que nous avons implémenté nos deux algorithmes de recommandation, nous pouvons comparer leurs temps d'exécution. Nous prenons un utilisateur ayant évalué plus d'une cinquantaine de recettes et nous effectuons 10 fois le processus de suggestions. De plus, nous générerons plus de 1000 clients dont chacun a noté 50 recettes.

Remarque : Notons A l'algorithme de recommandation basé sur le contenu et B l'algorithme de filtrage collaboratif.

On constate sur les deux figures ci-dessous, que le temps d'exécution de l'algorithme A est plus rapide que l'algorithme B. En effet, le temps d'exécution maximum de A est de 8 secondes alors que celui de B est de 10.5 secondes.

Cet écart de temps peut être dû au fait que la matrice d'unicité de A est beaucoup plus petite que celle de B. En effet, celle de A contient dans ses colonnes les ingrédients distincts des recettes évaluées positivement par l'utilisateur ciblé. En revanche, pour celle de B, elle contient l'ensemble des recettes évaluées par l'utilisateur ciblé et les utilisateurs semi-similaires. Par conséquent, le temps d'exécution des calculs vectoriels est plus long pour B. On constate sur la figure 6.25, que le nombre de colonnes de la matrice d'utilité de B est environ 6 fois plus élevée que celle de A.

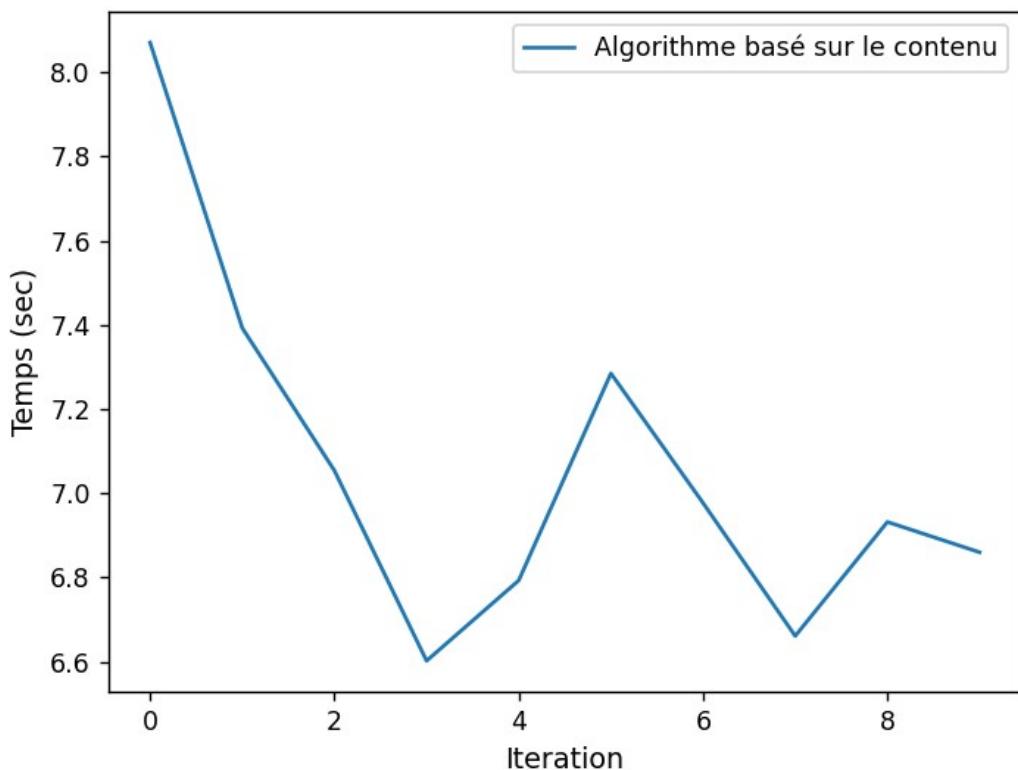


FIGURE 6.23 – Temps d'exécution pour l'algorithme basé sur le contenu

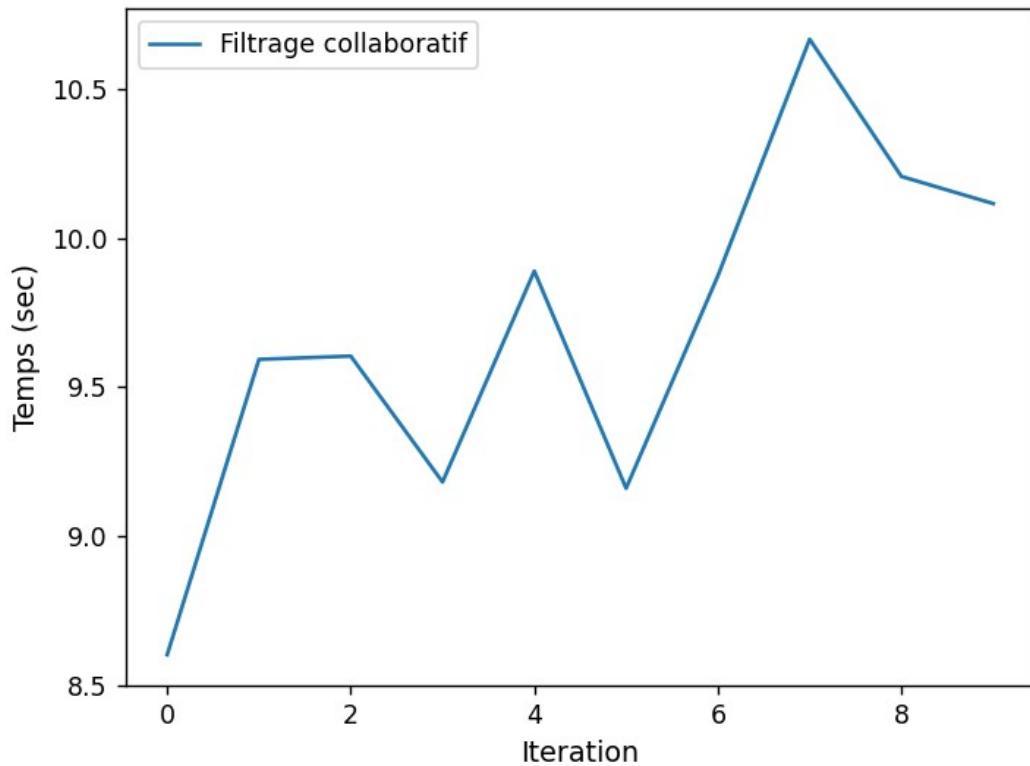


FIGURE 6.24 – Temps d'exécution pour l'algorithme de filtrage collaboratif

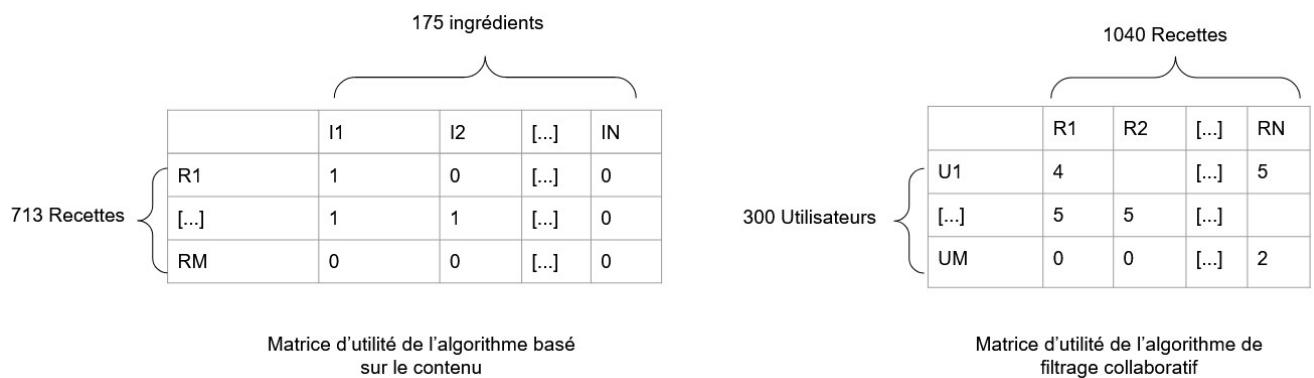


FIGURE 6.25 – Comparaison des matrices d'utilité

Chapitre 7

Rendu final

Ce chapitre a pour objectif de présenter l'IHM graphique de l'application web et les résultats des différentes fonctionnalités du projet.

7.1 Interface utilisateur finale

Dans cette sous-section, nous parlerons de la conception et du rendu final de l'interface homme-machine (IHM) de notre application. Les différentes pages et les fonctionnalités proposées à l'utilisateur seront détaillées.

Remarque : La construction de notre apprentissage supervisé (arbre de décision) et non supervisé (K-Means clustering) ne sont pas des fonctionnalités propres à l'utilisateur. Ces processus sont implémentés avant la mise en fonctionnement de notre application.

Lorsque l'utilisateur arrive sur notre application web, la page d'accueil s'ouvre. Cette page est celle où nous appellerons notre algorithme de recommandation afin de suggérer des recettes à notre utilisateur (figure 7.1). Lorsque l'on clique sur une recette, la page de celle-ci s'ouvre (figure 7.2). Cette page contient les détails de la recette (liste des ingrédients, nombre de commentaire, note globale, préparations de la recette, etc...). Si l'utilisateur est connecté à son compte, il peut évaluer la recette en cliquant sur le bouton "Donner votre avis".



Recettes que vous pourriez aimer



Esquimaux croquants chocolat au lait et pistache



Duo de mousse: chocolat et macarons



Poêlée d'asperges vertes et coppa



Conchiglioni au thon



Sauté de veau facile



Terrine de Saint-Jacques



Thé à la menthe marocain



Mossli (épaule de mouton rôtie) (Tunisie)



Limonade à l'ancienne maison

FIGURE 7.1 – Page d'accueil de notre application web



Recettes



Cupcake chocolat et cerise

Dessert;Gâteau;Petits gâteaux;Cupcake

Préparation: 20 min

Cuisson: 20 min

Repos: -

0.0/5 0 évaluations

[Donnez Votre Avis](#)



01. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum nec varius dui. Suspendisse potenti. Vestibulum ac pellentesque tortor. Aenean congue sed metus in iaculis. Cras a tortor enim. Phasellus posuere vestibulum ipsum, eget lobortis purus. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.

02. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum nec varius dui. Suspendisse potenti. Vestibulum ac pellentesque tortor. Aenean congue sed metus in iaculis. Cras a tortor enim. Phasellus posuere vestibulum ipsum, eget lobortis purus. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.

Ingredients

50g cerises

3 oeufs

250g chocolat noir

150g beurre

100g sucre

150g farine

Commentaires (1)

test

3/5

19/04/2022

j'aime beaucoup la recette

FIGURE 7.2 – Page de la recette

Si l'utilisateur veut s'inscrire ou se connecter à son compte, il devra cliquer sur le lien "profil" situé dans l'en-tête des différentes pages. Cette page contiendra les différentes données de l'utilisateur (nom, prénom, login, mot de passe). Il pourra modifier ses ingrédients de préférences qu'il devra séparer par un point-virgule. De plus, il pourra aussi choisir le type d'algorithme de recommandation.

Delicioso!



ACCUEIL RECETTES AJOUTER UNE RECETTE PROFIL QUI SOMMES-NOUS



Mon compte

Mon carnet

Déconnexion



Mon profil

Pseudo V89OQKLyC

Civilité Mr

Prénom

Nom

Email* kOSTs@gmail.com

Mot de passe [éditer](#)

Ingredients préférés tomate;cerise;jambon

Veuillez séparez les ingrédients par ";"

Algorithme de Suggestion
 Algorithme basé sur le contenu
 Algorithme de filtrage collaboratif

[SUBMIT](#)

FIGURE 7.3 – Page du profil de l'utilisateur

Enfin, nous avons la page de recherche où l'utilisateur peut effectuer une recherche par mot clefs ou par ingrédients.

Remarque : Nous avons une dernière page qui se nomme "Qui sommes-nous" et qui contient les détails du projet, de l'université et de notre groupe.

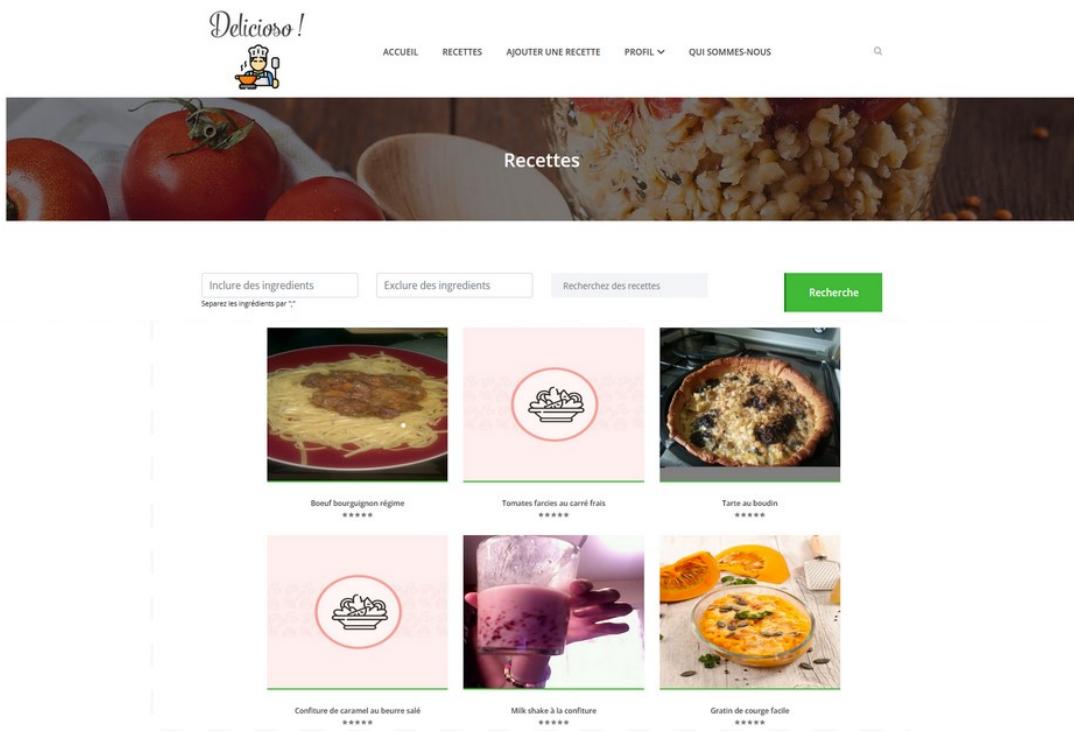


FIGURE 7.4 – Page de recherche

7.2 Tests utilisateur et certification

7.2.1 Recherche de recettes par ingrédients

La recherche de recettes par des ingrédients fait appel à notre arbre de décision. L'utilisateur doit ouvrir la page de recherche et écrire les ingrédients qu'il veut avoir dans les recettes (1), puis, il doit soumettre sa requête en appuyant sur un bouton (2). L'ensemble des recettes seront ensuite affichées (3). Notons que les recettes affichées contiendront au moins un des ingrédients demandés. Il n'y a pas d'obligation à ce qu'une recette contienne exactement tous les ingrédients demandés.

Remarque : l'utilisateur peut aussi écrire des ingrédients qu'il ne veut pas avoir dans des recettes ou effectuer une recherche par mots-clefs. Cette dernière ne nécessite pas l'utilisation de notre arbre de décision.

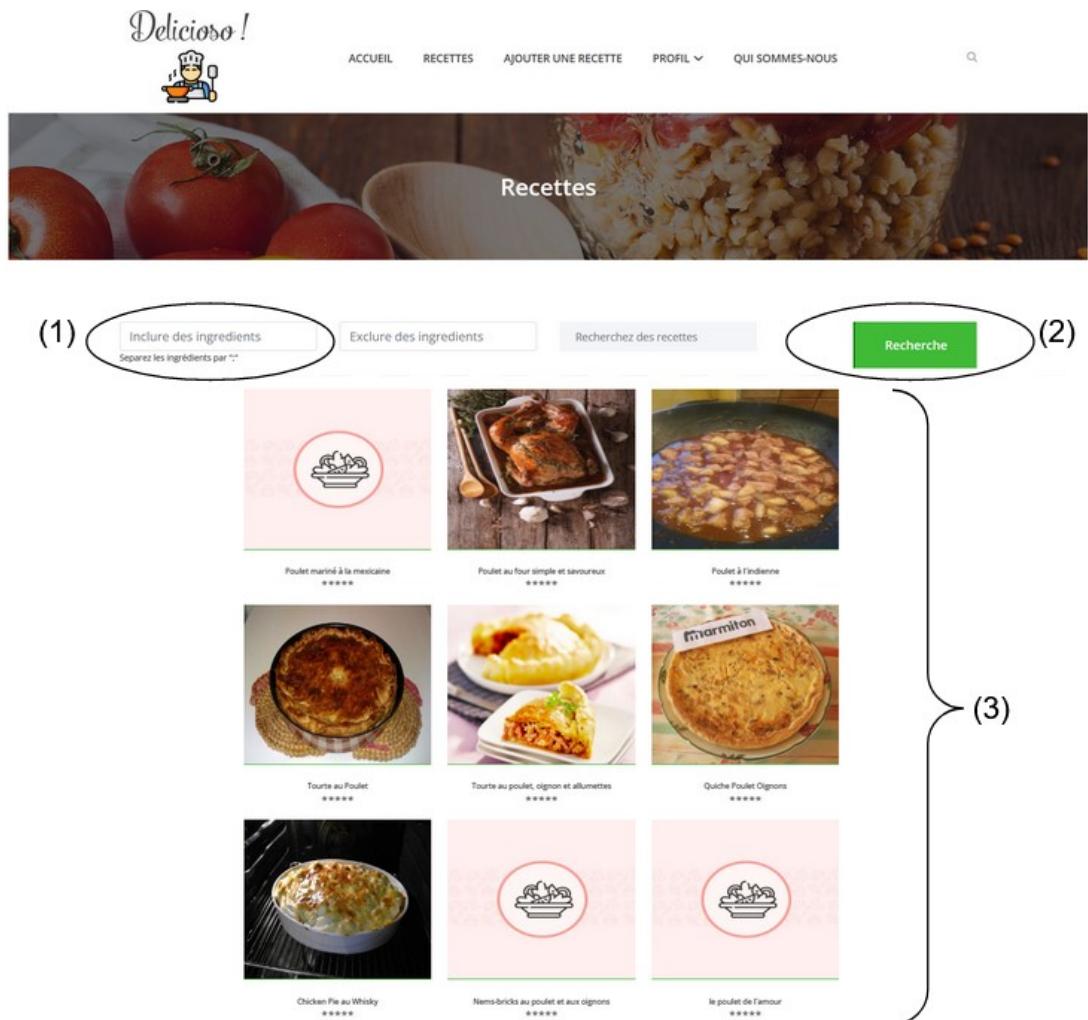


FIGURE 7.5 – Recherche de recettes par ingrédients

7.2.2 Ajout de recette

Pour pouvoir ajouter une nouvelle recette, il faut cliquer sur le lien "Ajouter une recette" qui ouvrira une fenêtre dans laquelle nous devons écrire les caractéristiques de la recette. Puis, en cliquant sur "Soumettre", la recette sera stockée dans la base de données et il sera possible de la visualisé (figure 7.6).

Ajoutez une recette

Nom

Catégorie +
 ×

Image de la recette (URL)

Temps de préparation heures minutes

Temps de cuisson heures minutes

Temps de repos heures minutes

Ingédients

400	g	Philadelphia	+
200	g	framboises	×
200	g	Beurre	×
140	g	Sucre	×
300	g	speculoos	×

Préparations

+
 ×

Soumettre

FIGURE 7.6 – Ajout des caractéristiques de la nouvelle recette

Cheese Cake à la framboise

Desserts;Gateau

Préparation: 1h20

Cuisson: -

Repos: 2h

     0.0/5 0 évaluations

Donnez Votre Avis



- 01.** Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum nec varius dui. Suspendisse potenti. Vestibulum ac pellentesque tortor. Aenean congue sed metus in iaculis. Cras a tortor enim. Phasellus posuere vestibulum ipsum, eget lobortis purus. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.

- 02.** Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum nec varius dui. Suspendisse potenti. Vestibulum ac pellentesque tortor. Aenean congue sed metus in iaculis. Cras a tortor enim. Phasellus posuere vestibulum ipsum, eget lobortis purus. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.

Ingredients

	400 crème fraîche
	300 spéculoos
	200 beurre
	140 sucre
	100 framboises
	400 Philadelphia

FIGURE 7.7 – Visualisation de la nouvelle recette

7.2.3 Suggestion de recettes

Pour pouvoir tester l’algorithme de recommandation, il faut tout d’abord que l’utilisateur crée un compte sur l’application (figure 7.8). Sur son profil, il pourra également écrire des ingrédients de préférence (ici nous écrivons "chèvre" et "salade"). Si l’utilisateur n’a évalué aucune recette, nous suggérons des recettes selon ces ingrédients (figure 7.9).



Inscription

[Se connecter](#)

Pseudo*

Civility*

Prénom

Nom

Email*

Mot de passe*
Le mot de passe doit contenir entre 8 et 30 caractères

Confirmation mot de passe*

Ingredients préférés
Veuillez séparez les ingrédients par ";"

SOUMETTRE

FIGURE 7.8 – Crédit à la création du compte utilisateur



Recettes que vous pourriez aimer



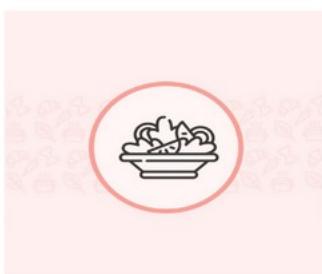
Cake à la Courgette Lardons et Fromage de Chèvre
★★★★★



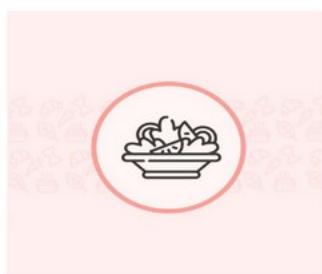
Mini hamburgers
★★★★★



Gâteaux moelleux Saumon Chèvre Épinard
★★★★★



Flan de tomates et chèvre
★★★★★



Salade de mâche aux crevettes et lait de coco
★★★★★



Tarte courgette saumon chèvre
★★★★★

FIGURE 7.9 – Suggestion de recettes basées sur les ingrédients de préférence

Maintenant, faisons évaluer positivement deux recettes (figure 7.10 et figure 7.11) à l'utilisateur.

Remarque : les recettes évaluées négativement ne seront pas prises en compte dans l'algorithme de recommandation.

The screenshot shows the Delicioso! website interface. At the top, there is a navigation bar with links: ACCUEIL, RECETTES, AJOUTER UNE RECETTE, PROFIL, and QUI SOMMES-NOUS. A search icon is also present. Below the navigation bar is a large banner image featuring a bowl of rice and some tomatoes. The word "Recette" is centered over the image. Underneath the banner, the title of the recipe is "Queue de lotte au curry". Below the title, the text "Plat principal; Viande; Viande en sauce; Viande à l'indienne" is listed. To the right of the title is a small image of the dish, which appears to be a piece of fish in a creamy, yellowish sauce. On the left side of the recipe card, there is a box containing preparation and cooking times: "Préparation: 15 min", "Cuisson: 10 min", and "Repos: -". Below these times is a rating section showing 2.9/5 stars from 19 evaluations. At the bottom of the card is a green button labeled "Donnez Votre Avis".

FIGURE 7.10 – 1ère recette évaluée par l'utilisateur

The screenshot shows the Delicioso! website interface, similar to Figure 7.10. The navigation bar and search icon are at the top. Below the navigation bar is a large banner image featuring a bowl of rice and some tomatoes. The word "Recette" is centered over the image. Underneath the banner, the title of the recipe is "Poêlée de riz, brocolis et poulet au lait de coco". Below the title, the text "Plat principal; Pâtes riz semoule; Plat de riz" is listed. To the right of the title is a small image of the dish, which appears to be a mix of rice, broccoli, and chicken in a coconut milk sauce. On the left side of the recipe card, there is a box containing preparation and cooking times: "Préparation: 30 min", "Cuisson: 15 min", and "Repos: -". Below these times is a rating section showing 2.6/5 stars from 23 evaluations. At the bottom of the card is a green button labeled "Donnez Votre Avis".

FIGURE 7.11 – 2ème recette évaluée par l'utilisateur

Evaluation

x

Donnez une note



Laisser un commentaire

Message

Post Comments

FIGURE 7.12 – Pop-in pour évaluer une recette

En retournant sur l'accueil, cela activera le processus de notre algorithme de recommandation qui affichera les recettes à suggérer à l'utilisateur. Comme nous pouvons le constater sur la figure 7.13, les recettes sont similaires à celles évaluées par l'utilisateur.

The screenshot shows a user profile with the name 'Delicioso!' and a chef icon. Navigation links include ACCUEIL, RECETTES, AJOUTER UNE RECETTE, PROFIL, QUI SOMMES-NOUS, and a search icon. A section titled 'Recettes que vous pourriez aimer' displays six recommended recipes with images and titles:

- Curry de lentilles (★★★★★)
- Poulet curry coco simplissime (★★★★★)
- Poulet aux cacahuètes (★★★★★)
- LASAGNES AU SAUMON (★★★★★)
- Ballotins de saumon fumé à l'oeuf poché (★★★★★)
- Lasagnes maison de A à Z (★★★★★)

FIGURE 7.13 – Affichage des recettes à suggérer

Chapitre 8

Gestion de projet

Ce chapitre sera consacré au déroulement et la gestion de notre projet durant cette année. Nous détaillerons le cycle de vie du projet et des différentes phases qui le compose. Nous parlerons ensuite de la répartition des tâches entre les membres de l'équipe et des différents outils logiciels utilisés.

8.1 Méthode de gestion

8.1.1 Phase d'étude

Pour commencer, nous avons identifié les besoins réels du client pour notre projet et les différentes manières de les concevoir. Au cours de cette étape, nous avons :

- définit les différents objectifs réalisables et les principaux résultats attendus.
- réalisé une étude de faisabilité pour identifier les problèmes principaux et déterminer si notre projet apportera la solution à ces problèmes.
- identifié les résultats pour définir les services à fournir et la porté pour définir l'étendue du projet.
- élaboré un énoncé des tâches en documentant les objectifs et les résultats que nous avons identifiés.

Suite à cela, nous avons réparti les différents rôles de notre équipe. Nous avons un chef de projet ainsi qu'un assistant chef de projet afin de répartir la surcharge de travail de gestion. Les autres membres de l'équipe sont des développeurs.

Membre de l'équipe	Rôle
Arthur MIMOUNI	Chef de projet
Mamadou Bella DIALLO	Assistant Chef de projet
Marouane RACHIDY	Développeur
Imane CHBIRA	Développeuse
Matthieu SAUVAGEOT	Développeur

FIGURE 8.1 – Répartition des rôles de l'équipe

8.1.2 Phase de planification

Une fois la phase d'étude du projet terminé, nous avons commencé la phase de planification. Cette phase du cycle de vie du projet consiste à séparer les différentes tâches, à constituer les différents

groupes du projet et à planifier les différentes releases des missions. De plus, nous avons veillé à ce que l'ensemble des missions soient réalisables dans les délais impartis. Pour répartir les tâches, nous avons effectué une réunion de lancement afin d'intégrer l'équipe et présenter dans les grandes lignes les différentes missions pour travailler le plus rapidement possible.

Sur la figure 8.2, se trouve la planification prévisionnelle effectuée durant cette première réunion. Nous avons choisi de répartir l'équipe en plusieurs groupes de deux et d'effectuer une release chaque mois.

Pour la première release, plusieurs objectifs devaient être atteints :

- effectuer le traitement de nos données (collecte, nettoyage et insertion).
- élaborer une première version de notre K-Means Clustering et de notre algorithme de recommandation avec des données de test.
- construire la page d'accueil, d'inscription et du profil de notre IHM graphique.

Pour la deuxième release, nous devions :

- améliorer notre K-Means clustering et notre algorithme de recommandation en utilisant les données de notre base de données.
- concevoir une première version de notre arbre de décision avec des données de test.
- concevoir la page de recherche et la page des recettes.

La troisième release consistait à :

- améliorer notre arbre de décision en utilisant les données partitionnées par notre K-Means Clustering.
- élaborer la deuxième version de notre algorithme de recommandation en utilisant les données partitionnées et notre arbre de décision précédemment conçu.
- implémenter les fonctionnalités de l'utilisateur sur l'IHM graphique (visualisation et évaluation des recettes).

L'objectif de la quatrième release fut de terminer l'algorithme de recommandation ainsi que notre IHM graphique. Enfin, la dernière release consistait à améliorer les différents modules de notre projet en ajoutant des extensions intéressantes que nous détaillerons dans la section 9.2 du chapitre 9.

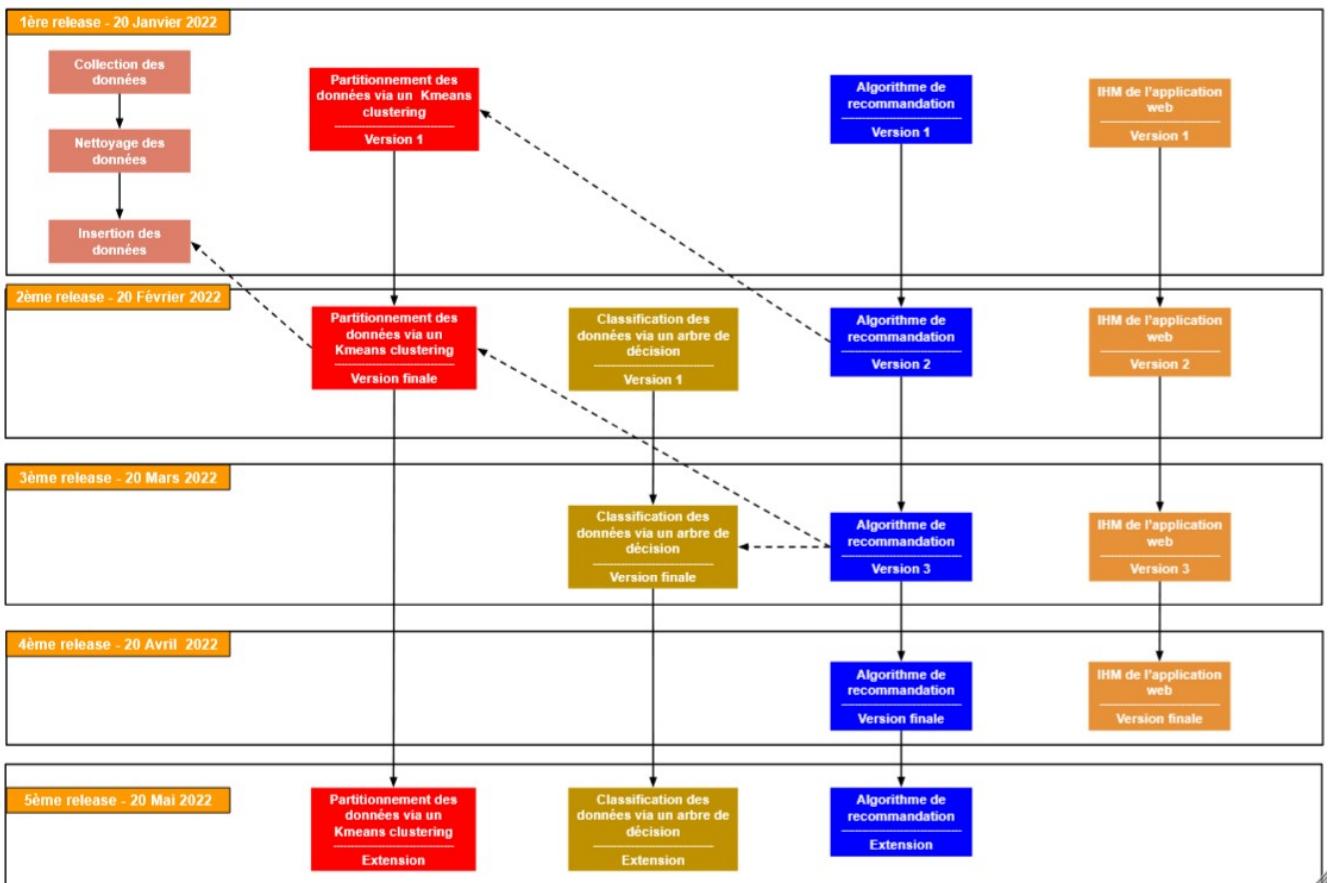


FIGURE 8.2 – Planification prévisionnelle

8.1.3 Phase d'exécution

Après avoir reçu l'approbation et les différentes remarques de notre tuteur technique, nous avons commencé à exécuter notre plan d'action. Une des principales missions de gestion dans cette phase de cycle concernait le chef de projet qui avait pour objectif de :

- créez les tâches et les affectez aux membres de l'équipe concernés, en veillant à ne pas les surcharger de travail.
- expliquez les tâches aux membres de l'équipe en leur donnant les indications nécessaires pour les concevoir.
- communiquez régulièrement avec les membres de l'équipe et le tuteur technique afin de faire des compte-rendus.
- contrôlez la qualité du travail et s'assurez que chaque membre de l'équipe respecte leur objectif de temps et de qualité pour chaque tâche.

8.1.4 Phase de clôture

Une fois le travail sur le projet terminé, nous sommes entré dans la phase de clôture dans laquelle chaque membre de l'équipe a fourni les résultats finaux de leur tâche. C'est aussi dans cette phase, où nous avons déterminé la réussite du projet et évalué ce qui a fonctionné ou non dans le cadre du projet. Nous avons analysé les performances de chaque membre de l'équipe en évaluant leur rapidité et la qualité de leur travail. Enfin, nous avons effectué un bilan de post-réalisation sur l'ensemble de nos travaux.

Sur la figure 8.3, vous pouvez trouver la planification réalisée. On constate quelques changements par rapport à la planification initiale, notamment l'ajout d'un module d'extension au projet. En effet, de nombreuses contraintes et conflits entre les membres de l'équipe sont apparus durant le projet et ont affecté notre planification. Nous détaillerons plus précisément ces aspects dans la section 8.4.

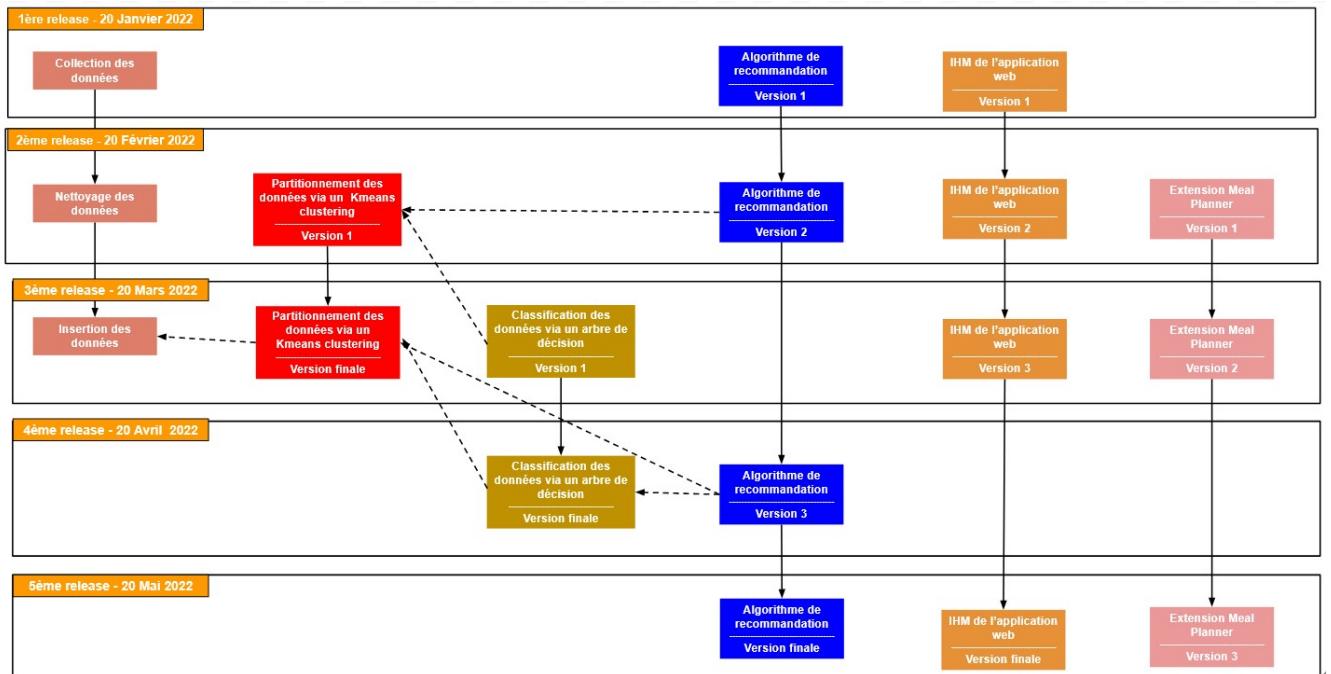


FIGURE 8.3 – Planification réalisée

8.2 Répartition de tâches

8.2.1 Répartition de tâches prévisionnelles

Lors de la phase de planification, nous avons décidé de répartir l'équipe en plusieurs sous-groupes de deux personnes. Un groupe pour la collecte des données et du partitionnement de celles-ci, un autre pour la classification des données et l'implémentation de notre algorithme de recommandation. Et enfin, un dernier groupe pour la création de l'IHM graphique. L'équipe étant impaire, nous avons décidé qu'une personne assistera les autres groupes.

Répartition des tâches

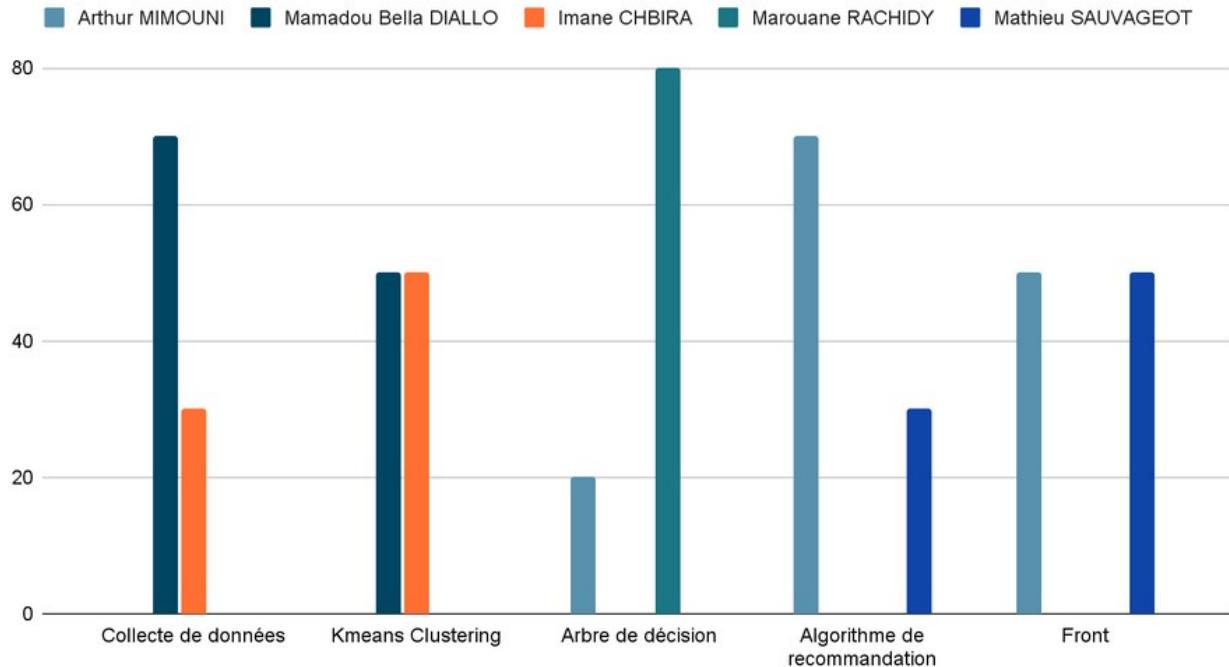


FIGURE 8.4 – Répartition des tâches prévisionnelles

8.2.2 Répartition de tâches réalisé

Comme vous pouvez le constater sur la figure 8.5, la répartition des tâches a beaucoup évolué durant le projet. Cette évolution est dû à certaines problématiques de gestions que nous avons fait face et qui seront abordés dans la section 8.4.

Répartition des tâches

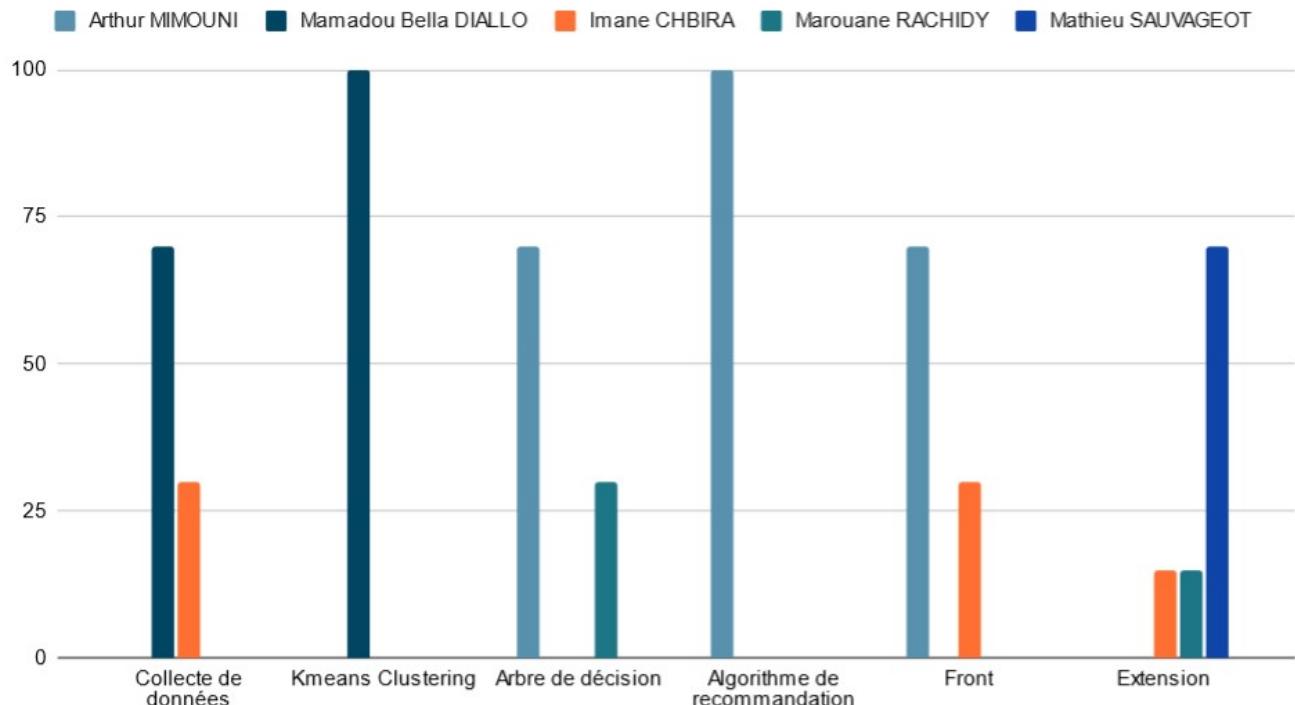


FIGURE 8.5 – Répartition des tâches réalisée

8.3 Utilisation des outils de gestion de projet au sein de l'équipe

8.3.1 Trello

Pour répartir les différentes tâches du projet et structurer les missions de chaque module, nous avons utilisé le logiciel **Trello**. Il s'agit d'un outil de gestion de projet en ligne, lancé en septembre 2011 et inspiré par la méthode Kanban de Toyota. Il repose sur une organisation des projets en planches listant des cartes, chacune représentant des tâches. Nous avons eu recours à cet outil avant la phase d'exécution pour que chaque personne de l'équipe soit consciente des objectifs à atteindre

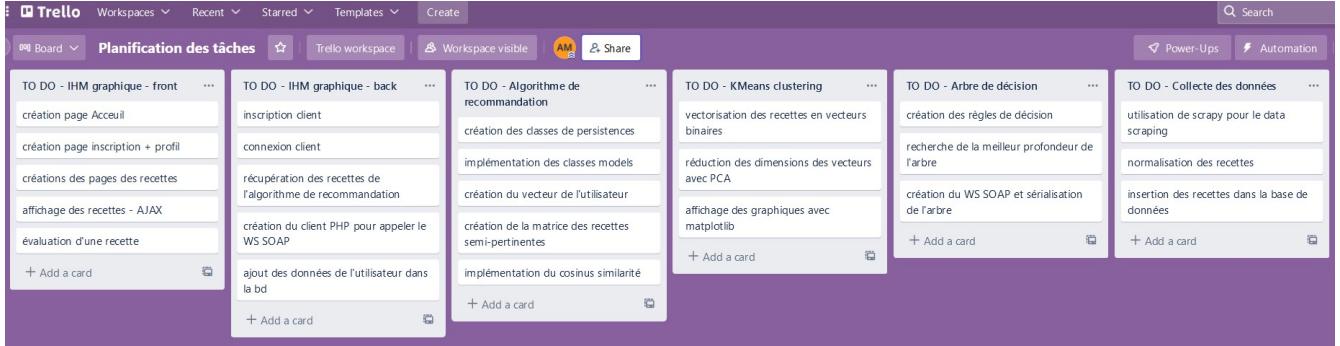


FIGURE 8.6 – Exemple de notre tableau Trello avant la phase d'exécution

8.3.2 YouTrack

YouTrack est un outil de suivi des problèmes et des bogues basé sur le Web. Cet outil a permis aux membres de l'équipe de connaître dynamiquement l'état d'avancement de chaque tâche, des tâches programmées et des bugs trouvées mais aussi de gérer les flux de travail. La figure 8.7 illustre un exemple du dashboard de notre youtrack. Nous pouvons voir dans l'encadré (1), les différentes personnes du groupe, et dans l'encadré (2), l'ensemble des tâches (qui se nomment "tickets") du projet. L'ensemble des tickets de chaque personne peut-être répartit dans des widgets pour une meilleure visualisation (3).

FIGURE 8.7 – Exemple de notre dashboard YouTrack

Lorsque l'on clique sur un ticket YouTrack, nous pouvons visualiser plusieurs détails sur celui-ci. Il y a bien entendu, le titre et le descriptif du ticket ainsi que les commentaires. En plus de cela, nous

pouvons connaitre la priorité du ticket (mineur, normal, critique, bloquant, etc..) (1), le type de tâche (bogue, évolution, etc..) (2), l'état du ticket (ouvert, en cours, clos, etc..) (3) ou encore le responsable du ticket. L'ensemble de ces dispositifs nous permet de gérer dynamiquement les tâches du projet.

The screenshot shows a YouTrack ticket interface for a task titled "BGD - [python] - Decision tree". The ticket details are as follows:

- Project:** Big Cooking Data
- Priorité:** Normal (1)
- Type:** Bogue (2)
- État:** Soumis (3)
- Responsable:** Arthur (4)
- Sous-système:** Aucun sous-système
- Versions de correction:** Non planifié
- Versions affectées:** Inconnu
- Corrigé dans la build:** Build suivante
- Tableaux:** Aucun tableau de bord visible

The ticket body contains the following text:

Terminer la conception de l'arbre de décision et effectuer la sérialisation de l'arbre dans un fichier.

+ Lier le ticket

Paramètres d'activité ▾

Arthur il y a environ 2 mois
État: Soumis → En cours
Responsable: Arthur

Arthur il y a 42 minutes
État: En cours → Soumis

Arthur il y a 38 minutes
Serialisation effectuée

Rédigez un commentaire, en utilisant @ pour mentionner des personnes

FIGURE 8.8 – Exemple d'un ticket YouTrack

8.3.3 Discord

Cet outil nous sert d'espace commun où nous partageons les ressources utiles au projet, organisons nos réunions et où nous demandons de l'aide en cas d'erreur urgente et critique. Concernant les réunions, nous sommes partis sur un rythme de trois réunions obligatoires par mois. Une en début de mois pour lister les tâches à effectuer, une autre en milieu de mois pour visualiser le travail de chacun et évaluer ce qu'il reste à effectuer, et une en fin de mois pour clôturer les tâches. Cependant, en cas d'urgence des réunions exceptionnelles peuvent être organisées.

8.3.4 GitHub

GitHub est un site web conçu pour fédérer et partager le code source d'un projet de développement d'application. Afin de simplifier le travail d'équipe, nous avons eu recours à ce logiciel de gestion. Nous avons ainsi pu éviter les erreurs de versionnage entre les membres de l'équipe, simplifier la gestion de conflits et organiser la validation du code au travers des Merge Request et des reviews. La figure 8.9 est un exemple de l'état de notre répertoire.

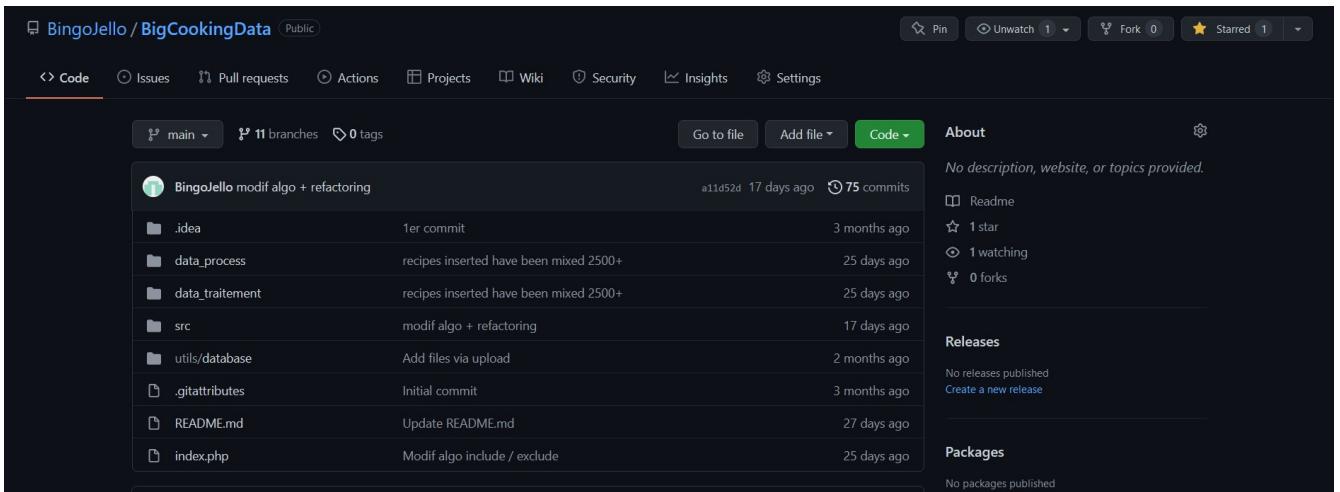


FIGURE 8.9 – Exemple de l'état de notre répertoire GitHub

8.4 Gestion des réunions et des conflits de l'équipe

Rassembliez un groupe de personnes pour travailler au sein d'un projet d'une longue durée peut amener à des conflits à un moment donné. Chaque personne du groupe a un besoin, un objectif, une attitude ou une vision différentes qui ne correspondent pas toujours à ceux des autres. Ce fut le cas dans notre projet où des différences en ce qui concerne les méthodes de travail sont apparues. L'objectif était donc de traiter rapidement et professionnellement ces différences pour éviter les problèmes affectant le moral et la productivité, et s'assurer qu'il n'y a aucun retard dans l'achèvement des tâches. Un des principaux conflits entre les membres de l'équipe fût le manque de connaissances pour certains langages informatiques et certaines notions importantes à connaître. Nous avons donc dû répartir d'une autre manière les tâches à effectuer afin de ne pas rester bloqué et de trouver des missions auxquelles tous les membres de l'équipe seront en mesure de participer. Une branche "Extension" a donc été ajouté dans la liste des tâches et consiste à travailler sur des fonctionnalités secondaires du projet. D'autre part, pour aider les personnes en difficulté sur certaines tâches, nous avons effectué plusieurs réunions d'urgence pour parvenir à diriger la personne ciblée vers une solution. D'autres courtes réunions ont été mise en place afin d'encourager la prise d'initiative et de vérifier que chacun apporte sa part d'implication. Même si les différents rendus des tâches n'ont pas toujours été correctes pour certains membres de l'équipe, l'intérêt de ces méthodes était de responsabiliser et professionnaliser chaque membre.

Chapitre 9

Conclusion et perspectives

9.1 Conclusion

Durant plusieurs mois de travail, nous avons eu l'occasion de mettre en pratique des notions informatiques apprises depuis plusieurs années d'études dans un projet qui englobe une large étendue de compétences différentes. Nous avons appris que la gestion d'un projet de cette envergure n'est pas évidente et nécessite un suivi constant et sérieux pour chaque personne du groupe. Nous avons au cours des différentes phases de travail, dû prendre des décisions importantes et faire des compromis. De plus, nous avons dû nous montrer souples en travaillant sur différents modules simultanément, et trouver des solutions rapidement pour optimiser nos ressources humaines afin d'éviter tout retard. D'autre part, le sujet de ce projet a cette particularité de rassembler divers domaines de l'informatique : Conception Orienté Objet, IA, Big Data. Cette pluridisciplinarité fut un obstacle que nous avons surmonté grâce aux compétences multiples et à la polyvalence de notre équipe.

De notre réflexion est né un projet pour le moins enrichissant et formateur et qui a mené, selon nous, à une application intelligente respectant les exigences demandées.

En ce qui concerne l'élaboration de notre K-Means Clustering, le partitionnement a été correctement effectué et nous a permis d'organiser de façon optimale les différents clusters de recettes.

La conception de notre arbre de décision et les différentes évaluations effectuées nous ont permis de construire un modèle de classification avec une précision de prédiction élevée (environ 95%).

Notre algorithme de recommandation est un système complexe qui génère des recommandations comme nous l'espérons, et l'IHM graphique permet de visualiser de façon optimale les résultats des suggestions de recettes.

Le fait d'être amené à développer une telle application de A à Z est représentatif d'un travail complexe que l'on pourrait réaliser en entreprise, c'est donc un exercice très enrichissant

9.2 Perspectives

Même si les différents objectifs du projet ont été atteints, plusieurs extensions au projet peuvent être opérées.

9.2.1 Planificateur de repas

Le planificateur de repas est une extension à notre algorithme de recommandation et qui permet de planifier des repas sur une semaine selon des préférences de l'utilisateur. Ces préférences contiennent le nombre de calories exigé par jour, le type de régime et le nombre de repas par jour. L'utilisateur peut décider d'estimer le nombre de calories qu'il doit consommer en détaillant certaines caractéristiques (poids, âge, niveau d'activité, etc..).

Pour pouvoir réaliser cette extension, il nous faut créer un système intelligent permettant de compter le nombre de calories à consommer par rapport à des données d'entrée. D'autre part, le site web **Marmiton** ne permet pas de récupérer les informations nutritionnelles des recettes, l'objectif est donc de trouver une API permettant de récupérer ces informations par rapport à des ingrédients donnés.

The form consists of several input fields and dropdown menus:

- Current diet type:** vegan
- I want to:** Lose weight, Maintain (selected), Build muscle
- Preferred units:** U.S. Standard, Metric
- I am:** Male (selected), Female
- Height:** ft, in
- Weight:** lbs
- Age:** years
- Bodyfat:** Low, Medium (selected), High
 - under 14%
 - 14% to 22%
 - above 22%
- Activity level:** Lightly Active
- Set a weight goal?** No thanks, Yeah let's do it! (selected)
- Enter your goal weight:** lbs
- Weight change rate:** Lose, lbs per week

Calculate

FIGURE 9.1 – Estimation du nombre de calories

9.2.2 Partitionnement des données avec les quantités des ingrédients

Le partitionnement de données que l'on effectue se fait à l'aide de données binaires. Si la recette contient l'ingrédient i alors son vecteur binaire aura la valeur 1 à l'indice i , sinon si l'ingrédient est absent, il aura 0. Il serait donc intéressant de prendre en considération non plus l'appartenance d'un ingrédient à une recette mais plutôt sa quantité. Puis, nous pourrions comparer le partitionnement de cette méthode avec celui de notre projet. Pour effectuer cette extension, il nous faut donc convertir les quantités des ingrédients en grammes afin d'avoir des valeurs de même unité dans nos vecteurs.

Remarque : Si un ingrédient ne possède pas d'unité (exemple : "une pincée de sel", "1 oeuf"), nous considérons la quantité comme étant en grammes (exemple : "1 oeuf" devient "1" dans notre vecteur).

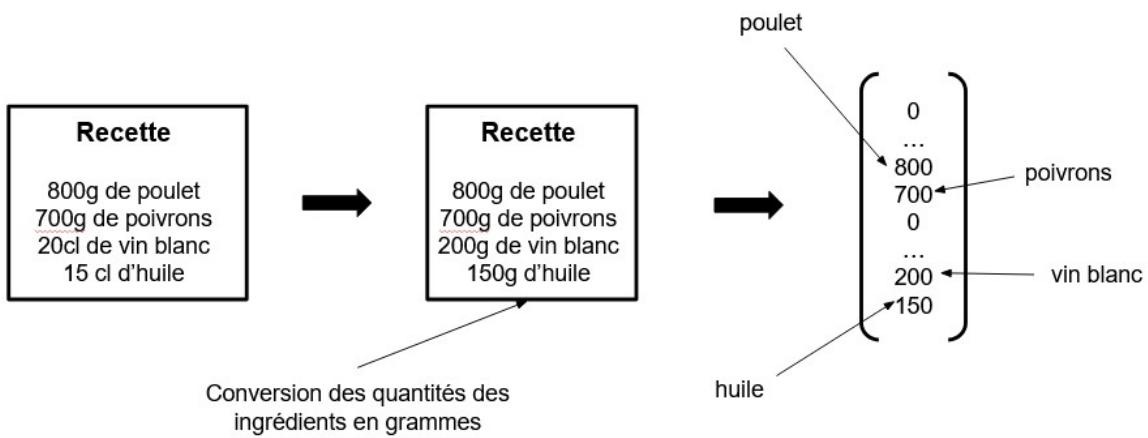


FIGURE 9.2 – Partitionnement des données avec les quantités des ingrédients

9.2.3 Inférence en temps réel de l'arbre de décision

Comme nous l'avons expliqué dans la section 5.5, une des problématiques lors de l'ajout d'une nouvelle recette est l'ajout de nouveaux ingrédients qui n'ont pas été entraîné par notre arbre de décision et notre Kmeans Clustering. Pour remédier à ce problème de manière optimale, nous pourrions implémenter un système temps réel qui aura pour objectif de détecter quand les performances de l'arbre passent sous un certain seuil. Ce procédé est ce que l'on appelle l'inférence en temps réel. Il s'agit d'une architecture dans laquelle l'inférence de modèle peut être déclenchée à tout moment, et où une réponse immédiate est attendue. C'est un modèle qui peut être utilisé pour analyser nos données en continu et nous permettre de tirer parti de notre modèle de Machine Learning.

Bibliographie

- [Ban21] Ankita Banerji. K-mean : Getting the optimal number of clusters, mai 2021.
- [Dev21] Google Developers. Recommendation systems, février 2021.
- [Dev22] Scrapy Developers. Architecture overview. 2022.
- [Kan19] Michel Kana. Principal component analysis and k-means clustering to visualize a high dimensional dataset, février 2019.
- [Kum20] Satyam Kumar. Silhouette method, better than elbow method to find optimal clusters, octobre 2020.
- [Oli21] Olivier. Créer un jeu de données avec scrapy, 2021.
- [Sim20] Milos Simic. How to find decision tree depth via cross-validation, mars 2020.
- [Uni21] Stanford University. Recommender systems and collaborative filtering, 2021.