PROJET RÉALISÉ PAR

LE GROUPE GRAPEMIND

RAPPORT DE GROUPE DE L'UE SCIENCES DES DONNÉES 4

Léna Charrière, Simon Godard–Piccot, Maxime Miroff.

Département MIASHS, UFR 6 Informatique, Mathématique et Statistique
Université Paul Valéry, Montpellier 3

Avril 2025

Soumis comme contribution partielle
pour le cours Science des données 3

Déclaration de non plagiat

Nous déclarons que ce rapport est le fruit de notre seul travail, à part lorsque cela est indiqué explicitement.

Nous acceptons que la personne évaluant ce rapport puisse, pour les besoins de cette évaluation:

- la reproduire et en fournir une copie à un autre membre de l'université; et/ou,
- en communiquer une copie à un service en ligne de détection de plagiat (qui pourra en retenir une copie pour les besoins d'évaluation future).

Nous certifions que nous avons lu et compris les règles ci-dessus.

En signant cette déclaration, nous acceptons ce qui précède.

Signature: Date: 10/04/2025
Léna CHARRIÈRE

Signature: Date: 10/04/2025
Simon GODARD–PICCOT

Signature: Date: 10/04/2025
Maxime MIROFF

10/04/2025.

Remerciements

Nous exprimons notre profonde gratitude à nos encadrants pédagogiques, Sandra Bringay, Théodore Michel–Picque et Namrata Patel, pour leur soutien précieux et leurs orientations judicieuses tout au long de ce projet.

Nous tenons également à remercier l'Université Paul Valéry et le Département MIAP (Mathématiques et Informatique Appliquées) pour leurs ressources et le cadre favorable à la réalisation de ce projet. Nous adressons également nos remerciements à nos camarades présents sur ce projet pour leur engagement, leurs idées et leur collaboration précieuse.

Ce projet nous a permis d'acquérir une expérience importante dans la gestion d'un projet. Ainsi qu'un goût particulier pour le développement web et tout ce qui l'entoure. Nous sommes reconnaissants pour toutes les opportunités d'apprentissage qu'il nous a offert.

10/04/2025.

Résumé

GrapeMind est une plateforme web innovante conçue pour faciliter la découverte et la recommandation de vins. Elle propose aux utilisateurs une expérience personnalisée grâce à un quiz de préférences gustatives et à une analyse poussée des données, permettant ainsi de suggérer des vins adaptés à leurs goûts.

Le projet intègre diverses fonctionnalités interactives, telles qu'une carte des domaines viticoles, des outils de gestion de compte (inscription, connexion, historique des consultations), et des algorithmes de recommandation.

Développé dans le cadre d'un projet universitaire en MIASHS, GrapeMind combine des techniques de scraping pour collecter des données (notamment depuis `Vivino.fr`) et des méthodes avancées de modélisation pour analyser et présenter ces informations de manière intuitive et visuelle.

Table des matières

Chapitre 1

Introduction

Notre projet 'GrapeMind' s'inscrit dans le cadre de notre dernière année de licence MIASHS (Mathématiques et Informatique Appliquées aux Sciences Humaines et Sociales) à l'Université Paul-Valéry Montpellier 3. À travers ce projet universitaire, nous avons cherché à développer une plateforme innovante de recommandation de vin, offrant une expérience personnalisée grâce à l'analyse des préférences gustatives des utilisateurs.

L'ambition de GrapeMind est de devenir un véritable sommelier personnel en ligne, capable de guider chaque utilisateur qu'il soit débutant ou novice dans le choix de ses vins, en tenant compte de ses préférences. Pour y parvenir, nous avons collecté et traité des données, tout en utilisant des techniques avancées de modélisation et de développement web.

Ce rapport présente les travaux réalisés au cours du second semestre. Nous y détaillons les améliorations apportées aux fonctionnalités existantes, ainsi que l'implémentation de nouvelles fonctionnalités clés, en justifiant les choix techniques faits pour optimiser l'efficacité de la plateforme. Nous évoquons également les principales difficultés rencontrées et les solutions mises en place pour y répondre.

Chapitre 2

Sujet

2.1 Description du sujet

GrapeMind, c'est notre idée pour révolutionner la façon dont on choisit et découvre des vins. On s'est dit que face à la multitude de choix en magasin ou en ligne, il est facile de se sentir perdu, surtout quand on ne s'y connaît pas beaucoup.

C'est là qu'intervient notre plateforme, pensée pour rendre cette expérience plus simple, amusante, et surtout, personnalisée.

Notre site se veut un véritable sommelier numérique, capable de vous proposer des vins adaptés à vos goûts. On a pris en compte les besoins des utilisateurs en créant un espace où chacun peut explorer des vins sans avoir à comprendre toutes les subtilités œnologiques.

Grâce à un test de sensibilité gustative que nous avons développer, GrapeMind analyse vos préférences et vous proposera des recommandations personnalisées. L'idée est que vous puissiez facilement découvrir des vins qui vous plairont, sans stress ni hésitation.

Mais ce n'est pas tout ! GrapeMind va encore plus loin en vous aidant à associer vos vins avec les plats parfaits. Vous pourrez ainsi sublimer vos repas en choisissant les meilleurs vins pour accompagner chaque plat.

Pour les amateurs de découvertes, GrapeMind propose aussi une carte interactive où vous pouvez explorer les domaines viticoles ainsi qu'un petit descriptif culturel à propos de la région. Cette fonctionnalité vous permet de visualiser les régions où sont produits vos vins préférés et de découvrir des vignobles.

Et si vous souhaitez poursuivre votre passion pour le vin dans la vie réelle, notre page *Événements* est faite pour vous. Nous y recensons des dégustations, des salons, et des rencontres avec des vignerons, pour vous permettre de partager votre intérêt pour le vin avec d'autres passionnés.

GrapeMind vous accompagne ainsi bien au-delà de l'écran, en vous connectant aux expériences et aux personnes qui font vivre le monde du vin.

En somme, GrapeMind est là pour vous simplifier la vie, que vous soyez un amateur de vin débutant ou un passionné curieux d'élargir vos horizons. Nous avons pensé à cette plateforme pour que chacun puisse trouver le vin parfait, explorer des domaines viticoles, participer à des événements, et créer des souvenirs inoubliables, tout en apprenant à mieux connaître ses propres goûts.

Alors, prêt à découvrir votre prochain vin préféré et à vivre l'expérience complète ?

2.2 Objectifs et enjeux

Aujourd'hui, avec l'immense variété de vins disponibles, il est facile de se sentir perdu. Choisir un vin qui correspond vraiment à ses goûts reste un défi pour beaucoup, surtout quand les recommandations sont souvent trop générales.

De plus, il manque des outils qui connectent l'expérience numérique à des expériences réelles, comme visiter des vignobles ou participer à des événements autour du vin.

Pour répondre à ces défis, notre projet *GrapeMind* vise à simplifier la découverte du vin en proposant des recommandations personnalisées et accessibles à tous.

Nos objectifs sont clairs :

- **Personnalisation des recommandations** : Offrir une expérience unique en analysant les préférences gustatives de chaque utilisateur, pour l'aider à trouver des vins qui lui plaisent vraiment.
- **Amélioration des accords mets-vins** : Faciliter la création d'accords parfaits pour enrichir les repas de nos utilisateurs.
- **Connexion avec le monde réel** : Intégrer une carte interactive des domaines viticoles et une page d'événements pour prolonger l'expérience en dehors du numérique.
- **Accessibilité et simplicité** : Concevoir une interface intuitive qui s'adresse aussi bien aux novices qu'aux amateurs de vin.

Dans la seconde partie du semestre, nous avons recentré notre travail sur l'aspect *data science* du projet.

Nous avons mis en place un système de recommandations personnalisées basé sur un **quiz de départ**, permettant de mieux cerner les goûts de chaque utilisateur.

Nous avons également intégré un **chatbot interactif**, capable de suggérer un plat à partir d'un vin (ou l'inverse), afin de faciliter les accords mets-vins de manière ludique et intuitive.

Une nouvelle fonctionnalité permet aussi d'effectuer une **recherche par plat** au sein de sa cave personnelle. Celle-ci s'appuie sur une liste d'accords mets-vins déjà présente dans notre base de données.

Enfin, nous avons porté une attention particulière à l'**adaptabilité du site sur tous les formats d'écran**. Grâce aux retours des utilisateurs, nous avons ajusté certains éléments de design pour améliorer l'expérience globale.

Chapitre 3

Développement et outils

3.1 Logiciels

Dans le cadre du développement du site *GrapeMind*, plusieurs logiciels et technologies ont été utilisés. Voici la liste des principaux outils :

- **PhpStorm** : éditeur de code principal utilisé pour écrire et organiser les fichiers PHP, HTML, CSS et JavaScript du projet.
- **MAMP** : environnement de développement local permettant d'exécuter un serveur Apache, une base de données MySQL et PHP pour tester l'application en local.
- **phpMyAdmin** : interface graphique utilisée pour gérer la base de données MySQL du projet (création de tables, insertion de données, etc.).
- **Infomaniak** : hébergeur du site et de la base de données en ligne.
- **LaTeX** : utilisé pour rédiger ce rapport de manière professionnelle avec une mise en page propre.
- **Git** (et GitHub) : utilisé pour le versionnage du code et la collaboration entre les membres du groupe.

3.2 Bibliothèques et frameworks utilisés

En complément des logiciels mentionnés précédemment, notre projet repose également sur plusieurs bibliothèques et outils externes qui ont facilité le développement de fonctionnalités spécifiques. Voici une liste non exhaustive des principales ressources utilisées :

- **Leaflet.js** : bibliothèque JavaScript open-source utilisée pour afficher des cartes interactives sur le site. Elle nous a permis d'intégrer une carte visuelle avec des marqueurs géolocalisés, pour mettre en valeur certaines régions viticoles.
- **PhpMailer** : utilisée pour l'envoi automatique d'un e-mail contenant un lien de réinitialisation sécurisé.
- **jQuery** : bibliothèque JavaScript qui simplifie les manipulations du DOM, la gestion des événements et les requêtes AJAX. Elle a été utilisée notamment pour améliorer les interactions avec les utilisateurs.
- **IG Boost Chatbox** : plugin intégré pour ajouter une boîte de dialogue interactive sur le site, permettant aux utilisateurs de poser des questions ou d'être guidés dans leur navigation.

3.3 Hébergement site internet

L'hébergement du site web a représenté un point important dans le déploiement de notre projet. Dans un premier temps, nous avons opté pour l'offre étudiante gratuite proposée par *Infomaniak*, qui permet d'héberger un site web avec une base de données MySQL. Cette solution nous a permis de mettre en ligne rapidement une première version fonctionnelle du site, et de commencer à tester certaines fonctionnalités dynamiques comme l'inscription, la connexion, ou encore les interactions avec la base de données.

Cependant, nous avons rapidement été confrontés à une limitation majeure : Infomaniak repose sur un hébergement mutualisé, ce qui signifie que certaines technologies serveur, comme l'exécution de scripts Python, ne sont pas disponibles. Or, notre projet intègre plusieurs scripts Python, notamment des automatisations comme les recommandations basées sur les préférences.

Face à cette contrainte technique, nous avons décidé de changer d'environnement. L'un des membres du groupe disposait déjà d'un serveur personnel chez *OVH*, offrant davantage de flexibilité. Ce serveur, basé sur une architecture VPS (Virtual Private Server), nous a permis de déployer à la fois l'interface web et les scripts Python, tout en conservant un contrôle complet sur la configuration du système.

Nous avons ainsi pu intégrer nos services de traitement de données, notre chatbot, et nos APIs internes sans restriction.

En somme, ce passage d'un hébergement mutualisé à un serveur dédié s'est révélé déterminant pour le bon développement et le déploiement de l'ensemble des fonctionnalités de *GrapeMind*.

Chapitre 4

Intégration des fonctionnalités

4.1 Carte intéractive

Dans le but d'enrichir l'expérience utilisateur et de visualiser les données de manière plus intuitive, nous avons intégré une carte interactive affichant les domaines viticoles présents dans notre base de données. Cette fonctionnalité repose sur la bibliothèque JavaScript `Leaflet`, qui permet d'afficher une carte personnalisée, fluide et interactive directement dans le navigateur.

Lors du chargement de la page, une requête est automatiquement envoyée à notre serveur pour récupérer les coordonnées géographiques de chaque domaine viticole. Ces données sont stockées dans la base sous forme de latitude et longitude (`winery_lat`, `winery_lon`) et sont extraites via une API en PHP, qui renvoie les informations des domaines au format `JSON`. Chaque domaine comprend également son identifiant, son nom, son site web, ainsi que ses coordonnées.

Une fois les données chargées, des marqueurs sont placés sur la carte pour représenter les différents domaines. Afin de garantir la lisibilité lorsque plusieurs domaines sont proches les uns des autres, nous avons utilisé un système de regroupement automatique (cluster), basé sur la bibliothèque `Leaflet MarkerCluster`. Ce système regroupe visuellement les domaines situés dans une même zone, tout en permettant de zoomer pour les visualiser individuellement.

L'utilisateur peut également interagir avec une interface latérale qui propose une liste déroulante des régions. Lorsqu'une région est sélectionnée, les informations associées s'affichent, et la carte est centrée automatiquement sur la zone correspondante.

En complément, une requête est effectuée pour récupérer les vins produits dans le domaine sélectionné. Cette seconde API prend en paramètre l'identifiant du domaine (`WineryID`) et renvoie la liste des vins associés, avec leur nom, leur type, leur image et leur prix. Ces informations sont ensuite affichées dans une section dédiée à droite de la carte.

Une fonctionnalité de recherche par nom de vin a également été ajoutée. Elle permet de saisir une chaîne de caractères dans un champ de recherche, et de recevoir dynamiquement des suggestions de vins correspondants, accompagnées de leur image et de leur identifiant. Cette recherche s'effectue de manière asynchrone via une requête `AJAX`, et les résultats sont retournés au format `JSON`.

Cette carte offre donc une navigation visuelle et immersive à travers les domaines et les vins proposés sur la plateforme. Elle constitue une passerelle interactive entre les données géographiques et les informations produits, tout en s'intégrant naturellement à l'esthétique et à la philosophie du projet GrapeMind.

4.2 Réinitialisation du mot de passe

Pour offrir une expérience fluide et sécurisée, nous avons mis en place une fonctionnalité complète de réinitialisation du mot de passe. Celle-ci se déroule en deux étapes : la demande de réinitialisation, puis la mise à jour du mot de passe via un lien sécurisé envoyé par email.

Lorsqu'un utilisateur indique son adresse email via le formulaire dédié, le système vérifie que cette adresse est bien associée à un compte existant dans la base de données. Si l'adresse est valide, un jeton unique (`token`) est généré de manière sécurisée à l'aide de la fonction `random_bytes`, et une date d'expiration est fixée à deux heures. Ces informations sont enregistrées dans la base de données, dans les champs `reset_token` et `reset_expires`.

Une fois cette étape réalisée, un email est envoyé automatiquement à l'utilisateur grâce à la bibliothèque `PHPMailer`. Ce message contient un lien vers une page sécurisée permettant de réinitialiser son mot de passe. Le lien intègre le `token` généré en paramètre, ce qui permet d'identifier la demande de manière sûre. L'envoi se fait depuis l'adresse `noreply@grapemind.fr`, et les identifiants du serveur SMTP sont protégés grâce à la bibliothèque `Dotenv`, qui permet de stocker les données sensibles en dehors du code.

Lorsque l'utilisateur clique sur le lien, il accède à une page de réinitialisation du mot de passe. Avant d'autoriser la modification, le système vérifie que le `token` est valide, qu'il est bien associé à un compte, et qu'il n'a pas expiré.

Si toutes les conditions sont réunies, le nouveau mot de passe est haché à l'aide de `password_hash()` puis enregistré en base. Le `token` est alors supprimé (remis à `NULL`), de même que sa date d'expiration, pour éviter toute réutilisation.

Si le `token` est invalide ou expiré, l'utilisateur est informé par un message d'erreur, et aucun changement n'est effectué.

Cette fonctionnalité garantit une gestion sécurisée des mots de passe tout en offrant à l'utilisateur une solution simple et efficace pour retrouver l'accès à son compte en cas d'oubli.

4.3 Gestion de la cave, recherche par plat et traduction

Dans notre projet, nous avons développé une cave à vin virtuelle personnalisable ainsi qu'une liste d'envies. L'objectif est de permettre à l'utilisateur de constituer et gérer sa propre sélection de vins, tout en facilitant la recherche du vin idéal à partir d'un plat qu'il souhaite déguster.

Cave personnelle et liste d'envies

Chaque utilisateur peut :

- Ajouter des vins à sa cave, comme s'il gérait une vraie collection.
- Créer une liste d'envies, indépendante de la cave, pour repérer les vins à acheter plus tard.
- Retrouver facilement ses sélections dans un espace dédié, accessible depuis son compte.

Cette séparation entre cave et liste d'envies s'inspire d'un usage réel : d'un côté, les bouteilles déjà en possession, de l'autre, celles que l'on souhaite acquérir ou tester.

Recherche par plat (requête sur la base de données)

Lorsqu'un utilisateur saisit un plat dans la barre de recherche, le système interroge uniquement les vins présents dans sa cave réelle. La recherche repose sur une requête SQL préparée exécutée côté serveur, qui filtre les vins associés à l'utilisateur connecté, en vérifiant si le champ `Harmonize_FR` contient le nom du plat saisi.

Extrait de la requête SQL utilisée :

```
SELECT * FROM cave
JOIN descriptifs ON cave.idwine = descriptifs.idwine
JOIN scrap ON scrap.idwine = descriptifs.idwine
WHERE cave.id_user = ?
  AND cave.type = 'real'
  AND descriptifs.Harmonize_FR LIKE ?
```

La recherche s'appuie sur le champ `Harmonize_FR`, qui contient des associations directes entre plats et vins.

- Si une correspondance est trouvée (ex : un vin dont `Harmonize_FR` contient `"porc"`), alors les résultats s'affichent.
- Sinon, aucun résultat n'est proposé. Il n'y a pas de correspondance secondaire basée sur les arômes ou une logique de similarité.

API DeepL : pour une recherche multilingue

Pour améliorer l'expérience utilisateur, nous avons intégré l'API de traduction `DeepL` afin de traduire les correspondances plat-vin de notre base de données, de l'anglais vers le français.

Les associations initiales étaient saisies en anglais, dans le champ `Harmonize`. Nous avons donc utilisé `DeepL` pour créer un champ `Harmonize_FR`, contenant les versions françaises.

Ainsi, lorsqu'un utilisateur tape un plat en français dans la barre de recherche (par exemple : `"boeuf"`, `"fromage"`, `"poulet"`), la recherche s'effectue directement dans le champ `Harmonize_FR`, sans avoir à traduire dynamiquement l'entrée utilisateur.

4.4 Algorithme de recommandations et quizz intéractif

Dans le but d'offrir une expérience encore plus personnalisée à chaque utilisateur, nous avons mis en place un système de quizz couplé à un algorithme de recommandations. Ce système permet à chacun de recevoir des suggestions uniques, parfaitement adaptées à ses préférences.

Le quizz

Le quiz représente l'étape la plus importante dans notre démarche de recommandation, car il nous permet de proposer des vins précis et adaptés à chaque utilisateur. Pour cela, nous avons conçu un quiz composé d'une quinzaine de questions, dont l'objectif est d'établir une sorte de fiche de préférences. À travers ces questions, nous cherchons à connaître les goûts de l'utilisateur en matière de couleur de vin, d'arômes préférés et de budget.

Nous avons fait le choix d'intégrer directement ces questions dans notre base de données. Ce fonctionnement nous offre une grande flexibilité, car il nous permet d'ajouter ou de supprimer des questions facilement en fonction des besoins ou des retours utilisateurs. À la fin du quiz, l'ensemble des réponses est enregistré et rattaché à l'identifiant de l'utilisateur, ce qui nous permet de conserver un historique clair de ses préférences.

Ce système nous permet donc de passer en revue toutes les données liées aux caractéristiques des vins, qui sont susceptibles d'influencer les choix et les goûts des utilisateurs. Grâce à cette base, notre algorithme peut ensuite fonctionner de manière efficace et ciblée.

L'algorithme de recommandations

Pour développer notre algorithme de recommandation, nous nous sommes appuyés sur un ensemble de caractéristiques propres à chaque vin, comme les arômes (fruité, boisé, etc.), les accords mets-vins, le degré d'alcool, la couleur ainsi que des critères liés au prix. Lorsqu'un utilisateur remplit notre quiz, il exprime ses préférences personnelles, par exemple une préférence pour les vins rouges, pour certains types d'arômes ou pour une certaine gamme de prix. À partir de ces réponses, notre algorithme se met en marche en utilisant un système de notation par points.

Le fonctionnement est le suivant : à chaque réponse donnée, les vins correspondant aux préférences de l'utilisateur reçoivent un nombre de points supplémentaire, tandis que ceux qui ne correspondent pas perdent des points. Ce système de score est appliqué à toutes les dimensions évaluées dans le quiz, qu'il s'agisse de la couleur du vin, de ses arômes, de son prix ou des mets avec lesquels il s'accorde.

Une fois l'ensemble des réponses prises en compte et les points attribués, l'algorithme trie les vins selon leur score total et sélectionne les vingt vins ayant obtenu les meilleures notes, considérés comme étant les plus susceptibles de plaire à l'utilisateur. Ces recommandations sont ensuite enregistrées dans notre base de données en étant associées à l'identifiant de l'utilisateur. Cela permet à notre interface, notamment le carrousel présent sur la page d'accueil, d'effectuer une requête SQL pour récupérer automatiquement les recommandations personnalisées, que l'utilisateur peut ainsi retrouver de manière simple et autonome.

Chapitre 5

Machine Learning

5.1 Pourquoi un algorithme de Machine Learning

Dans le cadre de notre projet, nous avons choisi d'implémenter un algorithme de machine learning afin d'offrir une expérience de recommandation personnalisée. L'objectif principal est de prédire, à partir d'un texte en langage naturel fourni par l'utilisateur, les associations possibles entre des vins et des aliments. Plus précisément, le système doit être capable de suggérer un vin qui se marie bien avec un plat mentionné, ou inversement, de recommander des aliments qui s'accordent avec un vin donné.
Le machine learning est utilisé pour apprendre à partir de notre base de données de vins et leurs associations connues avec différents aliments. En analysant ces données, l'algorithme sera capable d'identifier des correspondances pertinentes et de les généraliser. Ainsi, lorsqu'un utilisateur interagit avec notre interface via le chatbot, le système peut lui proposer des recommandations cohérentes et personnalisées, même pour des combinaisons qui ne sont pas explicitement présentes dans la base initiale.

5.2 Notre premier modèle - LSTM

Nous avons dans un premier temps choisi un modèle LSTM (Long Short-Term Memory) implémenté avec PyTorch. Il nous a paru le plus adapté pour résoudre notre problème de recommandation autour des accords mets-vins.
Notre modèle est défini de la forme suivante :

$$\text{logits}_i = \text{FC}(\text{LSTM}(\text{Embedding}(x_i))) \quad \text{pour } i = 1, \ldots, n$$

Avec :

- x_i une séquence de mots encodée à partir d'une entrée convertie en mots clés (par exemple : "vin chardonnay type white grapes pinot" ou "aliment beef").
- Chaque mot est converti en entier à l'aide d'un vocabulaire, puis transformé en vecteur via une couche d'embedding.
- Ces vecteurs sont ensuite traités par une couche LSTM (Long Short-Term Memory) qui prends en compte l'ordre des mots dans l'entrée.
- Le dernier état caché de la dernière couche LSTM est passé à une couche linéaire pour produire les sorties (vecteur de scores logits qui donnent un score pour chaque classe aliment ou vin).

Le modèle LSTM, bien qu'adapté au traitement de données textuelles, s'est révélé très rapide lors des prédictions, mais peu précis dans les résultats obtenus. Cela s'explique principalement par le fait que ce type de modèle n'est pas bien adapté à la nature de nos données. Les LSTM sont conçus pour capturer des relations séquentielles ou temporelles (analyse de mouvements bancaire suspects, transformation d'audio en texte...), ce qui est moins pertinent dans notre cas, où les descriptions des vins et leurs attributs sont courtes et propre a chaque vin. De plus,

l'absence de word embeddings adaptés a également pénalisé les performances des prédictions. Notre base de données contient de nombreux mots rares, notamment des noms de vins spécifiques. Ces mots sont peu reconnus ou mal compris par les vecteurs classiques, ce qui complique la recherche de bonnes associations entre vins et aliments.

5.3 Vers un modèle plus performant : Random Forest multi-label

Nous avons donc opté pour un modèle plus adapté a nos données et qui, avec de l'optimisation, pourrait combiner précision et rapidité d'execution des requêtes : un *Random Forest* combinée à une approche *multi-label*. Ce modèle permettra de prédire plusieurs classes simultanément, ce qui est parfaitement adapté à notre tâche : recommander plusieurs aliments pour un vin donné ou plusieurs vins adaptés a un plat ou aliment.

Le modèle est défini mathématiquement comme suit :

$$\hat{\mathbf{y}} = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})]$$

où :

- x est le vecteur de caractéristiques (type de vin et cépages, ou aliment),
- $f_j(\cdot)$ est un classificateur Random Forest entraîné pour la j^{e} classe,
- $\hat{\mathbf{y}} \in \{0, 1\}^k$ est le vecteur de sorties indiquant les classes (aliments ou vins) prédites,
- k est le nombre total de classes (aliments ou vins selon la direction de la prédiction).

Concrètement, nous utilisons une structure `MultiOutputClassifier` du module `scikit-learn`, qui entraîne un modèle Random Forest par classe cible. Chaque forêt prédit si l'entrée $\mathbf{x}$ correspond ou non à cette classe.

Ce modèle a significativement amélioré les performances de prédiction, comme le montrent nos métriques d'évaluation.

```
  "debug_info": {
    "input_search_term": "merlot",
    "top_predictions": [
      {
        "item": "poultry",
        "probability": 0.6674760244115083
      }
/* ------ Prediction aliment --> vin ------ */
      {
        "item": "chateau la canorgue luberon blanc",
        "probability": 0.69
      },
```

5.3.1 Passage a XGBoost

Cependant, en pratique, nous avons constaté que les prédictions du modèle `Random Forest` étaient particulièrement lentes (1min 30sec), ce qui rendait son utilisation peu compatible avec une expérience utilisateur fluide, notamment dans le cas de notre chatbot.
En discutant de cette problématique avec nos encadrants, il nous a été suggéré d'explorer l'utilisation du modèle `XGBoost` en version `multi-label`. Ce choix nous a semblé pertinent car d'une part, XGBoost est réputé pour sa rapidité d'entraînement (et de prédiction), souvent supérieure à celle de l'algorithme Random Forest, tout en conservant une précision identique.

Pour effectuer ce changement de modèle entre le `Random Forest (multi-label)` et `XGBoost (multi-label)`, nous avons centralisés les hyperparamètres pour les faire varirer rapidement via un fichier `config.py` (exemple ci-dessous).

```python
XGB_PARAMS = {
    'n_estimators': 100,
    'max_depth': 6,
    'learning_rate': 0.1,
    'subsample': 1.0,
    'colsample_bytree': 1.0,
    'random_state': 0,
    'tree_method': 'hist',     # Methode optimisee pour des datasets volumineux
    'device': 'cuda'           # Acceleration via GPU (W11 -> RTX 4060)
}
```

Ensuite, nous avons simplement remplacé le `RandomForestClassifier` par le `XGBClassifier` en l'encapsulant dans une structure `MultiOutputClassifier`, ce qui nous permet de conserver la classification multi-label, un aspect essentiel compte tenu de la nature de nos données.

```python
from xgboost import XGBClassifier
from sklearn.multioutput import MultiOutputClassifier
from config import XGB_PARAMS

# Entrainement du modele Vin -> Aliment
wine_to_food_model = MultiOutputClassifier(XGBClassifier(**XGB_PARAMS))
wine_to_food_model.fit(X_wine, Y_wine)

# Entrainement du modele Aliment -> Vin
food_to_wine_model = MultiOutputClassifier(XGBClassifier(**XGB_PARAMS))
food_to_wine_model.fit(X_food_occ, Y_food_occ)
```

L'entraînement sur les prédictions `vin` → `aliments` s'avère relativement long, mais le modèle parvient à bien apprendre, notamment grâce à un nombre conséquent d'exemples (plus de 12 000).

Table 5.1: Performances vin → aliments

Métrique	**Valeur**
precision_micro	0.8734746
precision_macro	0.6231063
recall_micro	0.8854773
recall_macro	0.5612745
f1_micro	0.8794350
f1_macro	0.5668385
subset_accuracy	0.7156486
hamming_loss	0.0258950
jaccard_micro	0.7848139
jaccard_macro	0.4864771

En revanche, pour les prédictions `aliment` → `vins`, le modèle rencontrait des difficultés à identifier des similarités pertinentes, ce qui limitait son apprentissage.

Table 5.2: Performances aliment → vins

Métrique	**Valeur**
precision_micro	1.0
precision_macro	1.0
recall_micro	1.0
recall_macro	1.0
f1_micro	1.0
f1_macro	1.0
subset_accuracy	1.0
hamming_loss	0.0
jaccard_micro	1.0
jaccard_macro	1.0

Pour remédier à cela, nous avons réalisé qu'il était nécessaire de considérer chaque occurrence du CSV comme un exemple distinct — ce qui n'était pas le cas initialement. L'idée était de faire comprendre au modèle qu'un vin donné, caractérisé par certains attributs (nom, type, cépage), est associé à des aliments spécifiques. Cela permet ensuite au modèle de généraliser, en apprenant qu'un autre vin aux caractéristiques similaires pourrait également bien s'accorder avec ces mêmes aliments.

```
food_occurrences = []
wine_occurrences = []
for entry in data:
    wine_lower = entry['wine'].lower()
    for food in entry['foods']:
        food_occurrences.append(food.lower())
        wine_occurrences.append(wine_lower)

# Encodage one-hot pour le modele Aliment -> Vin
X_food_occ = np.zeros((len(food_occurrences), len(food_label_vocab)), dtype=
    int)
for i, food in enumerate(food_occurrences):
    if food in food_label_vocab:
        X_food_occ[i, food_label_vocab[food]] = 1

Y_food_occ = np.zeros((len(wine_occurrences), len(wine_label_vocab)), dtype=
    int)
for i, wine in enumerate(wine_occurrences):
    if wine in wine_label_vocab:
        Y_food_occ[i, wine_label_vocab[wine]] = 1
```

Cette approche a permis au modèle de mieux établir des liens entre les caractéristiques d'un vin et les aliments compatibles, améliorant ainsi sa capacité à prédire des accords pertinents.

5.4 Optimisation - Grid Search

Afin d'améliorer d'avantage notre modèle, nous avons réalisé une recherche par `Grid Search`, visant à explorer différentes combinaisons d'hyperparamètres pour notre modèle `XGBoost (multi-label)`. Cette optimisation a pour objectif principal de réduire le temps de prédiction. En particulier dans le cas `aliments` → `vins` tout en maintenant une précision élevée, afin de garantir une expérience utilisateur optimale.

Ne sachant pas sur quels hyperparamètres nous baser pour le `Grid Search` nous avons demandé a un LLM de nous conseiller une série d'hyperparamètres a tester en cohérence avec nos données et ce que nous voulions en faire. Il nous a été conseillé d'entrainer nos modèles sur les combinaisons suivantes :

```
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [3, 6, 9],
    'learning_rate': [0.1, 0.05]
}
```

Pour chaque combinaison, un appel à l'endpoint `/train_hyper` (Flask) permet de relancer l'entraînement avec les hyperparamètres en cours. Ensuite, le script mesure le temps nécessaire pour exécuter une prédiction (par exemple pour la requête `"aliment beef"`).

Voici un extrait de code où l'on mesure le temps d'exécution de la requête la plus longue, afin de trouver la combinaison d'hyperparamètres avec le meilleur équilibre entre précision et rapidité.

```
start_time = time.time()
pred_response = requests.post(PREDICT_URL, json={"query": "aliment beef"})
elapsed_time = time.time() - start_time
print(f"Params: {current_params} -> Temps de prediction: {elapsed_time:.2f}
    sec")
```

À l'issue de l'entraînement de nos deux modèles (`aliment/vin, vin/aliment`) sur l'ensemble des combinaisons d'hyperparamètres (plus de 48 heures d'entraînement continu sur GPU), voici les résultats obtenus :

Figure 5.1: Résultats détaillés du Grid Search

Hyperparamètres	Temps (s)	F1_macro	F1_micro	Subset Acc.	Jaccard_micro
n_estimators=50, max_depth=3, learning_rate=0.1	45.57	0.41224	0.83601	0.56311	0.71823
n_estimators=50, max_depth=3, learning_rate=0.05	54.67	0.36258	0.80786	0.50721	0.67765
n_estimators=50, max_depth=6, learning_rate=0.1	53.90	0.43739	0.85861	0.64020	0.75225
n_estimators=50, max_depth=6, learning_rate=0.05	53.44	0.40441	0.84833	0.59789	0.73661
n_estimators=50, max_depth=9, learning_rate=0.1	**53.80**	**0.46627**	**0.86438**	**0.66472**	**0.76116**
n_estimators=50, max_depth=9, learning_rate=0.05	47.85	0.41496	0.85656	0.62492	0.74910
n_estimators=100, max_depth=3, learning_rate=0.1	74.38	0.42299	0.84788	0.60066	0.73592
n_estimators=100, max_depth=3, learning_rate=0.05	86.87	0.40847	0.83408	0.55762	0.71538
n_estimators=100, max_depth=6, learning_rate=0.1	88.91	0.46593	0.86456	0.65354	0.76144
n_estimators=100, max_depth=6, learning_rate=0.05	85.39	0.43725	0.85842	0.63943	0.75196
n_estimators=100, max_depth=9, learning_rate=0.1	68.96	0.49428	0.86888	0.68205	0.76816
n_estimators=100, max_depth=9, learning_rate=0.05	89.43	0.46510	0.86396	0.65005	0.76050
n_estimators=150, max_depth=3, learning_rate=0.1	93.49	0.43151	0.85425	0.61461	0.74558
n_estimators=150, max_depth=3, learning_rate=0.05	120.55	0.41728	0.84317	0.57917	0.72887
n_estimators=150, max_depth=6, learning_rate=0.1	113.64	0.47140	0.86818	0.67626	0.76706
n_estimators=150, max_depth=6, learning_rate=0.05	114.75	0.46300	0.86172	0.64677	0.75704
n_estimators=150, max_depth=9, learning_rate=0.1	116.75	0.49826	0.87120	0.69129	0.77179
n_estimators=150, max_depth=9, learning_rate=0.05	121.06	0.46982	0.86743	0.67780	0.76590

Une explication détaillée de notre démarche de `Grid Search` est disponible sur notre GitHub.

5.4.1 Entraînement GPU avec CUDA

Pour accélérer considérablement les entraînements, nous avons configuré XGBoost afin d'utiliser le GPU. En effet, grâce à l'architecture CUDA de Nvidia, le modèle exploite les capacités de calcul parallèle de la carte graphique (ici, une RTX 4060) pour réduire le temps d'entraînement. Les hyperparamètres spécifiques pour cette accélération incluent :

```
XGB_PARAMS = {
    'n_estimators': 100,
    'max_depth': 6,
    'learning_rate': 0.1,
    'subsample': 1.0,
    'colsample_bytree': 1.0,
    'random_state': 0,
    'tree_method': 'hist',     # Utilisation d'une methode d'histogramme
       optimisee
    'device': 'cuda'           # Exploitation de CUDA pour l'entrainement sur
        GPU !
}
```

Cette configuration permet de réduire considérablement le temps de traitement même pour des entraînements avec beaucoup de donénes, rendant ainsi le grid search plus rapide malgré le grand nombre d'entrainement à réaliser.

La mise en œuvre du grid search, a été une étape cruciale dans notre recherche optimisation. En exploitant la puissance du GPU grâce à CUDA, nous avons pu explorer efficacement une série d'hyperparamètres pour notre modèle XGBoost multi-label. Ce processus automatisé nous a permis d'identifier la configuration optimale qui minimise le temps de réponse pour les prédictions `aliments` $\rightarrow$ `vins`. Après une analyse approfondie des différentes combinaisons, les meilleurs hyperparamètres sont les suivants :

- n_estimators = 50
- max_depth = 9
- learning_rate = 0.1

Bien que cette configuration ne soit pas la plus rapide sur le papier, elle offre le meilleur compromis entre rapidité d'exécution (40sec) et précision (0,87, pertes : 0,02) des prédictions. Ainsi, les temps de prédiction sont suffisamment courts pour être exploitables en situation réelle, sur notre site GrapeMind garantissant une réponse dans un temps raisonnable aux utilisateurs.

5.5 Traitement des requêtes utilisateurs

Le traitement des requêtes utilisateurs repose sur une API Flask qui interagit avec deux éléments essentiels : une API de machine learning pour générer des recommandations, et l'API Mistral pour prétraiter les requêtes en langage naturel.

5.5.1 *Protection des informations sensibles avec* `.env`

Afin d'éviter la divulgation des clés API et des identifiants de base de données sur GitHub ou involontairement, nous avons fait le choix de charger les variables d'environnement via le fichier `.env`. Ce fichier contient notamment la clé d'accès à l'API Mistral ainsi que les paramètres de connexion a notre base de données MySQL.

```
from dotenv import load_dotenv
import os
load_dotenv(dotenv_path="../.env")
api_key = os.getenv("MISTRAL_APIKEY")
# Recuperation des identifiants de connexion a la DB
conn = mysql.connector.connect(
    host=os.getenv("DB_HOST"),
    user=os.getenv("DB_USER"),
    password=os.getenv("DB_PASSWORD"),
    database=os.getenv("DB_NAME")
)
```

5.5.2 *Reformulation des requêtes utilisateurs via l'API Mistral*

L'API Mistral est utilisée pour convertir les messages des utilisateurs en une sortie structurée, composée de mots-clés exploitables par notre algorithme de machine learning. Cette étape permet de classer la demande en deux cas possibles : soit l'utilisateur fournit un aliment et souhaite une prédiction de vins, soit il décrit un vin et recherche une prédiction sur des aliments.

```
def simplify_text_with_mistral(api_key, user_input, preprompt):
    client = Mistral(api_key=api_key)
    response = client.chat.complete(
        model="mistral-small-latest",
        messages=[
            {"role": "system", "content": preprompt},
            {"role": "user", "content": user_input}
        ]
    )
    if response.choices:
        return response.choices[0].message.content.strip()
    return "Aucune reponse API"
```

La variable `preprompt` contient un prompt donné au LLM de Mistral qui donne les instructions afin de classer la demande de l'utilisateur dans la bonne catégorie et utiliser uniquement des mots clés que l'algorithme de Machine Learning pourra comprendre. Elle impose un format strict et attend des sorties comme `aliment chicken` ou `vin bordeaux type red grapes merlot`.

5.5.3 *Traduction automatique des aliments (vin → aliment)*

Lorsque le modèle de machine learning retourne des noms d'aliments en anglais, l'API Mistral est de nouveau appelée pour traduire `foods_list` en français avant d'afficher les aliments prédits à l'utilisateur.

```
def translate_food(api_key, foods_list):
    food_items = ", ".join(foods_list)
    translation_prompt = f"""Traduis les aliments suivants de l'anglais vers le francais.
Renvoie uniquement les noms traduits, sans explication.
Voici les aliments a traduire: {food_items}"""

    client = Mistral(api_key=api_key)
    response = client.chat.complete(
        model="mistral-small-latest",
        messages=[
            {"role": "system", "content": "..."},
            {"role": "user", "content": translation_prompt}
        ]
    )
    if response.choices:
        translated_text = response.choices[0].message.content.strip()
        return [item.strip() for item in translated_text.split(',')]
    return foods_list
```

5.5.4 *Appel de l'algorithme de prédiction*

Une fois la requête transformée en suite de mots clés, elle est envoyée à l'API Flask (Model XGBoost) de prédiction via une requête :

```
ml_output = requests.post(
    'http://51.210.243.151:5000/predict',
    json={"query": user_input}
)
```

Selon le type de la requête (`vin` ou `aliment`), la réponse est affichée soit sous forme de suggestions d'aliments (traduits en français), soit sous forme d'une liste de vins avec des liens cliquables redirigeant vers la fiche de détail du vin.

Chapitre 6

Difficultés rencontrées

6.1 Timeout des requêtes Flask

En raison des temps d'entraînement particulièrement longs générés par certaines configurations d'hyperparamètres, nous avons constaté que le serveur Flask se déconnectait ou redémarrait automatiquement (par exemple via le `reloader`) lors de l'entraînement. Dans un premier temps, nous avons fixé le `reloader` à `False`, ce qui a temporairement résolu le problème. Pour pallier définitivement ce problème, nous avons externalisé le `Grid Search` dans un fichier indépendant (`grid_search.py`). Cette solution a permis d'isoler et d'automatiser le processus de Grid Search sans interrompre le fonctionnement de l'API Flask. Chaque combinaison d'hyperparamètres est testée en relançant l'entraînement via un endpoint différent d'un entraînement « classique » (à savoir `/train_hyper`), ce qui garantit que le `Grid Search` est redémarré avec les nouvelles configurations à chaque itération, sans risque de déconnexion.

6.2 Traitement du fichier CSV pour l'entrainement

Afin d'éviter toute interruption liée à une requête SQL lors de l'entraînement du modèle, nous avons choisi d'extraire les données une seule fois sous forme d'un fichier CSV. Cela nous permet de travailler localement et exclusivement sur ce fichier pour l'ensemble des phases d'entraînement. Cette approche nous a également permis de prétraiter les données en amont : nous avons notamment supprimé les accents et caractères spéciaux présents dans les noms de vins et de domaines, afin de garantir une meilleure compatibilité avec les étapes suivantes du pipeline d'entraînement.

6.3 Problèmes liés à l'adaptabilité du site (responsive)

Nous avons aussi rencontré des problèmes d'alignement, de redimensionnement d'images, de scroll horizontal involontaire, ou encore de textes qui débordaient en dehors des blocs. Un point particulièrement bloquant a été l'utilisation de plusieurs éléments en `position: absolute`, notamment pour positionner des barres de recherche ou des boutons sur certaines images. Bien que cette méthode fonctionne sur grand écran, elle devient rapidement instable sur des résolutions réduites : les éléments se superposaient, sortaient du cadre ou devenaient inaccessibles sur mobile. Nous avons dû repenser ces positionnements en adoptant des approches plus flexibles, comme l'utilisation de `flexbox`, ou de positionnements relatifs, associés à des `media queries` pour adapter les tailles, espacements et ordres d'affichage en fonction des écrans.

6.4 Problèmes liés aux liens de redirection entre fichiers

Lorsque nous avons commencé l'hébergement de notre site chez notre premier hébergeur, Infomaniak, nous avons rencontré un problème majeur : l'intégralité du site ne fonctionnait plus. Ce dysfonctionnement était dû au fait que les chemins des liens absolus et relatifs ne correspondaient pas à la structure de l'hébergeur, dont la hiérarchie des fichiers était différente de celle de notre environnement local. Nous avons donc été contraints de reprendre tous les fichiers du projet pour modifier manuellement les chemins d'accès, afin de les adapter à cette nouvelle organisation.

Cette adaptation a cependant engendré un second problème. Une fois les chemins modifiés, il nous était devenu impossible de lancer le site en local, puisque les chemins utilisés n'étaient plus compatibles avec notre arborescence de développement. Pour contourner cette difficulté, nous avons créé une nouvelle branche GitHub, dédiée exclusivement au déploiement. Dans cette branche, les chemins étaient spécifiquement ajustés pour correspondre aux exigences de l'hébergeur.

Cette organisation a considérablement alourdi notre charge de travail. À chaque ajout de fonctionnalité ou mise à jour, nous étions obligés de faire un double travail : un premier push sur la branche de développement pour le fonctionnement local, puis un second sur la branche de déploiement avec les chemins modifiés. Ce processus, bien que fonctionnel, s'est révélé contraignant et source potentielle d'erreurs.

6.5 Obtention des données

Pour obtenir la première partie de nos données, nous avons pris contact avec Rogerio Xavier, professeur d'université à Rio de Janeiro, qui disposait d'une base de données très fournie dans le cadre d'un projet académique. Toutefois, le site hébergeant ses données fonctionnait mal et la base mise en ligne était incomplète. Nous avons donc dû échanger directement avec lui afin qu'il nous transmette personnellement l'intégralité de sa base.

Pour la seconde partie, nous avons eu recours à une technique de scraping afin de récupérer des données plus précises, telles que les prix, les arômes ou encore d'autres caractéristiques fines des vins. Cette méthode, bien qu'efficace, comporte plusieurs inconvénients. Elle implique notamment un changement régulier d'adresse IP afin d'éviter les bannissements mis en place par certains sites. Le processus de scraping s'est également révélé extrêmement long, avec de nombreux bugs et crashs, ce qui nous a obligés à repenser notre stratégie : nous avons mis en place un système de sauvegarde par tranches afin de limiter la perte de données. Enfin, certaines des informations extraites étaient erronées, en raison d'un mauvais encodage ou d'une lecture incorrecte par le programme, ce qui a nécessité un travail de nettoyage et de vérification manuelle.

Chapitre 7

Conclusion et perspectives

Le projet *GrapeMind* nous a permis de concevoir une plateforme web innovante, alliant data science, design interactif et découverte œnologique. Tout au long du semestre, nous avons travaillé sur des fonctionnalités variées : recommandations personnalisées, carte interactive des domaines viticoles, chatbot, recherche par plat, gestion de compte, et bien d'autres. Notre objectif était clair : rendre le monde du vin plus accessible, plus intuitif et plus connecté aux attentes réelles des utilisateurs.

Ce projet nous a permis d'apprendre énormément, notamment en développant des compétences "sur le tas" sur des aspects techniques que nous ne maîtrisions pas au départ, comme l'intégration de scripts Python, la gestion des bases de données, l'appel d'APIs, ou encore l'hébergement de services sur des serveurs web. Nous avons aussi été confrontés à des contraintes techniques concrètes, en particulier autour de l'hébergement, qui nous ont poussés à chercher des solutions alternatives et à gagner en autonomie.

Nous avons également recueilli plusieurs retours utilisateurs, qui nous ont permis d'identifier des pistes claires d'amélioration pour la suite du projet. Parmi les perspectives d'évolution envisagées :

- Ajouter des vins provenant d'autres pays pour enrichir la base de données ;
- Intégrer un rayon géographique afin de favoriser une consommation locale ;
- Proposer des vins bio et permettre un filtrage spécifique ;
- Améliorer l'algorithme de recommandation pour le rendre plus précis ;
- Revoir le responsive design de certaines pages encore limitées.

En définitive, *GrapeMind* a été bien plus qu'un exercice académique. Ce projet nous a permis de relever des défis concrets, d'explorer de nouvelles technologies et de collaborer autour d'une idée originale qui nous tenait à cœur. Nous avons construit une plateforme fonctionnelle et évolutive, tournée vers l'utilisateur, tout en développant des compétences utiles et transférables pour nos futurs projets professionnels.

Bibliographie

PHP Group. (2024). *PHP Manual*. https://www.php.net/manual/
Synchro. (2024). *PHPMailer – A full-featured email creation and transfer class for PHP*. https://github.com/PHPMailer/PHPMailer
Agafonkin, V. (2024). *Leaflet.js – JavaScript library for interactive maps*. https://leafletjs.com/
Jouglet, D. (2024). *Leaflet.markercluster plugin*. https://github.com/Leaflet/Leaflet.markercluster
DeepL GmbH. (2024). *DeepL API Documentation*. https://www.deepl.com/docs-api
The jQuery Foundation. (2024). *jQuery API Documentation*. https://api.jquery.com/
GitHub Inc. (2024). *GitHub – Collaborative code hosting platform*. https://github.com/
Lamport, L. (1994). *LaTeX: A Document Preparation System*. Addison-Wesley.
OpenAI. (2024). *ChatGPT – Large Language Model by OpenAI*. https://openai.com/chatgpt
Mistral AI. (2024). *Mistral – Open-weight Language Models*. https://mistral.ai
Stack Overflow. (2024). *Stack Overflow – Developer Community*. https://stackoverflow.com/