Sommaire

Liste des tableaux

Table des figures

Remerciements

En tout premier lieu, nous remercions Allah, le tout puissant, de nous avoir donné la force de nous surpasser.
Notre plus grande gratitude va à notre encadreur, pour sa disponibilité, sa rigueur ainsi que la confiance qu'il nous a accordée. Nous avons profité d'une année riche en savoir auprès de lui et de notre co-promoteur qui nous ont consacré leur temps et leur énergie. Nous aimerions aussi les remercier pour l'autonomie et l'assurance qu'ils nous ont permis d'acquérir, ainsi que pour les précieux conseils qu'ils nous ont prodigués afin de mener à bien ce travail.
Nous remercions Monsieur le Professeur H.AZZOUNE de nous faire l'honneur d'être notre président de jury et Madame la Docteure F.BELLALA celui d'être l'examinatrice de notre travail, pour le temps et l'effort qu'ils nous auront accordé. Afin de n'oublier personne, nos vifs remerciements s'adressent à tous ceux qui nous ont aidés à la réalisation de ce mémoire.

Nous voudrions exprimer notre reconnaissance envers nos amis qui nous ont apportés leur soutien moral et intellectuel tout au long de notre démarche. Un grand merci à Mérouane, Amine, Mouloud, Said, Boudjema ainsi qu'à tous nos amis du club Open Minds et pour tous ceux qui nous ont encouragés à persévérer, à repousser nos limites, et ont cru en nous. A nos parents pour leurs patience et bienveillance, témoignages quotidien de leur soutien indéfectible.

Introduction Générale

Face à une explosion de données, qui devrait atteindre les 44.000 milliards de gigaoctets en 2020, soit 10 fois plus qu'en 2013 selon un rapport du cabinet de recherche IDC [1], la détection de structures et liens particuliers, l'organisation et la recherche de connaissances exploitables dans cette masse d'informations deviennent un enjeu stratégique pour la prise de décision mais aussi à des fins de prédiction.

1.1 Problématique

Combien de temps un utilisateur perd-t-il à passer au crible ses fichiers numériques pour trouver une n-ième version sauvegardée dans le mauvais dossier, d'un brouillon, d'une vidéo ou d'une photo mal étiquetée ? Pour des millions de travailleurs à travers le monde, ces minutes s'accumulent et représente une perte de temps très pesante.

Selon un rapport publié en 2012 par IDC, les employés passent 4,5 heures par semaine en moyenne, à chercher les documents dont ils ont besoin au lieu d'exploiter ce temps à traiter ces derniers. Six ans plus tard, peu de choses ont changé. Si l'essor de l'informatique a rendu possible le stockage de volumes de données toujours plus importants à des coûts toujours plus faibles, l'utilisateur, submergé par cette masse d'informations, ne se pose plus la question d'accéder à l'information. Son problème devient : comment trouver l'information dont il a besoin parmi toutes celles qui lui sont accessibles. Ce problème est d'autant plus vrai sur sa propre machine où il accumule les documents sans plus se demander s'il a l'espace nécessaire.

Le cœur du problème est que le stockage efficace n'est pas forcément efficient car si l'émergence de l'informatique a permis à l'utilisateur d'avoir accès à près de 2.7 Zettaoctets [2]de données (quantité de données existant dans l'univers numérique aujourd'hui selon le cabinet IDC). Ceci ne l'empêche pas moins de se perdre dans ses propres documents qu'il accumule à travers le temps car tout comme avec les systèmes papier, le fait de nommer incorrectement un fichier ou de le sauvegarder au mauvais endroit peut signifier des heures passées à le chercher ou encore répéter le travail qui a déjà été fait.

1. IDC est un acronyme pour International Data Corporation ; c'est une entreprise américaine spécialisée dans la réalisation d'études de marché dans les domaines des technologies de l'information.

2. 1 $Zettaoctets = 10^{21}$ octets

Dans le but de faciliter l'interaction de l'utilisateur avec une masse de données toujours plus grande de nouveaux outils de gestions ont vu le jour, toutefois ces systèmes de gestions de fichiers souffrent d'insuffisances que nous tenterons de combler en présentant une nouvelle approche d'organisation de fichiers basée sur l'apprentissage automatique.

Ce qui nous intéressera particulièrement dans notre projet sera la Catégorisation de Textes (ou TC pour Text Categorization en Anglais) consistant à chercher une liaison fonctionnelle entre un ensemble de textes et un ensemble de catégories (étiquettes ou classes). Cette liaison fonctionnelle, que l'on appelle également modèle de prédiction, est estimée par un apprentissage automatique (Machine Learning en anglais). Pour ce faire, il est nécessaire de disposer d'un ensemble de textes préalablement étiquetés, dit ensemble d'apprentissage ou corpus, à partir duquel les paramètres du modèle de prédiction sont estimés, comme représenté dans le diagramme de la figure 1.1.

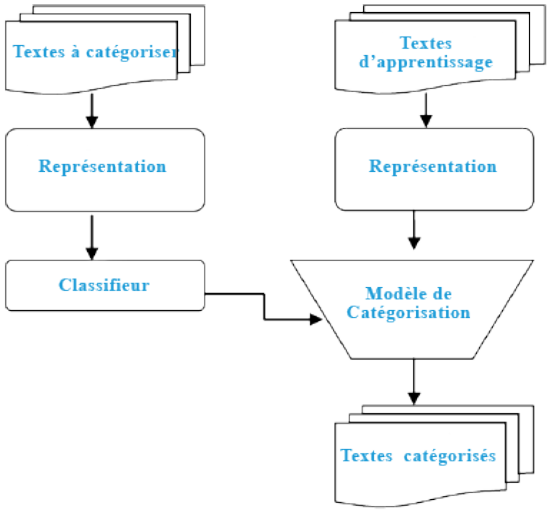


FIGURE 1.1 – Diagramme de catégorisation de texte

Ainsi, afin de permettre de répondre aux différentes problématiques, la TC comporte un choix de techniques d'apprentissage (ou classifieurs) dont les plus anciennes viennent du champ de la statistique et d'autres sont issues de l'Intelligence Artificielle.Parmi les méthodes d'apprentissage le plus souvent utilisées figurent l'analyse factorielle discriminante, la régression logistique, les réseaux de neurones, les plus proches voisins (KNN), les arbres de décision, les réseaux bayésiens et les machines à vecteurs supports (SVM).L'intérêt d'une telle démarche est d'organiser les connaissances de façon à pouvoir effectuer par la suite une recherche efficace.

Ce projet s'inscrit dans le cadre de la classification automatique de documents. Son but majeur est d'automatiser l'organisation des documents sur un ordinateur. Nous proposons ainsi une approche basée sur l'apprentissage automatique et la recherche heuristique afin de regrouper les documents de l'utilisateur en une structure en arbre, où chaque document se verra associer une catégorie regroupant ainsi les fichiers d'un utilisateur selon leurs contenus et facilitant, dans un deuxième temps,

la recherche d'un document quelconque.

La suite de ce mémoire est organisée en trois chapitres et une conclusion générale, dont le contenu général sera le suivant :

Dans le premier chapitre, à forte teneur introductive, nous plaçons de prime abord notre approche au sein du traitement de la langue naturelle, en y dégageant quelques principes directeurs. Dans un deuxième temps, toujours en prenant appui sur le TALN, nous préciserons notre approche avec une présentation générale de la catégorisation de textes : définitions, objectifs généraux et domaines d'application, et nous justifierons le recours à une nouvelle méthode d'organisation à travers une critique des systèmes existants où nous précisons les besoins d'une automatisation totale. Enfin, nous présenterons un nombre de méthodes de l'apprentissage automatique, précisant l'orientation de notre approche vers l'automatisation complète.

Dans le second chapitre, après une description des grands principes de la théorie, nous passons à une description détaillée du fonctionnement général et l'architecture du système, en abordant plus en détail les différents traitements que subit un document avant sa catégorisation ; il sera question de la préparation des différents éléments nécessaires au bon fonctionnement de la prédiction. Nous prendrons soin de séparer chaque étape de la catégorisation, en commençant par le prétraitement du document jusqu'à son affectation dans la catégorie correspondante.

Enfin, le chapitre 3 sera centré sur l'implémentation de notre solution, les tests effectués et résultats obtenus qui nous auront permis de décider parmi les différentes techniques qui s'offraient à nous au commencement celle qui satisferait au mieux nos besoins.
Nous terminerons ce mémoire par une conclusion générale où nous rappellerons notre contribution et présenterons certaines perspectives de développement de ce travail.

Chapitre 1 : Introduction

2.1 Les Systèmes de gestion de fichiers sur ordinateur

A l'ère du Big Data, où près de 29.000 Gigaoctets de données sont créés par seconde[1], le besoin d'organiser ses fichiers de manière à retrouver rapidement ce que l'on cherche s'est fait sentir. Les systèmes de gestion rencontrent ainsi un franc succès ; nous en présentons ici les plus répandus.

2.1.1 GNU/Linux

Le système d'exploitation GNU/Linux, organise les fichiers sous forme d'une seule et unique arborescence, chaque dossier de cette arborescence a un contenu précis : fichiers temporaires, de configurations, etc. L'inconvénient majeur de cette organisation des fichiers est que les fichiers personnels ne sont jamais catégorisés sémantiquement. Leur organisation peut être laissée au gré de l'utilisateur qui peut oublier avec le temps et le nombre croissant de fichiers qu'il a sur son ordinateur, où les retrouver.

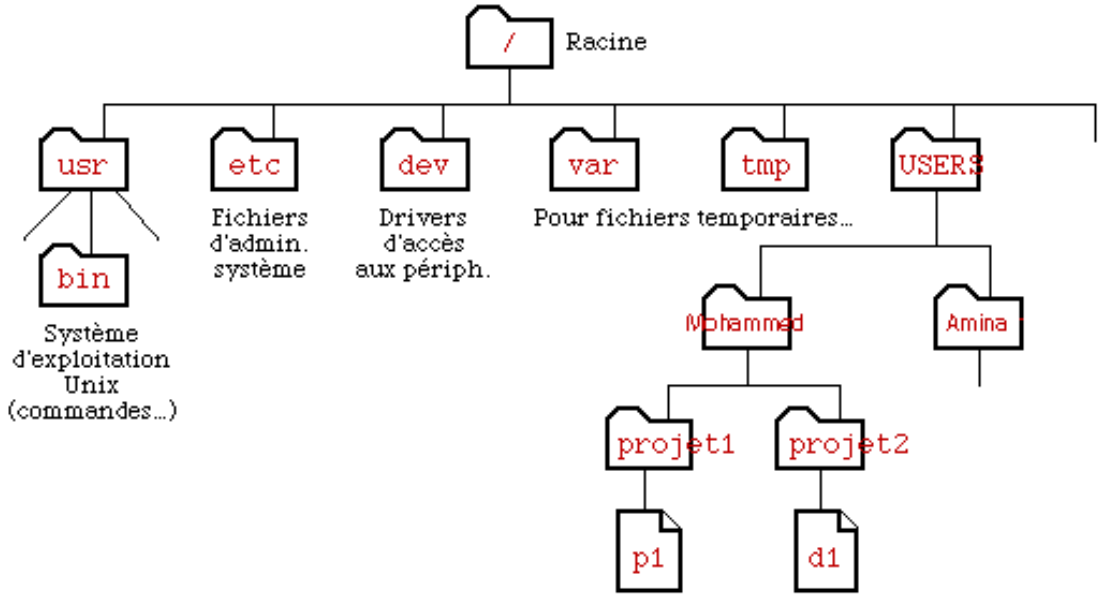


FIGURE 2.1 – Arborescence de Linux

2.1.2 Windows

Windows quant à lui organise tous les fichiers sous plusieurs arborescences (une pour chacune des partitions de l'utilisateur).
Les inconvénients de cette proche sont les suivants :

— L'installation des applications peut se faire n'importe où ; des fichiers de configuration peuvent se retrouver sauvegardés dans le mauvais répertoire ce qui perturbe l'organisation logique des fichiers.

— Aucune catégorisation sémantique n'est appliquée aux fichiers.

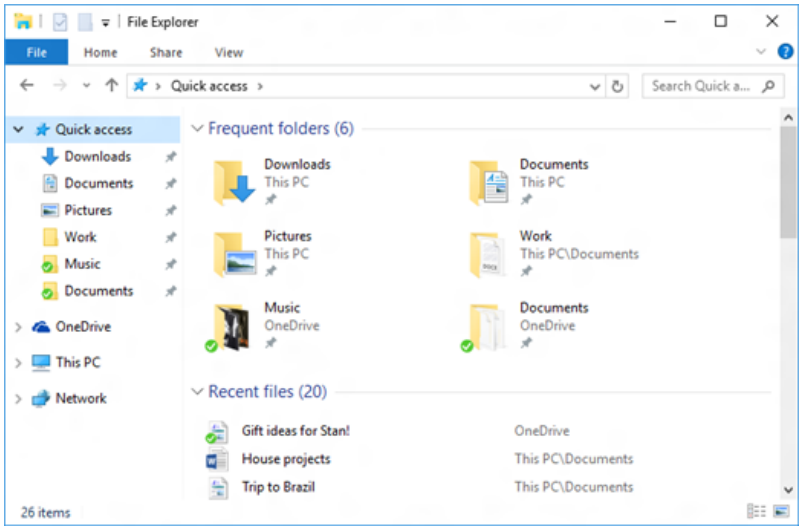


FIGURE 2.2 – Gestionnaire de fichiers Windows

2.1.3 Mendeley

Mendeley est avant tout un outil de gestion de références. Il permet néanmoins la gestion de documents et organise les fichiers de l'utilisateur en les regroupant d'après les métadonnées extraites de ces derniers et offre également la possibilité de lire et annoter ses documents. Mendeley représente l'un des plus populaires réseaux sociaux orientés vers la recherche scientifique qui permet de collaborer et de partager ses données avec des chercheurs du monde entier.

L'illustration de la figure 2.3 représente l'interface de travail sur Mendeley Desktop, par laquelle on peut aisément se rendre compte des fonctionnalités offertes par cette application.

Avantages :

— Accéder à ses données à distance grâce à la possibilité de les mettre sur le cloud.
— Collaborer avec d'autres personnes et partager ses données.

— Retrouver des articles en PDF d'autres utilisateurs ainsi que créer et intégrer des réseaux.
— Gérer ses références en créant et citant le contenu de ses bibliographies.
— Importer et organiser ses documents.

Désavantages :

— Mendeley est orienté vers la gestion de références bibliographiques.
— La bibliothèque de l'utilisateur est sauvegardée dans le cloud, permettant ainsi à l'application un accès non contrôlé aux documents.
— Le logiciel Mendeley n'est pas transparent : Mendeley n'étant pas un logiciel Open Source, l'utilisateur n'a aucun moyen de savoir avec certitude si ses documents sont protégés ou utilisés à son insu. Ceci est d'autant plus effrayant au vu des récents évènements comme le scandale Facebook-Cambridge Analytica [1], l'utilisateur ne peut plus se permettre le luxe d'avoir foi en des applications qui manipulent ses données personnelles.
— L'organisation des documents est non intelligente car elle ne se base que sur des métadonnées pour regrouper les documents en catégories et faciliter la recherche.
— L'application se limite aux documents PDF, sans tenir compte les autres types de fichiers.

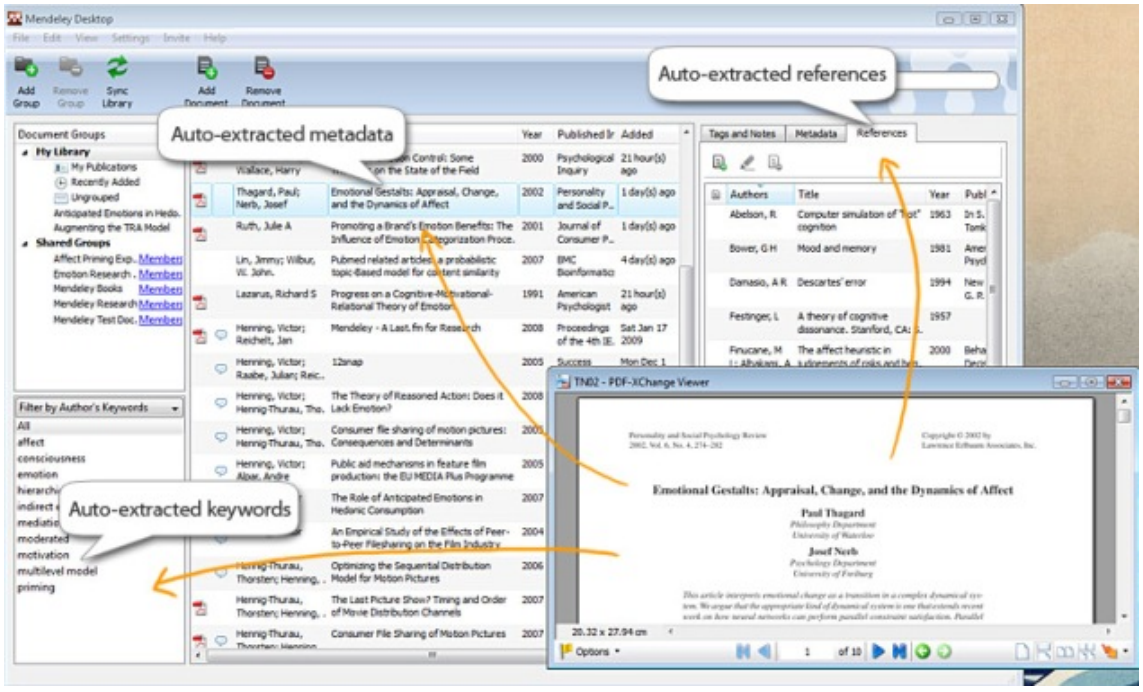


FIGURE 2.3 – Mendeley Desktop

1. Il s'agit du scandale où Facebook a mis à la disposition de Cambridge Analytica les données personnelles de près de 87 millions d'utilisateurs sans leur consentement et ces données ont été utilisées à des fins électorales

2.1.4 TagSpaces

TagSpaces est un logiciel Open Source, multiplateforme, sans backend. Il organise les fichiers, photos et autres documents avec des tags. Dit autrement, ce logiciel permet à l'utilisateur d'étiqueter manuellement ses fichiers à l'aide de tags afin de les retrouver plus facilement à l'avenir en ayant juste à entrer une liste de mots clés (correspondant aux tags), comme montré dans la capture d'écran de la figure 2.4.

Avantages :

— TagSpaces permet une organisation personnalisée et une recherche rapide et efficace des fichiers.
— Il permet l'accès à un même document au travers différents chemins.
— Pour trouver un fichier, il suffit de rechercher un ou deux mots-clés définissant la recherche. Durant la frappe, une proposition de tags adaptée à la recherche est faite.
— Les documents de l'utilisateur restent sur son ordinateur et ne sont pas déplacés dans le cloud à moins de sélectionner cette option.

Désavantages :

— TagSpaces ne propose pas une organisation intelligente car c'est à l'utilisateur d'étiqueter manuellement le contenu de son ordinateur, ses fichiers ne sont donc pas classifiés automatiquement par l'application.

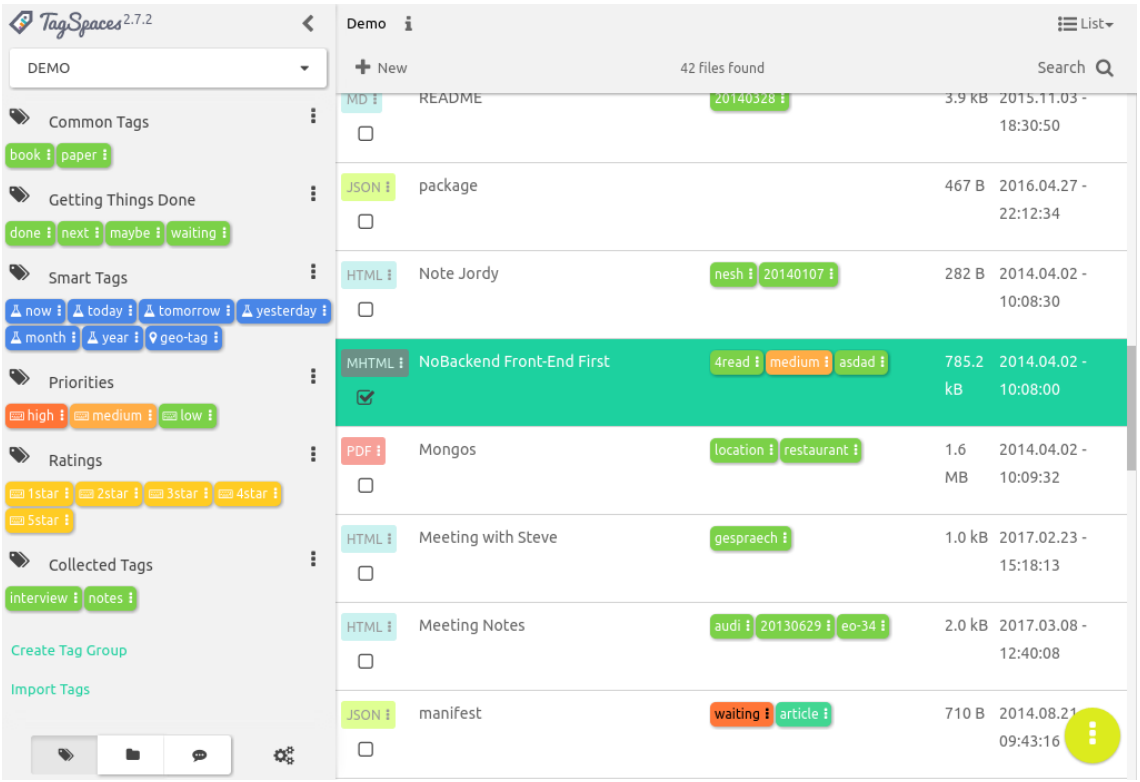


FIGURE 2.4 – TagSpaces

2.2 Problématique

Dans cette première partie nous avons pu constater la nécessité d'un gestionnaire de fichiers. Nous avons aussi révélé les lacunes dont souffrent les systèmes de gestion de fichiers actuels tels que le manque de transparence de certains, l'absence d'automatisation ou encore divers autres insuffisances qu'il reste à combler.

L'objectif de notre travail sera donc de concevoir un gestionnaire de fichiers automatisé et ainsi de simplifier et d'optimiser l'interaction de l'utilisateur avec ses fichiers en automatisant complètement l'organisation de ceux-ci.

C'est donc dans cette optique que notre application a vu le jour. Afin de simplifier l'expérience de l'utilisateur et son interaction avec ses documents, il serait très utile de développer un système qui permettrait de classifier ces derniers de manière automatique.

2.3 Traitement Automatique du Langage Naturel

Le traitement des données textuelles fait face à plusieurs problèmes, dû principalement au fait que le langage naturel (par opposition aux langages informatiques) n'est pas univoque : « un langage univoque est un langage dans lequel chaque mot ou expression a un seul sens, une seule interprétation possible et il n'existe qu'une seule manière d'exprimer un concept donné »[2].

Ainsi, une branche de recherche associant étroitement linguistes et informaticiens fût instaurée, afin de trouver des méthodes et des programmes qui permettent un traitement par ordinateur des données langagières ; cette branche est le Traitement Automatique du Langage Naturel (TALN) ou NLP (pour Natural Language Processing en anglais). On regroupe sous le vocable de traitement automatique du langage naturel, l'ensemble des recherches et développements qui étudient la compréhension, la manipulation et la génération du langage naturel par les machines. Parmi ses applications et domaines on retrouve : la génération de texte, le résumé automatique, l'identification de langage, etc.

2.3.1 Prétraitement du texte

Le prétraitement se divise en 2 sous-étapes. Premièrement, il s'agit d'éliminer tout symbole qui ne correspond pas à une lettre de l'alphabet (points, virgules, traits d'union, chiffres,etc.). Cette opération est motivée par le fait que ces caractères ne sont pas liés au contenu des documents et ne change rien au sens s'ils sont omis ; par conséquent, ils peuvent être négligés. Ensuite, vient le StopWordsRemoval qui correspond à la suppression de tous les mots qui sont trop fréquents (ils n'aident donc pas à distinguer entre les documents) ou jouent un rôle purement fonctionnel dans la construction des phrases (articles, prépositions, etc.). Le résultat du StopWordsRemoval est que le nombre de mots dans la collection, ce qu'on appelle la masse des mots, est réduit en moyenne de 50%[3]. Les mots à éliminer, connus

comme stopwords, sont récoltés dans la stoplist qui contient en général entre 300 et 400 éléments en langue anglaise.

2.3.2 Vectorisation de l'ensemble des mots (Tokenization)

Pour mettre en œuvre des méthodes de classification, il faut choisir un mode de représentation des documents, car les méthodes d'apprentissage ne sont toujours pas capables de représenter directement des données non structurées (textes). Un document (texte) d_i est représenté par un vecteur numérique de la façon suivante :

$$d_i = (V_{1i}, V_{2i}, ..., V_{|T|i})$$

où T est l'ensemble des termes (ou descripteurs) qui apparaissent au moins une fois dans le corpus. (|T| est la taille du vocabulaire), et V_{ki} représente le poids (ou la fréquence du terme ki).

La représentation la plus simple des documents textuels est appelée «représentation en sac de mots » (Bag of words, BoW) [4].Elle consiste à transformer des textes en vecteurs où chaque élément représente un mot.

2.3.3 Racinisation (Stemming)

Vient l'étape du stemming ou racinisation qui consiste à remplacer chaque mot du document par sa racine comme par exemple les mots : rationnel, rationalité et rationnellement, sont remplacés par leur racine «rationnel» et les verbes conjugués par leur infinitif (rationalisèrent devient « rationaliser»). La racinisation n'a aucun impact sur la masse des mots (nombre des mots), mais réduit de 30% en moyenne la taille du document [3].

2.3.4 Lemmatisation

Le processus de lemmatisation consiste à utiliser des règles grammaticales pour remplacer les mots par leurs formes canoniques. Les lemmes correspondent donc à la forme des mots du dictionnaire ; par exemple, la forme canonique d'un verbe est l'infinitif, et celle d'un adjectif est au masculin singulier. Cette étape est utilisée pour préparer la suivante qui est l'étape cruciale à savoir la vectorisation (numérisation).

2.4 Corpus

Un corpus est un ensemble de documents regroupés, prêts à l'usage pour les tâches de traitement du langage naturel ou d'apprentissage automatique. Ainsi on peut les retrouver pré-étiquetés ou non. Etiqueter un texte, c'est lui associer une ou plusieurs catégories (étiquettes) selon la thématique qu'il aborde.L'étiquetage peut être automatisé au moyen d'un programme qu'on appelle un étiqueteur (tagger), ou bien résulter d'une annotation manuelle faite par des experts, ou encore découler d'une combinaison des deux. Le jeu ou données d'apprentissage (en anglais « dataset ») est un élément essentiel au processus d'apprentissage : si cet ensemble

est inconsis -tant, le modèle d'apprentissage peut être de mauvaise qualité, un bon étiquetage est donc primordial au bon fonctionnement du système de classification automatique.

Plusieurs sites web proposent gratuitement des corpus d'apprentissage et de tests bien structurés pour réaliser des travaux portant sur la classification automatique des documents. Les plus importants sont : le corpus d'apprentissage Reuters 22173 (l'agence Reuters a proposé en 1987 un corpus de dépêches en langue anglaise) et le corpus 20-Newsgroups(ce corpus est constitué de messages échangés dans 20 forums de discussion).

2.4.1 DMOZ

DMOZ [5], pour Directory Mozilla aussi connu sous le nom de Open Directory Project (ODP) était un annuaire de sites web actif de 1998 à 2017, sous licence Open Directory. DMOZ était le plus large répertoire du web géré par une communauté d'éditeurs bénévoles, chacun étant responsable de catégoriser manuellement un site dans une ou plusieurs catégories. Ce répertoire est organisé en un peu plus de 1 million de catégories et regroupe près de 4 millions de sites internet dans déférentes langues (90 langues au total).

2.4.2 WordNet

WordNet [6] est une grande base de données lexicale de l'anglais basée sur des études linguistiques et développée à l'Université de Princeton. Les noms, les verbes, les adjectifs et les adverbes sont regroupés dans des ensembles de synsets (synonyms sets), chacun exprimant un concept distinct. Le synset (ensemble de synonymes) est la composante atomique sur laquelle repose WordNet. Ils sont reliés par des relations conceptuelles sémantiques et lexicales, qui sont : L'hyperonymie, l'hyperonymie, l'hyponymie, la méronymie et l'antonymie.

Prenons X et Y deux synsets.

- Hyperonymie : X « est-un » Y ; exemple :X= Blanc et Y=Couleur.
- Hyponymie : Est la relation inverse de de l'Hyperonymie, Y est la « spécialisation » de X ; exemple :X= Couleur et Y=Noir.
- Méronymie : X est une part de Y ; exemple :X= Feuille et Y=Arbre.
- Antonymie : Est une relation lexicale d'opposition, X est l'opposé de Y ; exemple :X= Bien et Y=Mal.

WordNet ressemble superficiellement à un dictionnaire, en ce qu'il regroupe les mots en fonction de leurs significations. Cependant, il y a quelques distinctions importantes. L'une d'entre elles est la séparation des données en quatre bases de données associées aux catégories de verbes, noms, adjectifs et adverbes. Chaque base de données est organisée différemment des autres.

2.4.2.1 Gloss de WordNet

Les "Gloss" sont des définitions courtes fournissant des significations appropriées des mots et donc des synsets entiers. Les annotations gloss couvrent également les concepts, les collocations (formes multi-mots).

2.4.2.2 WordNet Domain

WordNet Domains[7] est une ressource lexicale créée de manière semi-automatique en augmentant WordNet avec des étiquettes de domaine. Les syntets WordNet ont été annotés avec au moins une étiquette de domaine sémantique, sélectionnée parmi un ensemble d'environ deux cents étiquettes structurées selon une certaine hiérarchie.

2.5 Représentation numérique (Normalisation)

Les algorithmes d'apprentissage ont toujours des difficultés à traiter directement les textes et plus généralement, les données non structurées. C'est pourquoi il est nécessaire de passer par une étape préliminaire dite de représentation afin de rendre le texte exploitable par les algorithmes d'apprentissage. Cette étape consiste en la représentation de chaque document par un vecteur de n termes pondérés et permet donc de passer des documents au texte, du texte au nombre, du nombre à l'analyse et de l'analyse à la prise de décision.

2.6 Représentation de Document (Pondération)

La plupart des approches dans le domaine de la catégorisation de textes sont centrées sur la représentation vectorielle des textes. Pour ce faire, elles utilisent principalement les mesures TF- IDF et TF-IDF-CF.

2.6.1 Représentation par TF-IDF

Les termes les plus fréquents ne sont pas nécessairement les plus pertinents.Au contraire, les termes qui apparaissent fréquemment dans un petit nombre de documents mais rarement dans d'autres tendent à être plus pertinents et spécifiques pour ce groupe particulier de documents, et donc plus utiles pour trouver des documents similaires. Afin de capturer ces termes et de refléter leur importance, le schéma de pondération TF-IDF (fréquence des termes et inverse de la fréquence des documents) est très utilisé [8] :

TF « Term Frequency » : représente le nombre d'occurrences du terme dans le corpus.
IDF : représente le nombre de documents contenant le terme. Ces deux concepts sont combinés (par multiplication), en vue d'attribuer un plus fort poids aux termes qui apparaissent souvent dans un document et rarement dans l'ensemble du corpus.

$$W_{x,y} = tf_{x,y} \times \log(\frac{N}{df_x})$$

où x est un terme et y un document,
$tf_{x,y}$ = Fréquence de x dans y,
df_x = Nombre de document contenant x, et
N= Nombre total de documents.
Inconvénients :
Les documents plus longs ont typiquement des poids plus forts parce qu'ils contiennent plus de mots, auquel cas « TF» tend à être plus élevé.

2.6.2 Représentation par TF-IDF-CF

Pour pallier aux lacunes de TF-IDF, a été introduit un nouveau paramètre CF (class frequency) ou « fréquence de classe » qui calcule la fréquence d'un terme dans une classe définit comme étant les groupes distinguables de documents.

$$TF.IDF.CF = tf_{x,y} \times \log(\frac{N}{df_{x,y}}) \times \frac{n_{cx,y}}{N_{cx}}$$

où $n_{c_{x,y}}$ représente le nombre de documents de la classe C à laquelle appartient y, où le terme x apparait.
N_{c_x} représente le nombre de documents de la classe C à laquelle appartient y.

2.7 Mesures de Similarité

Les algorithmes de catégorisation nécessitent une métrique pour quantifier la différence entre deux documents donnés. La mesure de similarité est la mesure de la ressemblance de deux objets donnés. La mesure de similarité est une distance avec des dimensions représentant les caractéristiques des objets. Si cette distance est petite, le degré de similarité est élevé et inversement.

$$Similarite = \begin{cases} 1 & siX = Y \\ 0 & siX \neq Y \end{cases}$$

Dans ce qui suivra nous posons $\vec{t_a}$ et $\vec{t_b}$ des vecteurs de dimension m sur l'ensemble de termes $T = t_1, ...t_m$ où chaque dimension représente un terme avec son poids dans le document, qui est non négatif et délimité entre [0,1].

2.7.1 Distance Euclidienne

La distance euclidienne entre deux points dans le plan, ou tout espace multidimensionnel, mesure la longueur d'un segment reliant ces derniers. Autrement dit, elle est la distance minimale entre deux points.C'est la façon la plus évidente de représenter la distance entre deux points. La Figure 2.5 représente la distance Euclidienne entre 2 Documents.

La distance euclidienne est largement utilisée dans les problèmes de clustering, y compris le regroupement de texte. C'est la mesure de distance par défaut utilisée avec l'algorithme K-moyennes[9].

$$D_e\{\vec{t_a}, \vec{t_b}\} = \sqrt{\sum_{i=1}^{m} |t_{a_i} - t_{b_i}|^2}$$

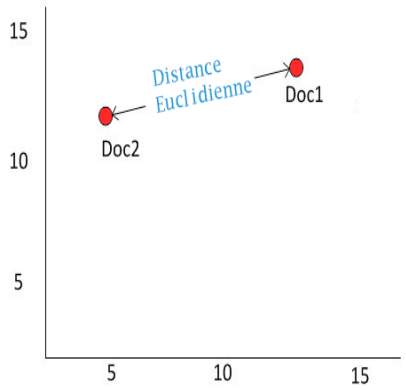


FIGURE 2.5 – Distance Eucledienne

2.7.2 Distance de Manhattan

La distance de Manhattan (ou distance rectilinéaire) est une métrique dans laquelle la distance entre deux points est la somme des différences absolues de leurs coordonnées cartésiennes. Autrement dit, la distance rectilinéaire relie deux points par deux droites perpendiculaires telles représentées dans la Figure 2.6 .Dans la plupart des cas, cette distance produit des résultats proches de ceux obtenus par la distance Euclidienne. Toutefois, avec cette mesure, l'effet d'un points aberrant est atténué et aura donc moins d'impact (car la différence n'est pas élevée au carré) .

$$D_m\{\vec{t_a}, \vec{t_b}\} = \sum_{i=1}^{m} |t_{a_i} - t_{b_i}|$$

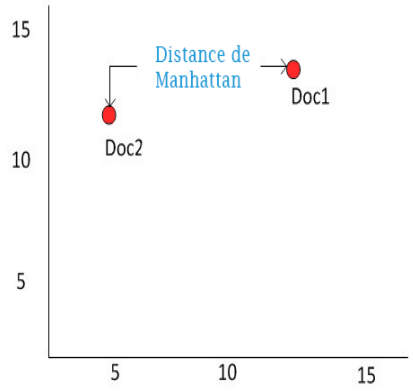


FIGURE 2.6 – Distance de Manhattan

2.7.3 Similarité Cosinus

Lorsque les documents sont représentés en tant que vecteurs de termes, la similarité de deux documents correspond au cosinus de l'angle entre les vecteurs (Figure 2.7). La similarité cosinus est l'une des mesures de similarité les plus populaires appliquées aux documents textuels dans le domaine de la catégorisation de textes.

$$\mathrm{SIM}_c\{\vec{t_a}, \vec{t_b}\} = \frac{\vec{t_a}\vec{t_b}}{|\vec{t_a}|\times|\vec{t_b}|}$$

Une propriété importante de la similarité cosinus est son indépendance de la longueur des documents. En d'autres termes, les documents ayant la même composition mais un total de mots différents seront traités de manière identique.

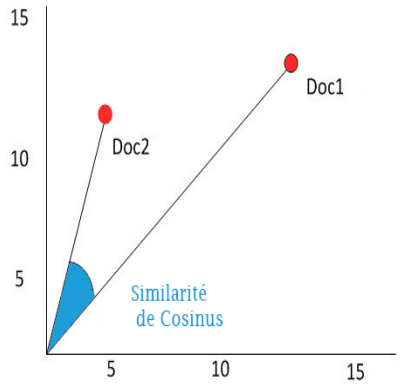


FIGURE 2.7 – Similarité Cosinus

2.8 Apprentissage Automatique

2.8.1 Définitions Générales

L'apprentissage est une problématique ancienne qui a été abordée dans de nombr -eux domaines. Le terme apprentissage ou apprentissage automatique ("machine learning", ou ML, en anglais) est un sous-domaine de l'Intelligence Artificielle et correspond à un ensemble de techniques aux bases mathématiques. Il s'agit plus précisément de leur capacité à organiser, construire et généraliser des connaissances, pour une utilisation ultérieure de leur capacité à tirer profit de l'expérience pour améliorer la résolution d'un problème.

Le concept du ML est le pouvoir de la machine d'effectuer de meilleures prédictions en se basant sur des données pertinentes, cumulées à travers le temps, permettant ainsi :

— La simplification de la résolution de bien des problèmes qui s'ils avaient été entamés sous le spectre de parcours itératifs ou récursifs auraient eu une grande complexité.
— Permettre d'entamer des problèmes qui seraient impossibles à résoudre sans passer par ces techniques.

Un des domaines d'application les plus répandus de l'apprentissage automatique est celui de la classification ; on prend en entrée une donnée pertinente x, on tente d'en sortir une prédiction correcte, qui sera représentée par une classe ou catégorie prédéterminée ou non (selon le type de technique choisie).

L'apprentissage automatique s'applique dans divers autres domaines, tel que la branche de la vision par ordinateur qui s'intéresse à la reconnaissance d'objets, de visages-mouvements et expressions- ou encore dans les prédictions financières, en passant par la catégorisation de textes.

2.8.2 Les types d'Apprentissage Automatique

Il existe trois modèles principaux d'apprentissage automatique chacun d'entre eux ayant ses utilisations propres : l'apprentissage supervisé, apprentissage non-supervisé, et l'apprentissage par renforcement.

2.8.2.1 Apprentissage Supervisé

Dans le cas de l'apprentissage supervisé, on dispose d'un ensemble de données étiquetées, ou entrées X qui se sont vues associées à une catégorie Y par un expert. Cet ensemble d'exemples constituera la base d'apprentissage. Les méthodes d'appre -ntissage supervisé se donnent alors comme objectif général de construire à partir de la base d'apprentissage, des classifieurs, ou fonctions de classement. Une telle fonction permet, à partir de la description d'un objet, de reconnaître un attribut particulier qui est la classe.

Une fois la prédiction déterminée, on dispose d'un étiquetage correct déjà stocké avec lequel on peut comparer notre prédiction. Si elle s'avère inexacte, on ajuste les paramètres d'apprentissage pour adapter le modèle d'apprentissage aux données réelles. Plus formellement, étant donné un ensemble de données D, décrit par un ensemble de caractéristiques X, un algorithme d'apprentissage supervisé va trouver une fonction de mapping qui fera le lien entre les variables prédictives en entrée X et la variable à prédire Y. La fonction de mapping décrivant la relation entre X et Y s'appelle un modèle de prédiction.

La catégorie de la variable prédite Y fait décliner l'apprentissage supervisé en deux sous catégories : La classification et la régression. Quand la variable à prédire prend une valeur discrète, on parle d'un problème de classification. Parmi les algorithmes de classification, on retrouve : les Machine à vecteurs de support (SVM), les Réseaux de neurones, l'algorithme Naïve Bayes, Logistic Regression... etc. La régression est le traitement de données dont la prédiction amène un résultat imprécis ou variable continue. Les algorithmes de régression peuvent prendre plusieurs formes en fonction du modèle qu'on souhaite construire. La régression linéaire est le modèle le plus simple qui consiste à trouver la meilleure droite qui s'approche le plus des données d'apprentissage.

2.8.2.2 Apprentissage Non-Supervisé

L'apprentissage non supervisé, encore appelé apprentissage à partir d'observations ou découverte, consiste à déterminer une classification « sensée » à partir d'un ensemble d'objets ou de situations données (des exemples non étiquetés). On dispose d'une quantité de données indiérenciées, et l'on désire savoir si elles possèdent une quelconque structure de groupes. Il s'agit d'identifier une éventuelle tendance des données à être regroupées en classes.

La plus grande application du non-supervisé est le " Clustering " ou regroupement qui consiste, à partir d'un tas de données, à déterminer des similitudes et se charger de créer des "clusters" ou groupes aux différentes caractéristiques ; c'est par exemple le cas du regroupement d'articles sur un site de référencement afin de définir ceux qui sont liés de manière directe (par genre ou thématique) ou indirecte (par type d'information traitée), et pas simplement par la catégorie de l'article.

2.8.2.3 Apprentissage par Renforcement

C'est un modèle où l'on considère les concepts de temps, état et récompense. A chaque instant T le modèle reçoit un nouvel état E en entrée, et lui associe une opération en sortie Y, puis se voit attribuer une récompense R en conséquence. Cette technique est particulièrement utilisée dans les jeux tels que les échecs, les dames ou encore pour la création d'IA dans les jeux vidéo.

2.9 Algorithmes de classification

2.9.1 Classification Naïve Bayésienne

La classification naïve bayésienne est un modèle probabiliste simple, basé sur le théorème de Bayes, avec une forte hypothèse d'indépendance des paramètres. La règle de décision bayésienne consiste à associer à chaque nouvel individu à classer, la classe la plus probable. On résume son utilisation comme suit :
- Lors de la phase d'entraînement, le classificateur calcule les probabilités qu'un nouveau document appartienne à telle catégorie à partir de la proportion des documents d'entraînement appartenant à cette catégorie. Pour chaque classe c_k, estimer :

$$\mathrm{P}(\mathrm{c}_k) = \frac{n_k}{n}$$

Le classificateur calcule aussi la probabilité qu'un mot donné soit présent dans une catégorie, pour tout mot $\mathrm{m}_j, (\mathrm{P}(\mathrm{m}_j|c_k))$.
où :
n_k : Nombre de textes de la classe c_k.
n : Nombre total de textes.
- Par la suite, quand un nouveau document D doit être classé, on calcule les probabilités qu'il appartienne à chacune des catégories à l'aide de la règle de Bayes et on lui attribue la classe c_k maximisant $P(c_k|D)$ avec :

$$\mathrm{P}(\mathrm{c}_k|D) = \prod_{j \in D} P(m_j|c_k) \cdot P(c_k)$$

Avantages :

- A besoin de moins d'apprentissage que les autres techniques que nous aborderons plus bas.
- Très simple, et facile à implémenter.
- Peut être utilisé pour la classification binaire ou multi critères.
- Non sensible aux entrées qui n'aident pas à l'apprentissage.

Désavantages :

- Temps de calcul conséquent.
- Difficulté à appliquer avec des variables continues comme le temps.
- Ne s'améliorera pas avec une base de données trop grande.

2.9.2 Classification par Arbre de Décision

L'apprentissage d'arbre de décision, dans le domaine de la classification de textes, est une technique d'apprentissage supervisé se basant sur la génération automatique d'un arbre de décision. Dans ces structures d'arbre, les feuilles représentent les valeurs de la variable-cible et les embranchements correspondent à des combinaisons de variables d'entrée qui mènent à ces valeurs. Autrement dit, chaque nœud de branche représente un choix entre un certain nombre d'alternatives et chaque nœud de feuille représente une décision.

En apprentissage et en fouille de données, un arbre de décision décrit les données mais pas les décisions elles-mêmes, l'arbre serait utilisé comme point de départ au processus de décision.

Une des variables d'entrée est sélectionnée à chaque nœud intérieur (ou interne, nœud qui n'est pas terminal) de l'arbre selon une méthode qui dépend de l'algorithme et qui sera discutée plus loin. Chaque arête vers un nœud-fils correspond à un ensemble de valeurs d'une variable d'entrée, de manière à ce que l'ensemble des arêtes vers les nœuds-fils couvrent toutes les valeurs possibles de la variable d'entrée.

Dans la figure 2.8, on observe un exemple d'arbre de décision pour le diagnostic. Chaque nœud correspond à une condition et chaque arrête à un scénario (ou réponse) possible à cette condition. Enfin, les feuilles de l'arborescence générée représentent la classe définie. Dans cet exemple, on distingue 5 classes distinctes qui sont : Rhume, Appendicite, Mal de gorge, Refroidissement, et Rien qui signifie que le patient ne souffre d'aucun mal.

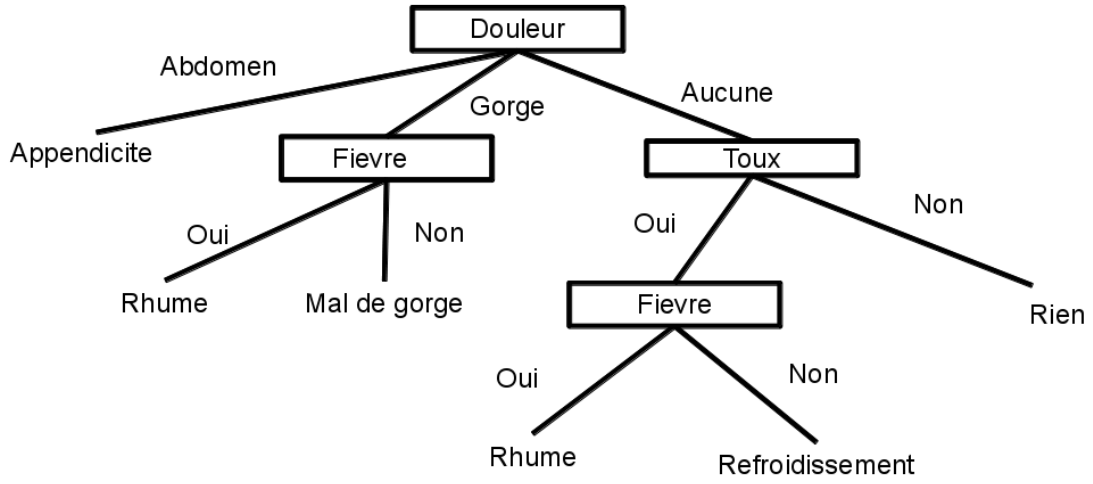


FIGURE 2.8 – Arbre de Décision (source : up2.fr)

Avantages :

— Facilité de compréhension et de génération des règles à partir d'un arbre de décision.

Désavantages :

— Problème de surapprentissage (overfitting).
— Gère difficilement les valeurs non-numériques.
— Peut générer un arbre trop large.

2.9.3 K-Plus Proches Voisins KNN

La méthode des k plus proches voisins, ou KNN (de l'anglais k-nearest neighbors), est une méthode d'apprentissage supervisé qui consiste à prédire la classe d'une entrée X en fonction de ses k plus proches voisins déjà étiquetés en mémoire.

Dans cet algorithme le "k" représente le nombre d'éléments voisins que l'on prendra en compte. KNN est une méthode très connue dans le domaine de la catégorisation automatique.

Dans la figure 2.9, on observe la nouvelle entrée « *?* » qui n'a pas encore de classe. Dépendamment du nombre de k défini, le rayon du cercle grandira afin de prendre en compte k entrées (triangles) étiquetées (triangle plein ou non) autour d'elle.

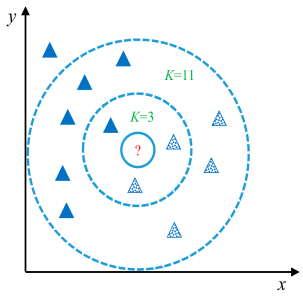


FIGURE 2.9 – KNN (source : mdpi.com)

Avantages :

— La méthode ne nécessite pas de phase d'apprentissage ; c'est l'échantillon d'apprentissage.
— Les approches basées sur K-NN sont simples et doivent leurs performances à l'identification dans le jeu d'apprentissage d'un vocabulaire discriminant.
— Très efficace dans les problèmes de reconnaissances.

Désavantages :

— Temps de calcul conséquent dû au fait que c'est un "lazy learner" : il ne génère pas un modèle d'apprentissage à l'avance ; il utilise simplement les données à sa disposition au moment où la classification est requise.

2.9.4 Machine à vecteurs de support SVM

Machine à vecteurs de support ou SVM (de l'anglais Support Vector Machine) est un algorithme d'apprentissage automatique supervisé qui peut être utilisé pour les défis de classification ou de régression. Cependant, il est principalement utilisé dans les problèmes de classification. Dans cette méthode l'algorithme cherche l'hyperplan qui séparerait les deux classes, en garantissant que la marge entre le plus proche des positifs et des négatifs soit maximale, permettant ainsi une généralisation aux nouveaux exemples à venir, qui eux pourront différer de l'existant. Le but est de sélectionner des vecteurs supports qui représentent les vecteurs de séparation qui définissent l'hyperplan.

Avantages :

- Les SVM possèdent des fondements mathématiques solides.
- Les exemples de test sont comparés seulement avec les supports vecteurs et non pas avec tous les exemples d'apprentissage.
- La classification d'un nouvel exemple consiste à voir le signe de la fonction de décision f(x) ce qui rend la décision rapide.

Désavantages :

- C'est une classification binaire d'où la nécessité d'utiliser l'approche un-contre-un.
- La grande quantité d'exemples en entrées implique un calcul matriciel important.
- Le temps de calcul est élevé lors d'une régularisation des paramètres de la fonction noyau.

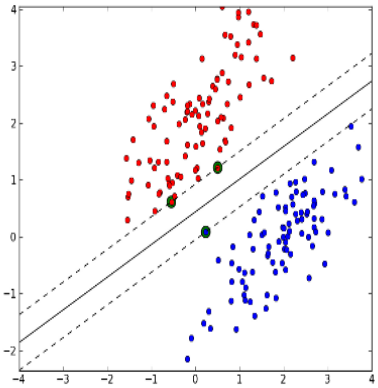


FIGURE 2.10 – SVM (source : quora.com)

2.9.5 K-Moyennes

Le regroupement des K-moyennes (K-means) est un type d'apprentissage non supervisé qui est utilisé lorsqu'il y a des données non étiquetées (c'est-à-dire, des données sans catégories ou groupes définis). Le but de cet algorithme est de trouver des groupes dans les données, avec le nombre de groupes représenté par le paramètre K. L'algorithme fonctionne itérativement pour affecter chaque point de données à l'un des K groupes en fonction des caractéristiques fournies. Les points de données sont regroupés en fonction de la similarité des entités. Les résultats de l'algorithme de clustering K-moyennes sont :

— Les centroïdes des groupes K, qui peuvent être utilisés pour étiqueter de nouvelles données.
— Les étiquettes pour les données d'apprentissage (chaque point de données est affecté à un seul cluster).

Dans la figure ci-dessous on observe une catégorisation à 3 classes, séparées par 3 vecteurs.

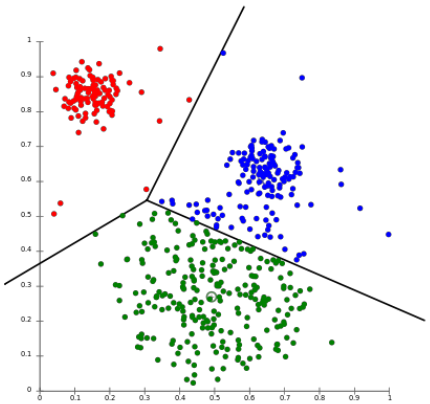


FIGURE 2.11 – K-moyennes

Avantages :
— Si les variables sont grandes k-means est l'algorithme le plus rapide en temps de calcul.
— K-means produit des groupes plus serrés que d'autre types de classification.

Désavantages :
— Difficulté de prédire la valeur de K.
— Fonctionne mal avec des groupes globaux.
— L'arrangement de base des partitions influe trop sur celui final (variété de dispositions des groupes finaux).

— A ne pas utiliser avec des groupes aux tailles et densités différentes.

2.10 Recherche par escalade (Hill Climbing)

Hill Climbing ou Recherche par Escalade est une recherche heuristique utilisée pour les problèmes d'optimisation mathématique dans le domaine de l'Intelligence Artificielle. Étant donné un grand nombre d'entrées et une bonne fonction heuristique, il essaie de trouver une solution suffisamment bonne au problème. La «recherche heuristique» signifie que cet algorithme de recherche peut ne pas trouver la solution optimale au problème, le maximum optimal global. Cependant, cela donnera une bonne solution dans un délai raisonnable. Cette méthode de recherche en profondeur utilise une fonction heuristique pour choisir à chaque étape le nœud à générer. Elle garde un seul état («l'état courant») et l'améliore itérativement jusqu'à converger à une solution. Une fonction objective doit être optimisée (maximisée ou minimisée).

2.11 Indicateurs de performance en classification

2.11.1 Matrice de confusion

Pour mesurer les performances d'un classifieur. Pour la représenter il faut distinguer 4 types d'éléments que nous expliquons dans le cas d'un classifieur binaire :
1. Vrai positif (VP) : Elément de la classe 1 correctement prédit.
2. Vrai négatif (VN) : Elément de la classe 0 correctement prédit.
3. Faux positif (FP) : Elément de la classe 1 mal prédit.
4. Faux négatif (FN) : Elément de la classe 0 mal prédit.

Ces informations peuvent être rassemblées et visualisées sous forme de tableau dans une matrice de confusion. Dans le cas d'un classifieur binaire, on obtient :

		Classe prédite	
		Classe 0	Classe 1
Classe réelle	Classe 0	VN	FN
	Classe 1	FP	VP

FIGURE 2.12 – Matrice de Confusion

En particulier, si la matrice de confusion est diagonale, le classifieur est parfait.

2.11.2 Indicateurs de performance de base

Il est possible de calculer plusieurs indicateurs résumant la matrice de confusion. Par exemple si nous souhaitons rendre compte de la qualité de la prédiction sur la classe 1, on définit :
• Précision. Proportion d'éléments bien classés pour une classe donnée :

$$Precision_{delaclasse} = \frac{VP}{VP + FP}$$

• Rappel. Proportion d'éléments bien classés par rapport au nombre d'éléments de la classe à prédire :

$$Rappel_{delaclasse} = \frac{VP}{VP + FN}$$

• F-mesure. Mesure de compromis entre précision et rappel :

$$F - mesure_{delaclasse} = \frac{2 \times (PrecisionRappel)}{Precision + Rappel}$$

$$Precision = \frac{1}{k} \sum_{i=1}^{k} \frac{VP_i}{VP_i + FP_i}$$

$$Rappel = \frac{1}{k} \sum_{i=1}^{k} \frac{VP_i}{VP_i + FN_i}$$

Il est possible de calculer tous ces indicateurs pour chaque classe. La moyenne sur chaque classe de ces indicateurs donne des indicateurs globaux sur la qualité du classifieur.

2.12 Conclusion

Ce premier chapitre nous a permis de constater la nécessité d'un système de gestion de fichiers et les avantages que promet d'offrir notre application afin de combler les lacunes des solutions déjà disponibles sur le marché. Par la suite, nous avons introduit les concepts les plus importants du Traitement Automatique du Langage Naturel et de l'Apprentissage Automatique, ainsi que leurs diérentes techniques qui nous serons utiles dans notre application. Dans le prochain chapitre nous présenterons la conception de notre solution.

Chapitre 2 : Conception

3.1 Introduction

A l'ère du Big Data le nombre de documents stockés sur les supports électroniques explose, l'utilisation d'outils facilitant leur organisation est devenu indispensable, pour aider les utilisateurs de ces masses de données à les retrouver efficacement. C'est dans cette optique que notre Système Intelligent de Gestion de fichiers a vu le jour ; il est la réponse à tous les problèmes inhérents au système de classification par dossiers.
Nous proposons au travers de ce système une gestion totalement automatisée des documents et une organisation hiérarchisée sémantiquement, sur la base des contenus des différents fichiers, et ce selon des catégories pensées et établies par des professionn -els en linguistique.
L'utilisateur pourra visualiser tous ses fichiers dans une certaine organisation au travers de l'application sans que ces derniers n'aient été déplacés physiquement.

3.2 Conception Générale du Système de Gestion de fichiers

Nous proposons, au vu des lacunes des systèmes de gestion de fichiers existants d'offrir une nouvelle approche d'organisation du contenu de son ordinateur. Cette approche se baserait essentiellement sur la catégorisation automatique des fichiers textes et se chargerait d'organiser les fichiers multimédias de façon semi-automatique.

L'application gère une large gamme de fichiers et traite les documents dans les trois langues l'Arabe, l'Anglais et le Français, visant ainsi un large public.
L'utilisateur a accès à un panel exhaustif de catégories pour ses fichiers : qu'importe le thème abordé dans son fichier, il s'accordera nécessairement à une parmi 1 million de catégories représentées dans l'arborescence principale du système. Cette arboresce -nce contient la totalité des catégories, ce qui représente précisément la totalité de la hiérarchie d'ODP (voir le chapitre précèdent). Cependant, l'utilisateur n'interagira qu'avec une version simplifiée de cette dernière, une arborescence logique qui n'inclut que les catégories qui s'accordent à ses besoins et à ses préférences, c'est-à-dire les catégories qui contiennent au minimum un document. Cette arborescence est mise à jour à chaque fois que l'application reçoit un signal l'avertissant de l'arrivée ou de la modification d'un fichier (signal émis par le système d'exploitation Linux).

La figure 3.1 est une représentation non exhaustive de l'arborescence principale du système.

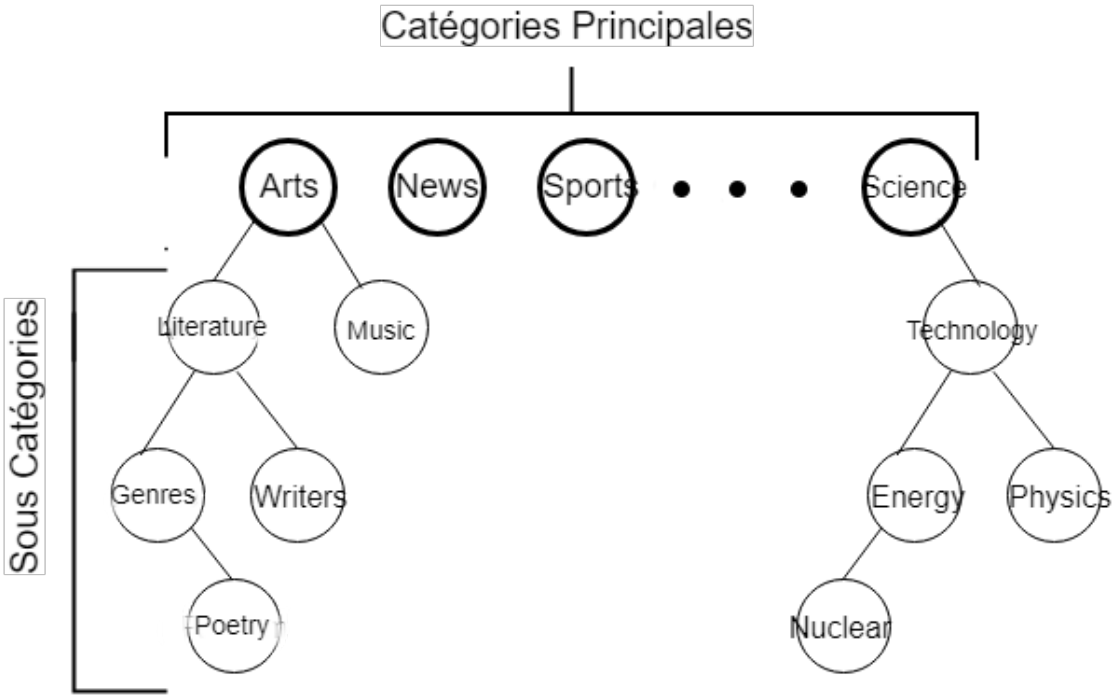


FIGURE 3.1 – Arborescence Principale du Système

Chaque fichier de l'utilisateur après prétraitement se verra attribuer une catégorie au moyen d'un classifieur et sera ajouté à la hiérarchie. De plus, l'utilisateur aura la possibilité d'étiqueter lui-même ses fichiers qu'ils soient textuels ou multimédias afin d'avoir une catégorisation personnalisée et donc de faciliter sa recherche.
Pour trouver un fichier, il suffit tout simplement d'entrer un ou plusieurs mots-clés définissant la recherche. Durant la frappe, une proposition de tags ou catégories adaptée à la recherche est faite. La recherche se fait en 3 phases : recherche par tags, recherche par catégories et enfin recherche par contenue.

3.3 Préparation du Corpus

3.3.1 Sélection du corpus

À la lumière de ce qui a été introduit dans le premier chapitre sur la catégorisation automatique de textes, nous avons pris conscience que plusieurs éléments sont impliqués dans le processus, que ce soit le mode de représentation des documents ou l'algorithme d'apprentissage mis en place. Indépendamment de ces facteurs, un point reste inchangé : un corpus d'entraînement est primordial à un apprentissage automatique ; la performance de l'apprentissage dépendra de cet impératif. Et une des difficultés rencontrées lors de la création d'un classificateur automatique de textes est

la collecte d'un grand nombre de documents préalablement étiquetés pour constituer ce corpus d'entraînement. Il est important de noter que plus le jeu d'apprentissage est important, plus le classificateur a l'opportunité de bien performer lors de la classification de nouveaux documents, c'est-à-dire de produire une classification qui réduit au maximum le risque d'erreurs.

Une première approche s'est présentée à nous : constituer notre propre corpus d'apprentissage à partir de l'ontologie WordNet. Cette méthode part de la relation qui existe entre les synsets de WordNet et les domaines organisés en hiérarchie du WordNet Domain.

Algorithm 1 Collecte de Corpus

```
for D ∈ WordNetDomain do
    f = OuvrirFichier(D.txt, encriture)
    ListeSynsets = D.ListeSynsets()
    for Syns ∈ ListeSynsets do
        Gloss = Gloss(Syns) + Gloss(Descendants(Syns))
        f.crire(Gloss)
    end for
    f.FermerFichier()
end for
```

Cependant, cette méthode a deux principaux inconvénients. Le premier est l'important déséquilibre de contenu entre certaines catégories et le second est le nombre réduit de domaines et par conséquent un échantillon de catégories non exhaustif.

Au vu de ses lacunes, notre choix s'est tourné vers une autre approche se basant sur la hiérarchie de DMOZ.

Comme présenté précédemment, DMOZ est un annuaire de Sites Web. Son organisation est celle d'un arbre, partant de 15 nœuds têtes (qui représenteront nos catégories principales) et se développe jusqu'à atteindre 12 niveaux de profondeur où chaque nœud fils (sous-catégorie) est une précision du nœud père. A chaque nœud est associé un ou plusieurs sites web dont le contenu, dans notre cas, représentera notre jeu de donnée d'apprentissage étiqueté. En effet, il suffit de collecter le contenu de chaque site web associé à un nœud (une catégorie) et l'associer à la catégorie dans laquelle il est indexé pour constituer notre corpus d'apprentissage nécessaire à l'apprentissage du modèle de prédiction automatique.

3.3.2 Web Scraping

De prime abord il est important de définir ce qu'est que le Web Scraping. Le Web Scraping est l'ensemble de techniques visant à extraire le contenu d'un Site Web.

Dans le cadre de nos travaux, il a fallu procéder en 2 étapes :

— Étape 1 : Parcours de dmoztools.net :
Il s'agit d'abord de parcourir l'entièreté du million de catégories que contient le site. Il faut pour cela effectuer un parcours des balises d'hyperliens de la page afin de récolter le contenu des DIVs associés à la classe "cat-item" qui représentent les liens vers les sous-catégories. A chaque niveau nous récoltons les DIVs de la classe "site-title" contenant les sites externes traitant de la thématique de cette catégorie, qui seront associés chaque fois à la catégorie actuelle, et représenteront son contenu (une fois extraits).
Parallèlement à cela, nous récoltons pour chaque catégorie les balises qui sont contenues dans les DIVs dont les classes sont "site-title".

— Étape 2 : Parcours des sites externes :
Dans un second temps, on parcourt chacun des sites répertoriés dans la première étape, et on en extrait le contenu des articles, ce qui représentera à l'avenir notre jeu d'apprentissage.

3.3.3 Traduction

Afin de maintenir une même structure d'organisation pour les trois langues traitées par le système, il a été décidé de :

1. Maintenir la structure d'arbre propre à la langue anglaise de ODP (DMOZ), car étant la hiérarchie la plus riche et la plus englobante. A titre comparatif, l'arborescence de la langue arabe comprend 617 catégories et sous-categories confondues, l'arborescence en langue française quant à elle est plus riche et atteint les 65061 catégories au total cependant les deux réunis ne représentent que 20% du nombre de catégorie concernant la langue anglaise).

2. Recueillir tout le jeu de données extrait des articles des sites indexés par ODP, toujours d'après l'arborescence Anglaise.

3. Dans le but d'unifier la hiérarchie pour les trois langues, à savoir l'Arabe, l'Anglais et le Français, et d'éviter des corpus pas assez riches et variés en langue Française et Arabe, nous avons préféré garder le même jeu d'apprentissage en Anglais et de le traduire au moyen de L'API de Google Traduction afin de concevoir nos propres corpus dans les 2 autres langues.

Autrement dit, à partir d'un ensemble pré-étiqueté de textes en langue Anglaise (résultant de l'extraction du contenue de sites indexés par ODP) faire une traduction automatique de tous ces textes vers les deux autres langues (Française et Arabe) et appliquer l'apprentissage sur l'ensemble de textes correspondant à la langue du document à catégoriser.
Nous avons jugés que cette approche est très raisonnable car notre but est la catégorisation plutôt que la traduction elle-même.

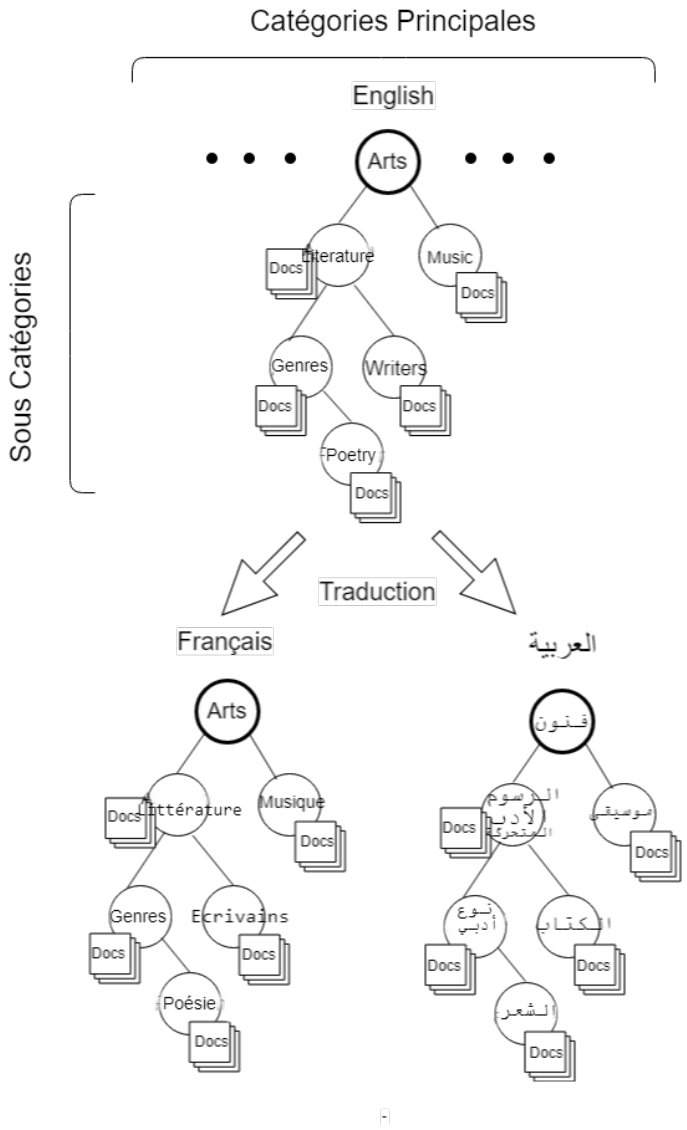


FIGURE 3.2 – Traduction du Corpus d'apprentissage

3.4 Détection de la langue

Notre Système de Gestion de Documents, propose la gestion de documents en 3 langues qui sont l'Arabe, l'Anglais et le Français. Pour cela il est donc nécessaire de détecter la langue du document afin de passer à sa catégorisation. Il y a plusieurs façons de détecter la langue d'un texte, même si la plus simple et efficace consiste en une approche basée sur les mots vides ou les StopWords, elle peut être décrite comme suit :
Le texte est initialement réduit en tokens (représentation du texte brut en liste de mots), puis on compare la liste des mots obtenus à la liste des mots contenus dans les différentes listes de StopWords des différentes langues, sachant que chaque langue a une liste de mots vides qui lui est propore.
Ainsi la langue dominante sera celle ayant la plus importante intersection de sa StopList et les mots du document.

Algorithm 2 DetectionLangue(Texte)

$Ratio\_Langue = []$
$Words = Tokeniser(Texte)$
for `Langue in StopWords.list()` **do**
`éléments_communs=Words` $\cap StopWords(Langue)$
`Ratio_Langue[Langue]=taille(éléments_communs)`
end for
return Max(Ratio-Langue)

3.5 Analyse et traitement des différents types de fichiers

3.5.1 Traitement des fichiers texte

Notre système a pour rôle d'organiser les documents de l'utilisateur de façon intelligente. Il s'intéresse donc à l'application de méthodes issues de l'apprentissage automatique à la catégorisation de textes multilingues.
La catégorisation se fera sur les types de documents suivants : Powerpoint, Excel, Word, PDF, ODT, ePUB, CSV.
Il est néanmoins important de noter que pour obtenir les meilleurs résultats possibles lors de l'application des méthodes d'apprentissage automatique au traitement des données non structurées, il nous faudra, d'abord, prétraiter ces documents et les convertirons en texte brut.

Les fichiers étant au niveau du système d'exploitation, chacun a un encodage différent correspondant à son extension. Par conséquent, l'application doit pouvoir, selon le fichier, décoder son contenu et le lire pour qu'il puisse être traité. Pour cela, il a été nécessaire de mettre au point un programme qui se chargera de trier les différents documents afin de les convertir en texte brut nécessaire aux traitements à venir.

3.5.2 Traitement des dossiers compressés

Lorsque le système rencontre un dossier compressé, il procède comme suit :

— Récupérer le contenue du fichier compressé sans le décompresser.
— Catégoriser chacun des documents compris dans le dossier.
— Attribuer au dossier compressé la catégorie dominante. Si par exemple un dossier compressé A contient trois documents d1, d2 et d3 catégorisés respectivement : C1, C2 et C2 et bien le dossier sera classé dans C2.

3.5.3 Traitement des fichiers multimédia

Les fichiers multimédias (de types JPEG, PNG, MKV, AVI, etc.), quant à eux, ne subiront aucun traitements et ne se verront pas catégorisés par une méthode d'apprentissage, mais seront néanmoins représentés dans l'application. L'utilisateur

pourra accéder à ses photos et vidéos via l'application au travers d'une arborescence indépendante de celle des documents textes. Il profitera d'une classification semi-automatique et totalement personnalisable de ses fichiers multimédias et ce au moyens de tags (étiquettes). En effet le système attribuera par défaut certains tags (catégories) aux fichiers en fonction de leurs caractéristiques (le type de document, la date de création, etc.) et proposera à l'utilisateur d'étiqueter ces documents, de leur associer ses propres tags afin de faciliter leur recherche.
Autrement dit, la catégorisation des fichiers multimédia se fera de façon semi-automatique.

3.5.4 Traitement des codes et fichiers de programmation

Le système cible un large public incluant notamment informaticiens et programmeurs en tous genres. Ainsi, un module a été pensé pour faciliter la recherche de leurs différents projets. Tous les dossiers incluant au minimum un fichier de programmation seront ajoutés à l'arborescence, où ils seront regroupés par langage de programmation. Il est important de noter que c'est le dossier en entier qui sera indexé et non le code, ce qui permettra de garder une cohésion et d'éviter d'éclater les dossiers et donc séparer les fichiers complémentaires.

3.5.5 Traitement des fichiers de configuration

Il reste néanmoins certains types de fichiers non pris en compte par notre système de gestion ; ce sont les fichiers de configuration. Toutefois sous Linux tous les fichiers de configurations n'ont pas forcément une extension ".ini " ; certains n'en ont tout simplement pas. Il a été donc nécessaire de programmer un Trieur qui se chargerait de différencier les fichiers de configurations de ceux qui n'en sont pas. Ce Trieur est un classifieur binaire qui, en fonction du contenu du fichier, prédit si c'est un fichier de configuration (1) ou un fichier traitable par notre système. (0)

3.5.5.1 Collecte du corpus d'apprentissage du Trieur

Afin que notre "Trieur" apprenne à prédire si un fichier est un fichier de configuration ou non, comme pour tout modèle d'apprentissage, il a fallu lui constituer un corpus d'apprentissage. Pour ce faire nous avons extrait tous les fichiers de configuration, présents sur différentes machines sous diverses distributions Linux, généralement stockés dans les répertoires "/etc/" (le répertoire contenant les fichiers de configurations des modules de base du système d'exploitation) et "/home/nom_utilisateur/.* qui est l'expression régulière représentant l'ensemble des répertoires ou fichiers cachés stockées dans le répertoire 'home' de l'utilisateur, car c'est à cet endroit que se stockent les fichiers de configurations des applications de ce même utilisateur.

3.5.5.2 Apprentissage du Trieur

La méthode apprentissage choisie pour la classification de fichiers en fichiers de configuration ou pas est SVM (Machine à Vecteurs de Support) car elle est très

populaire pour la classification binaire en particulier. Leur performance provient du fait qu'elles reposent sur l'application d'algorithmes de recherche de règles de décision linéaires ("hyperplan séparateur")[10]. Une extraction de caractéristiques a été appliquée à notre corpus d'apprentissage.
Les caractéristiques extraites sont :

— Taille de texte.
— Nombre de mots contenus dans le texte.
— Nombre d'occurrences du symbole "#".
— Nombre d'occurrences de "enable" et "disabled".
— Nombre d'occurrences de "1" ou "0".
— Nombre d'occurrences de "True" et "False".
— Nombre d'occurrences des ponctuations.

Ces caractéristiques permettront au modèle de différencier les fichiers et par conséquent donner une bonne prédiction.

3.6 Prétraitement et Représentation des Documents

En vue de décider où organiser un fichier quelconque (de quelque type qu'il soit, comme expliqué précédemment), il est important de représenter ces documents de façon appropriée.

Cette phase est importante et comporte deux choix qui affectent souvent les performances : le choix de termes (mot, lemme, stemme ou n-grammes) et le choix des poids associés à ces termes (absence/présence, nombre d'occurrences, fréquence, etc.).

Le prétraitement des documents comprend les étapes suivantes :

— Étape 1 : Tokenization : L'ensemble des traitements effectués pour transformer un texte brut en une liste de de mots.

— Étape 2 : Lemmatisation : Remplacer les mots dans la liste par leur forme canonique.

— Étape 3 : Suppression des StopWord : Supprimer de la liste tous les mots de la StopList, i.e. tous les mots vides (les mots non significatifs).

— Étape 4 : Vectorisation : Représenter un texte sous forme de vecteur en attribuant à chaque mot un poids suivant la méthode de pondération TF-IDF.

La figure 3.3 représente les étapes par lesquelles passe le document.

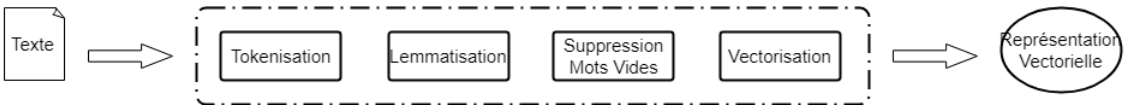


FIGURE 3.3 – Prétraitement et Représentation des Documents

3.7 Classifieur pour la catégorisation d'un document

Le classifieur ou encore le cœur de notre application aura pour rôle de prédire la catégorie de chaque nouveau document. Il opérera en 2 principales étapes : la première sera la prédiction de la Catégorie Principale en utilisant un modèle d'Apprentissage Automatique, tandis que la seconde étape se basera sur une recherche heuristique de type Hill Climbing (ou Recherche Par Escalade) pour affiner la catégorisation.

3.7.1 Apprentissage Automatique pour la prédiction de la catégorie principale

Le choix de la méthode d'apprentissage est primordial ; de nombreuses méthodes ont été présentées précédemment, chacune possédant des avantages, et des inconvénients. Après un comparatif de tous les modèles, nous avons constaté de meilleurs performances avec Multinomial Naive Bayes. Ces résultats comparatifs seront présentés dans le chapitre suivant

Cette variation du Classifieur Naïve Bayes, telle que décrite par[11], estime la probabilité conditionnelle d'un terme particulier donné à une classe comme la fréquence du terme t dans les documents appartenant à la classe c.

$$P(t/c) = \frac{T_{ct}}{\sum_{t' \in v} T_{ct'}}$$

Ainsi, cette variation prend en compte le nombre d'occurrences du terme t dans les documents du jeu d'apprentissage de la classe c, y compris les occurrences multiples.

3.7.2 Recherche par escalade (Hill Climbing) pour la recherche de la sous-catégorie

Après avoir attribué au document à catégoriser sa Catégorie Principale au moyen du modèle d'apprentissage automatique comme expliqué dans la section précédente, le document passe par une seconde étape afin d'affiner sa catégorisation à l'aide de la recherche heuristique par Escalade.
Nous pouvons résumer le comportement du Hill Climbing par le pseudo code suivant :

Algorithm 3 Hillclimbing(catégorie, document)

```
matrice=chargerCentroid(catégorie)
ti=tfidf(document)
min=distance-cosinus(tf-idf, matrice)
indice=-1
for i ∈ noeuds_fils(catgorie) do
  matriceFils=chargerCentroid(i)
  temp=distance-cosinus(tf-idf, matriceFils)
  if temp<=min then
    min=temps
    indice=i
  end if
end for
if indice!=-1 then
  return Hillclimbing(indice, document)
else if
return then
  catégorie
end if
```

3.8 Fonctionnalitées

3.8.1 Outil de Recherche

Le dessein principal de notre application est d'organiser les fichiers que l'utilisateur a accumulés sur son ordinateur, de façon à permettre une recherche intuitive et efficace. Grâce à une auto complétion intelligente, l'outil de recherche permet de retrouver les documents avec un ou plusieurs mots clés.
Cette recherche se fait en 3 étapes :

- Recherche par Tags :
  L'utilisateur a la possibilité d'étiqueter ses documents de la manière qui lui convient en leur attribuant des tags. Chaque fichier sera susceptible d'avoir un ou plusieurs tags. A la recherche, les résultats considérés comme prioritaires seront ceux dont les tags ont une concordance avec les mots clés tapés.

- Recherche par Catégorie :
  Les documents retournés et considérés pertinents juste après la recherche par tags sont ceux catégorisés sous la catégorie entrée dans la recherche.

- Recherche par Contenu :
  Enfin apparaitront les documents contenant les mots entrés, triés selon la distance qui sépare la chaine entrée et les documents trouvés.

3.8.2 Catégories Prédéfinies

Les fichiers sont automatiquement triés en fonction des habitudes de recherche de l'utilisateur. Ainsi sous l'onglet Pertinent, l'utilisateur aura accès à ses fichiers les plus souvent consultés en prenant en compte la récence suivant les formules données : La formule appliquée au document ouvert :

$$P = (1 - \alpha) \times P + \alpha$$

La formule appliquée à tous les autres documents du système :

$$P = (1 - \alpha) \times P$$

A l'arrivée d'un nouveau document, il est initialisé selon la formule :

$$P = \frac{1}{Total_{categories}}$$

Avec $\alpha = 0.1$.

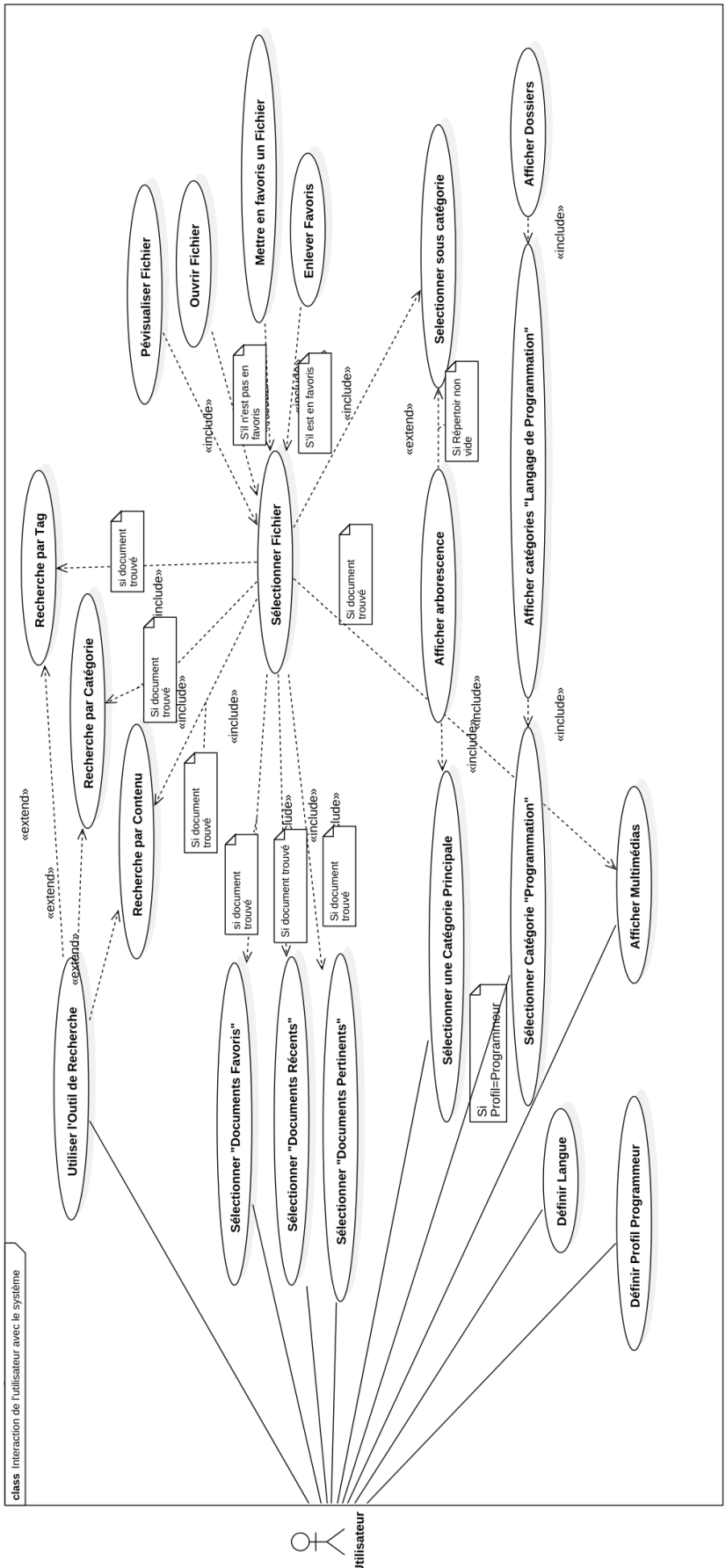
Sous l'onglet Récent, il trouvera ses fichiers récemment ouverts qui seront empilé dans la pile « Récents » selon la méthode LIFO (Dernier arrivé, premier sorti).
Enfin Favoris comprendra tous les fichiers que l'utilisateur a marqué comme favoris.

3.9 Diagrammes

3.9.1 UML

UML, abréviation de « Unified Modeling Language », est un langage de modélisation standardisé constitué d'un ensemble de diagrammes, il permet de visualiser, spécifier, construire et documenter les abstractions d'un système logiciel.

3.9.1.1 Diagramme de Cas d'Utilisation



Fiche de cas d'utilisation

Nom : Recherche Document.
Acteur : L'utilisateur.
Description : La recherche de documents se fait à l'aide de l'outil de recherche en entrant une séquence de mots.
Pré-conditions : Aucune. L'utilisateur peut entrer n'importe quel mot, dans n'importe quelle langue.

DESCRIPTION
Le scénario nominal :
Le système affiche une page contenant la liste :

1. Des fichiers dont les tags correspondent aux mots entrés ou leurs racines.
2. Des documents classés dans la catégorie correspondante aux mots entrés ou leurs racines.
3. Des documents qui contiennent les mots entrés ou leurs racines, classés selon le nombre d'apparitions de ces derniers.

3.9.1.2 Diagramme de Classe

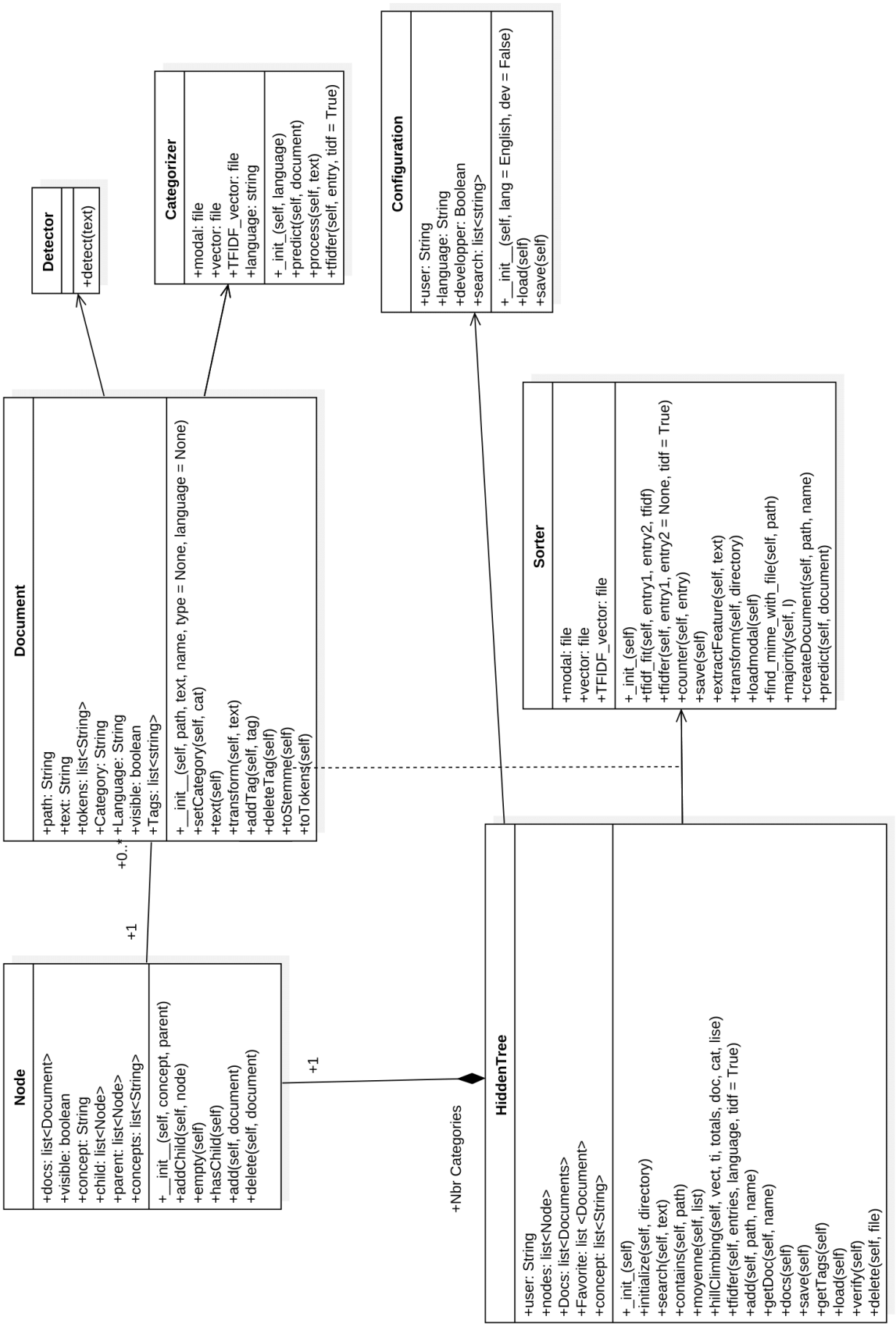


FIGURE 3.5 – Diagramme de Classe

3.9.1.3 Diagramme de Séquence

Recherche d'un Fichier

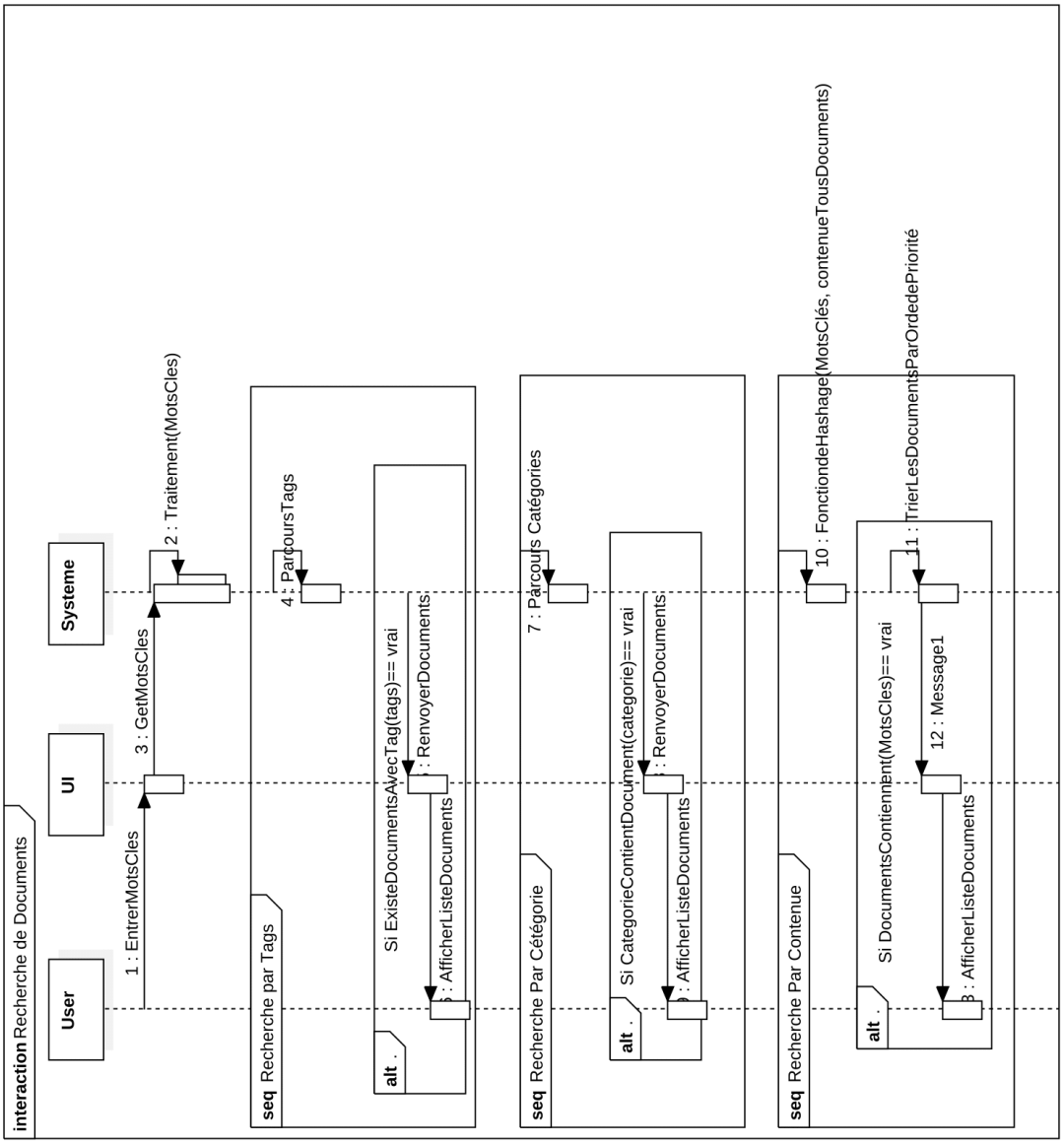


FIGURE 3.6 – Diagramme de Séquence de Recherche

3.10 Conclusion

En achevant notre premier chapitre, le but que nous nous étions fixé était de placer notre approche dans son contexte au sens large. Le second chapitre quant à lui s'est centré sur la conception de l'application est a pour principal dessein la description logique de la façon dont le système va fonctionner. Elle consiste à façonner le Système et lui donner une forme et une architecture. Elle constitue une entrée majeure pour les deux dernières à savoir l'implémentation et le test.

Chapitre 3 : Réalisation

4.1 Introduction

Dans ce chapitre, nous présentons l'architecture sur laquelle nous avons développé notre application, les différents outils utilisés ainsi que les composantes applicatives réalisées.

4.2 Initialisation et rafraichissement de la base des documents

Lorsque l'utilisateur utilise pour la toute première fois le système de gestion de fichiers, ce dernier lance un processus de recherche des fichiers contenus dans son ordinateur et parcourt la totalité des répertoires pouvant contenir des fichiers qu'il peut traiter, à savoir tous les répertoires excepté ceux ayant le chemin « /.* » ou « etc » car ceux-là sous Linux sont réservés aux fichiers de configurations, fichiers non catégorisables par notre système.

A chaque ajout, suppression ou modification d'un fichier ou d'un dossier, le système d'exploitation (dans notre cas Linux) envoie un signal système à l'application la prévenant d'un changement, autrement dit il existe un objet QFileSystemWatcher qui, à partir du moment où il est instancié, inspecte le système et envoie un signal qu'on peut récupérer. A partir de ce signal, l'application se charge de retrouver les informations nécessaires au rafraichissement de la base des documents et dossiers dans l'organisation. Un thread [1] est lancé pour permettre à l'utilisateur d'interagir avec l'application sans subir la moindre pénalisation, pendant qu'en parallèle le système se charge du rafraîchissement de l'organisation totale.

1. Thread : processus léger qui se lance en parallèle au processus principal.

4.3 Ajout d'un nouveau Document

Chaque nouveau document passe par les étapes suivantes :

4.4 Trieur des fichiers de l'utilisateur

La première étape sera la représentation des documents sous forme de texte brut, de données non structurées manipulables par nos algorithmes. Or, pour ce faire, le système doit séparer les fichiers traitables de ceux de configuration qui ne sont pas pris en charge pour l'application. Comme expliqué dans le chapitre 2, le Trieur sera le module responsable de ce traitement.

4.4.1 Collecte de corpus de classification de fichiers Documents/Configuration

Le Trieur étant un classifieur binaire, un corpus d'entraînement est primordial pour son apprentissage. Or faire la collecte de documents « classiques » (c'est-à-dire de fichiers qui ne sont pas des fichiers de configuration) de nos jours est chose aisée, mais comme il n'existe pas de dataset pour les fichiers de configuration, il a fallu nous en constituer un. Notre méthode se base sur le fait que sous Linux les fichiers de configuration sont concentrés dans le répertoire « /home/user/.* » (l'ensemble des fichiers/dossiers commençant par un point dans le dossier home de l'utilisateur). Nous avons écrit un programme Python pour collecter l'ensemble de ces fichiers et répété l'opération sur plusieurs machines différentes afin de collecter un dataset conséquent (15300 instances) pour la classe Configuration. Pour les fichiers qui ne sont pas de configuration nous avons utilisé les datasets Reuters et News pour la classe Document.

4.4.2 Apprentissage du Trieur de fichiers

Concernant l'apprentissage du Trieur pour les deux classes : Configuration et Document Classique, nous avons implémenté le modèle SVM que propose la bibliothèque scikit-learn.
Les résultats obtenus sont amplement satisfaisants avec une précision de 99%.

4.5 Détecteur de Langue

Le détecteur de langue d'un document est un module permettant la détection de langue se basant sur une approche de stopwords comme expliqué dans le chapitre précèdent. En implémentant l'approche basée sur les stopwords nous avons obtenu un f-score de [0.97 0.86 0.96].

4.6 Prétraitement et Représentation des documents

Python offre toutes sortes d'outils nous permettant de mener à bien le prétraitement et la représentation. Nous en avons utilisé les outils suivants :

Tokenization

Nous avons utilisé le package NLTK, pour tokeniser les phrases en mots.

Lemmatisation

Au moyen de 3 algorithmes implémentés via NLTK, nous avons transformé les tokens en leurs formes canoniques ; « nltk.stem.isri » pour la langue Arabe, « nltk.stem.porter » pour la langue Anglaise et « nltk.stem.snowball » pour la langue Française .

StopWord

Il existe dans la librairie NLTK une liste par défaut des stopwords dans plusieurs langues, notamment l'Anglais, l'Arabe et le Français ; nous nous en sommes servis afin de nettoyer les mots vides de notre corpus.

Count Vectorizer

Le CountVectorizer fournit un moyen simple à la fois de tokeniser une collection de documents texte et de construire un vocabulaire de mots connus, mais aussi d'encoder de nouveaux documents en utilisant ce vocabulaire. Il est utilisé comme suit :

Algorithm 4 CountVectorizer

$C = CountVectorizer()$
Création d'une instance de Count Vectorizer.
$C.fit([Document1, Document2, ..., DocumentN])$
Utilisation de la fonction fit pour apprendre un vocabulaire.
$vecteurP = c.transform([DocumentP])$
Application du type de vecteur appris par C pour transformer le document arrivant P selon ce dernier, et optimiser ses parties creuses.

Une matrice codée est renvoyée avec associé à chaque mot du vocabulaire le nombre de fois où le mot est apparu dans le document. Il faut savoir que ces matrices sont généralement creuses. Python fournit un moyen efficace de gérer et optimiser ça via "scipy.sparse".

Tfidif Transformer

C'est une classe prédéfinie dans la bibliothèque Sklearn "sklearn.feature_extraction.text .TfidfTransformer" qui a pour rôle de transformer une matrice de comptage en une

représentation normalisée tf ou tf-idf.

4.7 Classifieur d'un document selon les catégories de ODP

4.7.1 Collecte Corpus

Afin de collecter le jeu d'apprentissage nécessaire au modèle d'apprentissage, nous avons extrait le contenu de ODP. Pour cela il a été nécessaire d'écrire un script qui se charge de faire le scraping du contenu des sites indexés au niveau de la hiérarchie de ODP et de leur associer les catégories dans lesquelles ils sont indexés afin de produire un corpus étiqueté. Pour ce faire nous avons eu recours à Newspaper qui est un module Python utilisé pour extraire et analyser le texte utile d'un site Web. Mais il nous a d'abord fallu outrepasser les protocoles de sécurités qui imposent une limite au scraping.

Catégorie	Arts	Health	News	Science	Society	Business	Games	Home	Recreation	Sport
Nombre d'entrées	11550	7020	2432	12532	11238	35764	3794	5910	11778	10522

TABLE 4.1 – Entrées par catégorie.

Compte tenus de certaines particularités propres à la hiérarchie de ODP, tel que le fait que deux catégories principales peuvent avoir un ensemble de sous-catégories identiques avec les mêmes sites Web indexés dedans, nous nous sommes très vites retrouvé avec un corpus non représentatif de chaque catégorie principale. Pour résoudre ce problème, nous avons décidé en premier lieu d'appliquer l'extraction sur un nombre réduit de niveaux, à savoir 3 niveaux, et ce afin d'éviter d'arriver au point où les sous-catégories se répètent dans différentes catégories principales, ensuite de ne pas prendre en compte les catégories principales trop vagues ou trop précises telles que « Regional » qui traite des sous-domaines « Society », « News », « Sports », etc., mais propre à chaque pays. Autrement dit, « Regional » a pour premier sous-niveau les pays, ensuite elle suit l'organisation générale de ODP mais spécialisée à chaque pays ; « Computers », « Reference », « Games » et « Kids and Teens ».

4.7.2 Traduction du corpus ODP

Selon ODP les corpus en langues Arabe et Française ne sont pas suffisamment riches et ne sont pas organisés de la même façon que celui en langue Anglaise. Comme expliqué précédemment, la solution est d'utiliser Google Translate et de concevoir notre propre corpus multilingue en traduisant les articles et les catégories de l'Anglais à l'Arbre et au Français. Pour ce faire, nous avons eu recours à Googletrans qui est une bibliothèque Python gratuite et illimitée qui a implémenté l'API Google Traduction. Cela utilise l'API Ajax de Google Translate pour effectuer des appels à

des méthodes telles que la détection et la traduction.

4.7.3 Classification d'un fichier selon ODP

4.7.3.1 Choix du modèle d'apprentissage

Afin d'atteindre une précision maximale pour la catégorisation de textes, nous avons fait une étude préliminaire où nous avons comparé, au moyen de la Précison, les différents modelés d'apprentissages, modèles fournis par la bibliothèque « Scikit-learn ». Le tableau ci-dessous rend compte des résultats obtenus :

Catégories	Arts	Health	News	Science	Society
MultinomialNB	0.75	0.31	0.39	0.46	0.90
K-NN	0.10	0.61	0.76	0.44	0.91
RANDOM FOREST	0.29	0.43	0.52	0.39	0.52
Classifieur par Arbre de Décision	0.29	0.49	0.42	0.33	0.31

TABLE 4.2 – Comparatif de différents Classifieurs

Catégories	Business	Games	Home	Recreation	Sports
MultinomialNB	0.52	0.89	0.74	0.37	0.92
K-NN	0.08	0.52	0.65	0.73	0.72
RANDOM FOREST	0.33	0.29	0.43	0.34	0.48
Classifieur par Arbre de Décision	0.07	0.21	0.38	0.25	0.40

TABLE 4.3 – Comparatif de différents Classifieurs

En analysant ces données, nous pouvons observer une précision des plus moyennes. On remarque, d'une part, que de ces quatre modèles, la classiffication par Arbre de Décision obtient les résultats les plus médiocres. D'autre part, comparativement aux autres modèles, MultinomialNB arrive à atteindre de bonnes précisions pour certaines des catégories, allant même jusqu'à 94%.

4.7.3.2 Classe non équilibrée

Les données déséquilibrées font généralement référence à un problème de classification où les classes ne sont pas représentées de manière égale. Par exemple, dans notre cas, nous pouvons clairement constater un déséquilibre entre la classe Business qui regroupe près de 106325 articles ou entrées pour notre algorithme d'apprentissage et News qui n'en comprend que 5516. Ce déséquilibre peut engendrer un mauvais apprentissage.
Pour pallier à ce genre de problèmes l'une des méthodes préconisées est de générer des échantillons synthétiques. Un moyen simple de générer des échantillons synthétiques consiste à échantillonner au hasard les attributs des instances de la classe minoritaire.

Il est possible d'échantillonner de manière empirique dans le jeu de données ou encore utiliser une méthode comme Multinomila Naive Bayes qui s'en chargerait.

Nous avons donc repris notre apprentissage mais cette fois ci sur des classes équilibrées, et voici les résultats obtenus :

Catégories	**Arts**	**Health**	**News**	**Science**	**Society**
MULTINOMIAL NB	0.86	0.53	0.95	0.99	0.99
K-NN	0.67	0.56	0.77	0.8	0.93
RANDOM FOREST	0.61	0.51	0.85	0.77	0.80
Classifieur par Arbre de Décision	0.71	0.53	0.71	0.69	0.70

Catégories	**Business**	**Games**	**Home**	**Recreation**	**Sports**
MULTINOMIAL NB	0.98	0.69	0.84	0.64	0.94
K-NN	0.69	0.77	0.51	0.84	0.75
RANDOM FOREST	0.93	0.69	0.77	0.78	0.83
Classifieur par Arbre de Décision	0.70	0.65	0.74	0.67	0.78

TABLE 4.4 – Tableau récapitulatif des précisions pour les catégories après équilibrage.

Nous constatons qu'un modèle se détache du lot. En présentant des résultats supérieurs aux autres, le Multinomial Naive Bayes offre une précision de 74% en langue Anglaise,65% en langue Arabe et 66% en langue Française. Sa matrice de confusion est la suivante :

Catégories	Arts	Health	News	Science	Society	Business	Games	Home	Recreation	Sports
Arts	839	168	0	0	0	0	46	23	68	1
Health	8	1439	0	0	1	0	27	21	24	0
News	16	77	186	0	0	0	36	4	33	29
Science	1	142	0	373	0	0	14	54	103	0
Society	15	91	0	3	346	0	60	15	41	1
Business	23	49	0	2	0	97	16	17	45	1
Games	26	151	0	0	0	0	883	15	79	11
Home	8	226	0	0	1	0	13	943	55	2
Recreation	15	144	0	0	0	0	20	36	925	0
Sports	17	152	0	1	0	0	67	9	46	776

TABLE 4.5 – Matrice de Confusion

En observant cette matrice nous constatons une matrice contenant une bonne partie des entrées de tests. Pour les catégories aux précisions les plus faibles nous remarquons une distribution équivalente des exemples sur le reste des catégories. Ce résultat pouvant être expliqué par un manque de précision dans le corpus collecté.

Recherche heuristique de la sous catégorie

La décision de la catégorie principale ayant été effectuée par apprentissage en utilisant l'algorithme Multinomila Naive Bayes, il s'agit maintenant de décider de la sous-catégorie précise où classer un fichier. Pour cela, nous avons utilisé une stratégie heuristique de Hill Climbing ou parcours en largeur. Afin d'effectuer le parcours d'une catégorie à une autre, à chaque nœud nous comparons le tf-idf du document actuel avec celui de l'ensemble des documents de la catégorie en question.

4.8 Scénario d'utilisation

4.8.1 Recherche Manuelle

L'interface de l'application offre à l'utilisateur un perçu général des différentes organisations mise à sa disposition. En effet, il a le moyen d'accéder à un fichier en suivant le chemin qui mène à ce dernier. Cette recherche manuelle est illustrée par le scénario suivant :

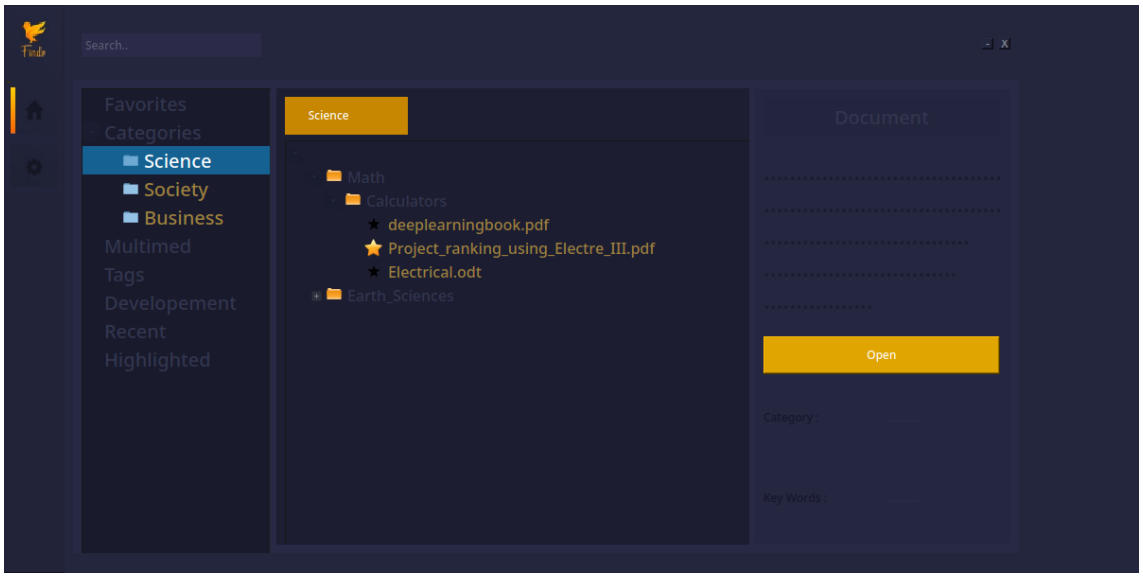


FIGURE 4.1 – Recherche Manuelle

4.8.2 Outil de Recherche

L'une des grandes forces de l'application est sa fonctionnalité de recherche avancée. Elle permet ainsi en un laps de temps réduit de trouver le ou les documents qui nous intéressent. La recherche propose ainsi un type de résultats varié :

— Recherche par tag : les documents ayant le ou les tags en question sont affichés sur la fenêtre. En effet, comme expliqué dans le chapitre 2, l'utilisateur a la possibilité d'annoter ses documents à l'aide de tags, ces tags lui faciliteront ainsi la recherche de ses documents et lui permettront une organisation entierement personnalisée.

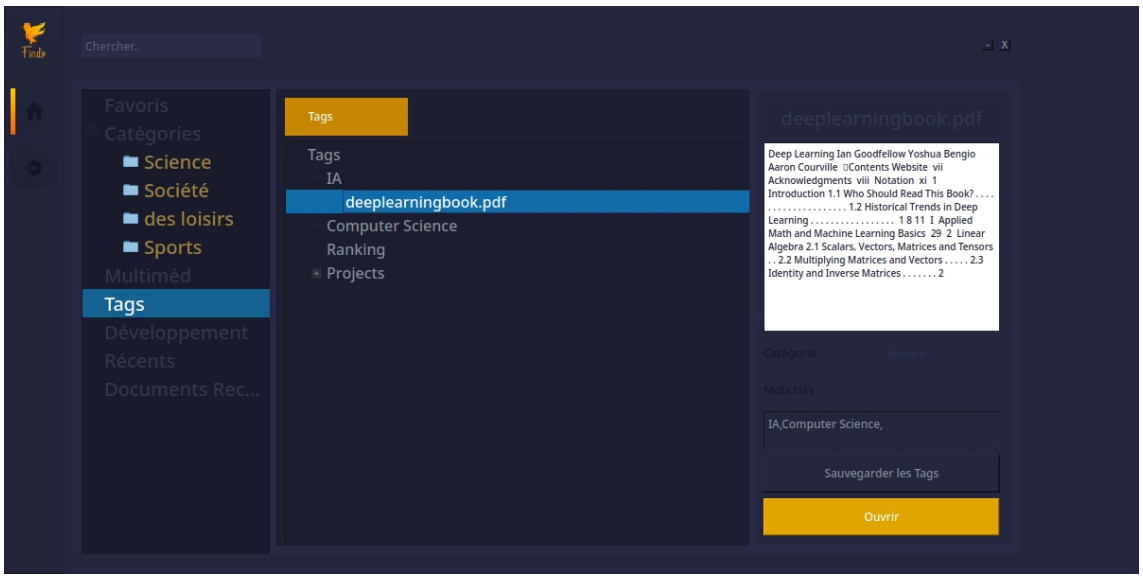


FIGURE 4.2 – Recherche par tag

— Recherche par Catégorie : si une catégorie portant ce nom est retrouvée non vide, ses documents sont retournés.

FIGURE 4.3 – Recherche par Catégorie

— Recherche par contenu : l'ensemble des documents contenant la séquence de mots ou leurs racines.

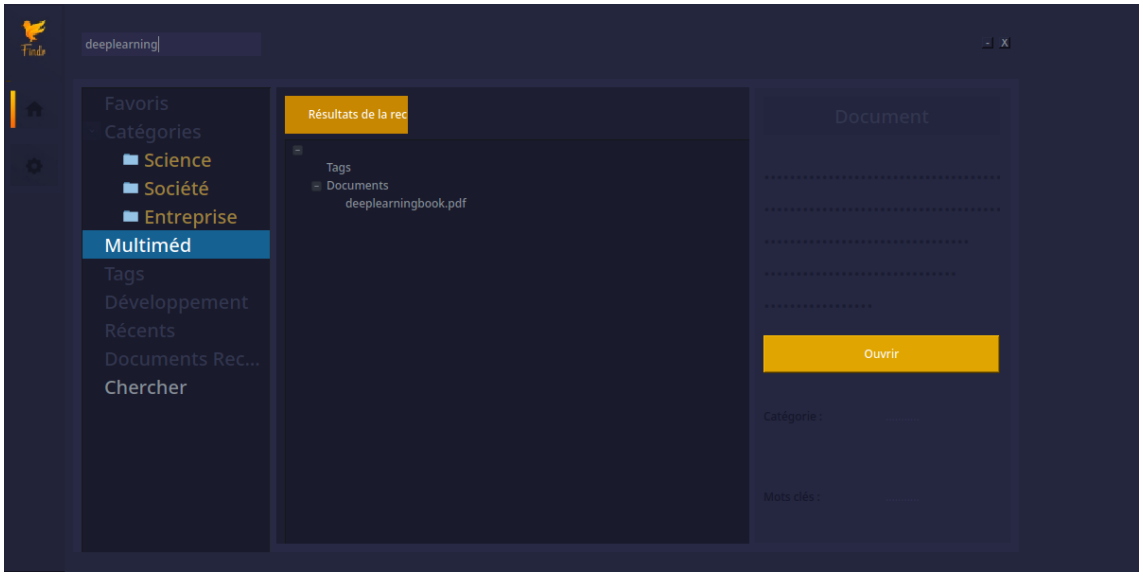


FIGURE 4.4 – Recherche par contenu

4.8.3 Documents Favoris

L'utilisateur peut choisir de mettre en favoris les documents qui l'intéressent afin d'y accéder plus facilement.

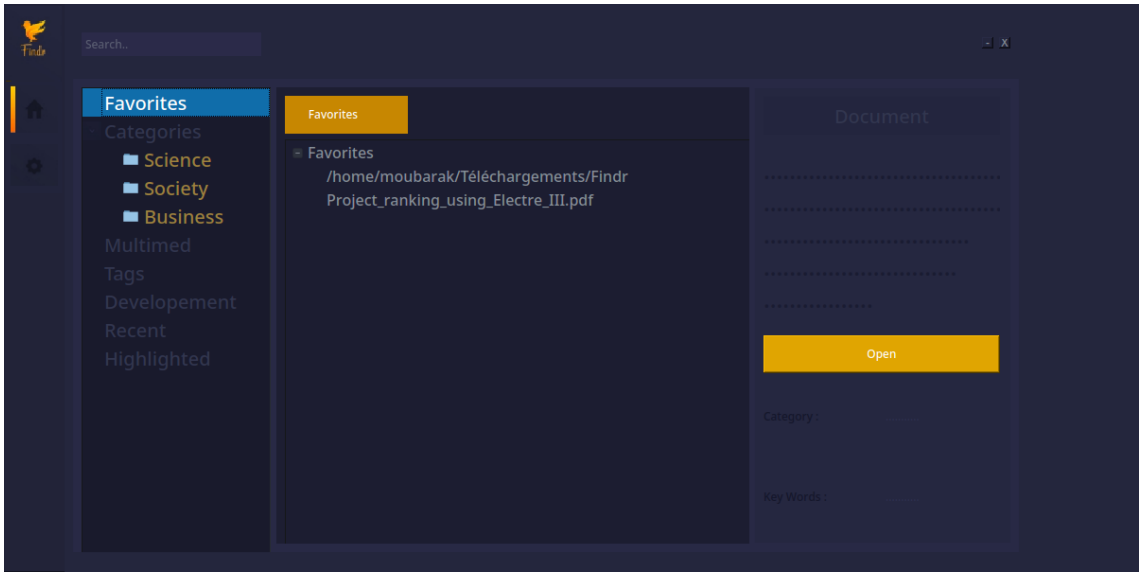


FIGURE 4.5 – Documents Favoris

4.8.4 Documents Pertinents

Le système se base sur la formule vue plus haut dans le mémoire, pour calculer le degré de pertinence prenant en compte la dernière consultation ainsi que le nombre d'accès au fichier en question.

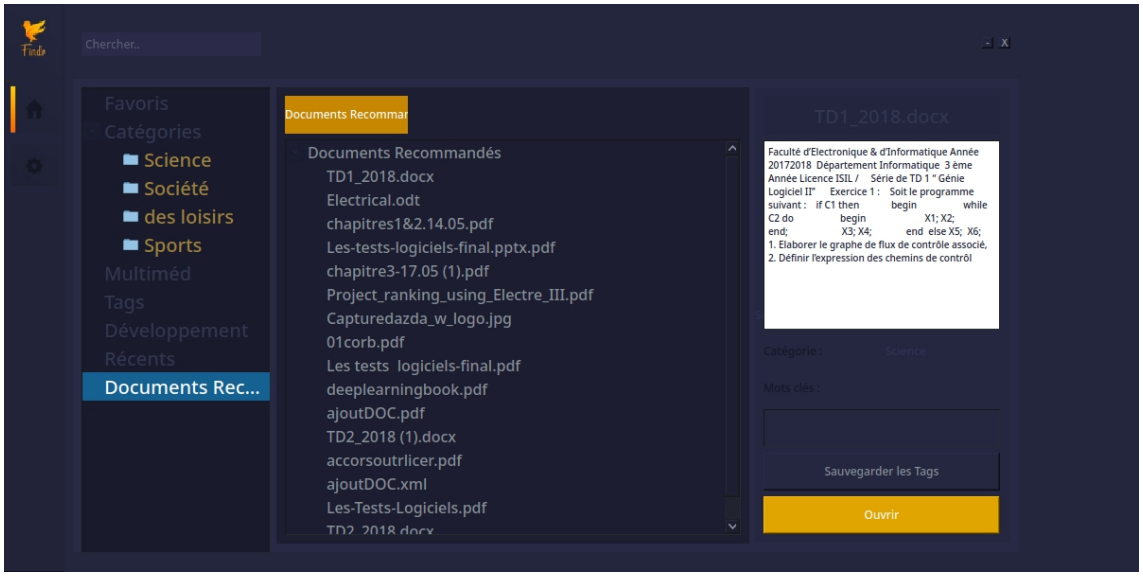


FIGURE 4.6 – Documents Pertinents

4.8.5 Documents Récents

Sous cet onglet l'utilisateur retrouvera tous ses fichiers récemment consultés.

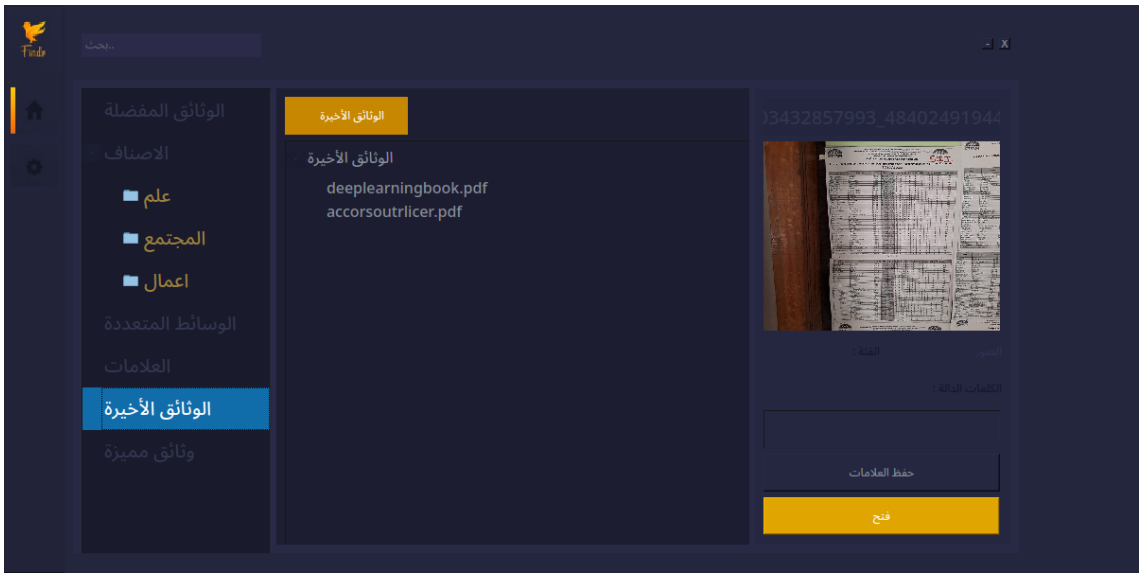


FIGURE 4.7 – Documents Récent

4.8.6 Ajouter un Tag à un fichier

Pour que l'utilisateur profite d'un système entièrement automatique, notre application offre à ce dernier la possibilité d'étiqueter chacun de ses documents avec une série de tags.

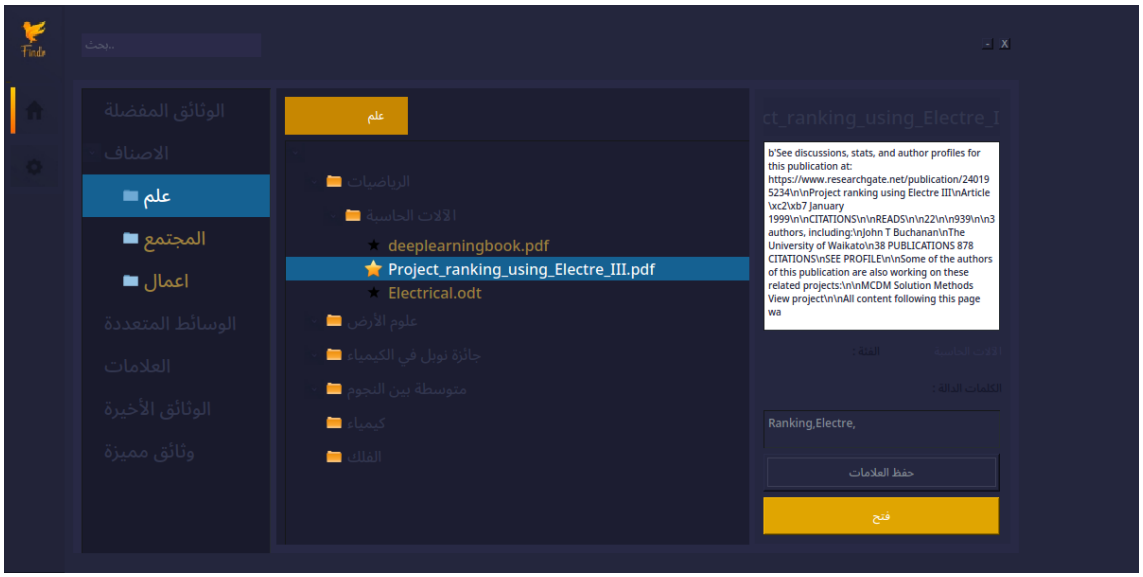


FIGURE 4.8 – Ajouter un Tag à un fichier

4.8.7 Optimisations

Notre implémentation a été optimisée par le fait que l'ensemble des mots clés des vecteurs représentatifs de nos documents est stocké dans un dictionnaire sous le format suivant :

$$dict[MotCl] = [Document1, Document2, ..., DocumentN].$$

Ici le mot clé est la racine du mot, et le vecteur de documents est celui des documents qui contiennent ce mot. Ainsi grâce aux fonctions de Hashage [2] integrées dans python, la vitesse est optimale.

4.9 Conclusion

Dans cette partie l'objectif est d'aboutir à un produit final, exploitable par les utilisateurs. Nous avons présenté en premier lieu, le modèle d'implémentation qui illustre l'architecture physique pour la construction de notre application. S'ensuit alors la partie réalisation pour finir avec une présentation des scénarios les plus généraux de notre application illustrés par des captures d'écrans.

2. Une fonction de Hashage est une fonction qui transforme les clés sous un format calculable, afin de permettre à l'avenir un accès plus rapide grâce, non pas à un accès séquentiel, mais au fait d'appliquer ce même calcul.

Conclusion générale et Perspectives

Le volume de données explose. Plus de données ont été créées au cours des deux dernières années que dans toute l'histoire de l'humanité. De plus, à titre d'exemple, en moyenne un document Word a une taille d'environ 200 Ko. Un rapide calcul permet donc de constater que dans un seul Téraoctet, l'équivalent de la capacité de stockage des ordinateurs personnels actuels, il est possible de stocker près de 5000 millions de fichiers. Comment un utilisateur peut-il alors se retrouver dans 5000 millions de fichiers ?

Il y a beaucoup d'efforts requis pour maintenir une carte mentale de l'endroit où se trouve chaque information et cette charge cognitive repose uniquement sur l'utilisateur. Or, plutôt qu'à rechercher l'information, elle serait mieux employée à la traiter. C'est donc de ce besoin urgent d'améliorer l'organisation des fichiers de l'utilisateur et de faciliter la recherche parmi eux que nos travaux ont vu le jour.

Notre système de gestion de fichiers a permis de venir à bout des problèmes inhérents au système de classification par dossiers en proposant une catégorisation automatique. Terminé les fichiers sauvegardés au mauvais endroit ; plus besoin d'avoir à se rappeler le nom de ses fichiers ; grâce à l'application, il suffit simplement de taper des mots clés pour retrouver les fichiers recherchés.
En plus de permettre un gain de temps énorme, l'application s'adapte aux besoins de chaque utilisateur, et offre une organisation globale du contenu d'un ordinateur via une seule application simple et intuitive à utiliser.

Tout au long de ce projet, plusieurs éléments propres à l'apprentissage automatique et plus précisément à la catégorisation de textes ont été abordés. Le premier défi a été de parvenir à une catégorisation satisfaisante. Pour cela il a d'abord fallu trouver un corpus d'apprentissage ce qui est primordial à une classification par apprentissage. La difficulté était surtout due au fait qu'il fallait trouver un ensemble de textes préalablement étiquetés et que ces étiquettes soient en accord avec nos besoins. Il fallait qu'elles soient explicites et qu'elles soient assez intuitives à l'utilisateur pour qu'il puisse facilement se familiariser à sa nouvelle organisation. A cet effet, des tests préliminaires ont été menés où nous avons comparé deux approches différentes. Suite à ces tests notre choix s'est tourné vers la hiérarchie de ODP, offrant une large gamme de catégories satisfaisant au mieux les besoins divers de chaque utilisateur.

Par la suite, un second défi est apparu en la nécessité de décider d'une technique de recherche pour compléter et affiner la catégorisation initiale, résultat du modèle d'apprentissage. La fonction de recherche Hill Climbing associée à la distance de

similarité de cosinus nous a permis de dépasser celui-ci et d'atteindre l'objectif de ces travaux, qui est de proposer un système intelligent de gestion automatique de fichiers.

Bien que notre système présente une approche novatrice de gestion de fichiers et offre un nombre important de fonctionnalités, quelques pas plus ambitieux restent tout de même envisageables.
D'une part, la catégorisation des fichiers contenant du code (source), en considérant les commentaires, constituerait une avancée de plus vers une organisation plus efficiente des projets de l'utilisateur. La classification de ces derniers ne se ferait plus selon le langage de programmation seulement mais s'affinerait pour représenter le thème abordé dans les commentaires.

D'autre part, dans le but de permettre une automatisation complète du système, il serait intéressant à l'avenir de procéder à une catégorisation sur les fichiers multimédias et de ne pas se contenter d'une gestion semi-automatique, établie sur des tags mais de leur appliquer un apprentissage profond afin de les regrouper en catégories permettant ainsi l'utilisateur de se repérer.

De plus, la structure modulaire de notre système favoriserait l'ajout d'extensions permettant de couvrir plus de langues, ciblant ainsi un public plus large.
Toujours dans cette optique de servir le plus grand nombre, il serait attrayant d'avoir ce système de gestion sur de multiple plateformes, disponible sous MAC, Windows ou encore Android.

Bibliographie

[1] Informations publiées dans le monde sur le net (en gigaoctets), mai 2018.

[2] Lefèvre.P. *La recherche d'informations : du texte intégral au thésauru.* Hermès Science, 2000.

[3] Lehireche.R Hamou.M. *Representation of textual documents by the approach wordnet and n-grams for the unsupervised classification (clustering) with 2D cellular automata : a comparative study.* PhD thesis, 2009.

[4] W. Pu, N. Liu, S. Yan, J. Yan, K. Xie, and Z. Chen. Local word bag model for text categorization. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 625–630, Oct 2007.

[5] Dmoz, mai 2017.

[6] Fellbaum.C. *WordNet : An Electronic Lexical Database.* MIT Press, 1998.

[7] Bernardo Magnini Luisa Bentivogli, Pamela Forner and Emanuele Pianta. *Revising WordNet Domains Hierarchy : Semantics, Coverage, and Balancing.* COLING 2004, 2004.

[8] Wen Zhang, Taketoshi Yoshida, and Xijin Tang. A comparative study of tf*idf, lsi and multi-words for text classification. *Expert Systems with Applications*, 38(3) :2758 – 2765, 2011.

[9] Archana Singh, Avantika Yadav, and Ajay Rana. Article : K-means with three different distance metrics. *International Journal of Computer Applications*, 67(10) :13–17, April 2013. Full text available.

[10] Nikolay Stanevski and Dimiter Tsvetkov. Using support vector machine as a binary classifier. In *International Conference on Computer Systems and Technologies–CompSys Tech*, 2005.

[11] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Text classification and Naive Bayes*, page 234–265. Cambridge University Press, 2008.

Annexe

Description des Classes

Méthode	Commentaire
init(self,lang='English',dev=False)	Initialise les données de configurations .
load(self)	Charge les informations déjà sauvegardées.
save(self)	Sauvegarde les information concernant les préférences de l'utilisateur.

TABLE 6.1 – Classe Configuration

Méthode	Commentaire
detect()	Détecte et renvoie la langue dominante du texte.

TABLE 6.2 – Classe Detector

Méthode	Commentaire
init(self,language)	Charge les modèles existants.
predict(self, document)	Prédit la catégorie principale du document.
process(self, text)	Se charge du prétraitement du texte.
tfidfer(self, entry, tidf=True)	pondère les termes d'un nouveau document.

TABLE 6.3 – Classe Categorizer

Méthode	Commentaire
ini(self,language)	Charge les modèles existants.
tfidf_fit(self, entry1, entry2,tfidf)	Crée et génère la représentation matriciel TFIDF.
tfidfer(self,entry1,entry2=None,tidf=True)	Applique la vectorisation tfidf existante sur un texte donné.
counter(self,entry)	Représente le fichier sous forme de Matrice avec CountVectorizer().
save(self)	Sauvegarde tous les modèles entrainés.
extractFeature(self, text)	Extrait les caractéristique du fichier.
transform(self, directory)	Renvoie le contenu des documents d'un répertoire donné.
loadModal(self)	Charge les modèles sauvegardés.
predict(self,document)	Prédit si le fichier est un fichier de configuration ou non.
find_mime_with_file(self,path)	Fait appel au système d'exploitation pour détecter le type du fichier.
majority(self, l)	Classe un dictionnaire en se basant sur les valeurs de celui-ci.
createDocument(self,path,name)	Si le fichier est traitable, elle instancit Document.

TABLE 6.4 – Classe Sorter

Méthode	Commentaire
init(self)	Charge l'arborescence, récupère la langue et crée les concepts associés.
initialize(self,directory)	Ajoute un répertoire à l'application, le parcourt et ajoute récursivement son contenu.
search(self,text)	Recherche d'un document par tags, contenu, catégorie.
contains(self,path)	Vérifie si le système contient un document donné.
moyenne(self,list)	Fais la moyenne des vecteurs représentatifs de chaque document d'une catégorie.
hillClimbing(self,vect,ti,totals,doc,cat,lise)	Recherche la sous-catégorie d'un document.
tfidfer(self,entries, language, tidf=True)	Applique la vectorisation TFIDF à un texte.
add(self, path,name)	Ajout d'un nouveau fichier.
getDoc(self, name)	Récupère l'instance d'un document dont on a le nom.
docs(self)	Retourne l'ensemble des documents que l'application traite.
save(self)	Sauvegarde les données de l'application.
getTags(self)	Récupère l'ensemble des tags existants.
load(self)	Récupère les données sauvegardées pour l'application.
verify(self)	Parcourt tous les fichiers, et met à jour s'il y a eu suppression ou ajout.
delete(self,file)	Supprime un fichier de l'application.

TABLE 6.5 – Classe HiddenTree

Méthode	Commentaire
init(self, concept,parent)	Initialise les attributs du Nœud .
addChild(self,node)	Ajoute un Nœud fils au Nœud courant .
empty(self)	Vérifie si le Nœud est vide .
hasChild(self)	Renvoie Vrai si le Nœud a un fils, Faux sinon .
add(self, document)	Ajoute un document au Nœud courant .
delete(self, document)	Supprime un document du Nœud courant .

TABLE 6.6 – Classe Node

Méthode	Commentaire
init(self,path, text, name, type = None, language = None)	Initialise les attributs.
setCategory(self, cat)	Permet la modification de la catégorie du document.
transform(in self, in text)	Décode le texte d'un document et en supprime tous les caractères spéciaux.
addTag(self,tag)	Ajoute un Tag d'un document.
deleteTag(self)	Supprime un Tag d'un document.
toStemme(self)	Fait la racinisation .
toTokens(self)	Représente le document en liste de Tokens.

TABLE 6.7 – Classe Document

Diagramme de Séquence

Arrivée d'un nouveau fichier :

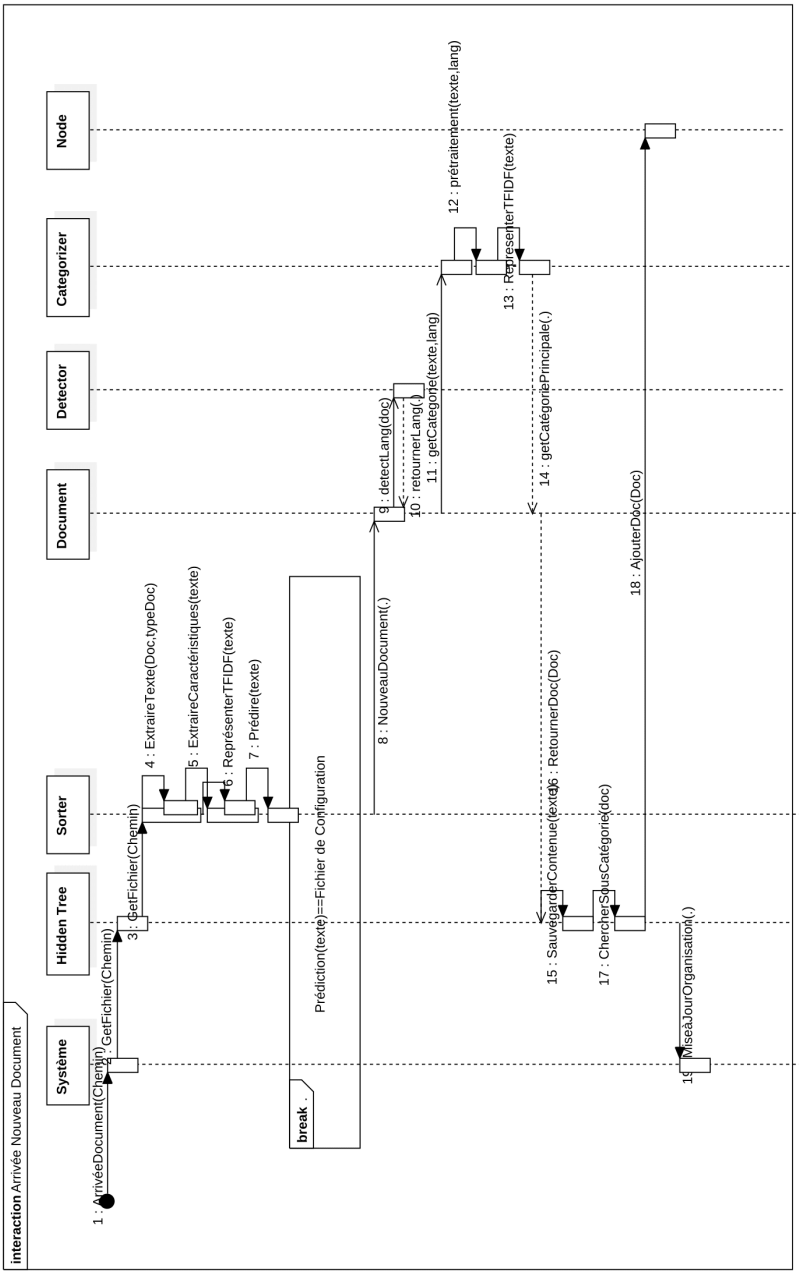


FIGURE 6.1 – Diagramme de Séquence de Recherche

Interface de l'Application

Interface Utilisateur en langue Arabe

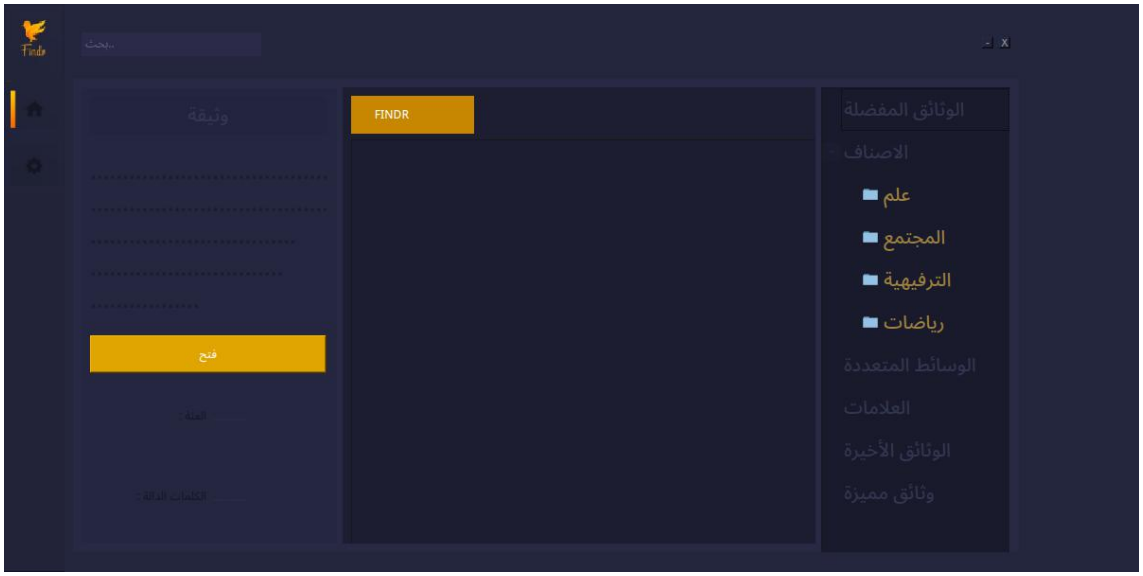


FIGURE 6.2 – Interface Utilisateur en langue Arabe

Configuration des paramètres

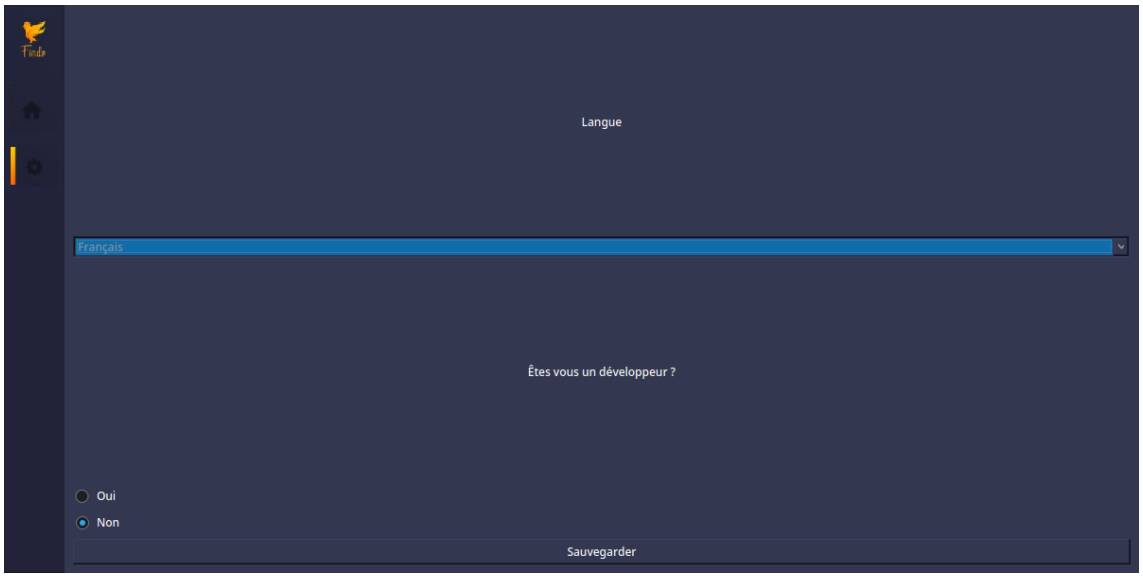


FIGURE 6.3 – Configuration des paramètres

Visualisation d'un Fichier

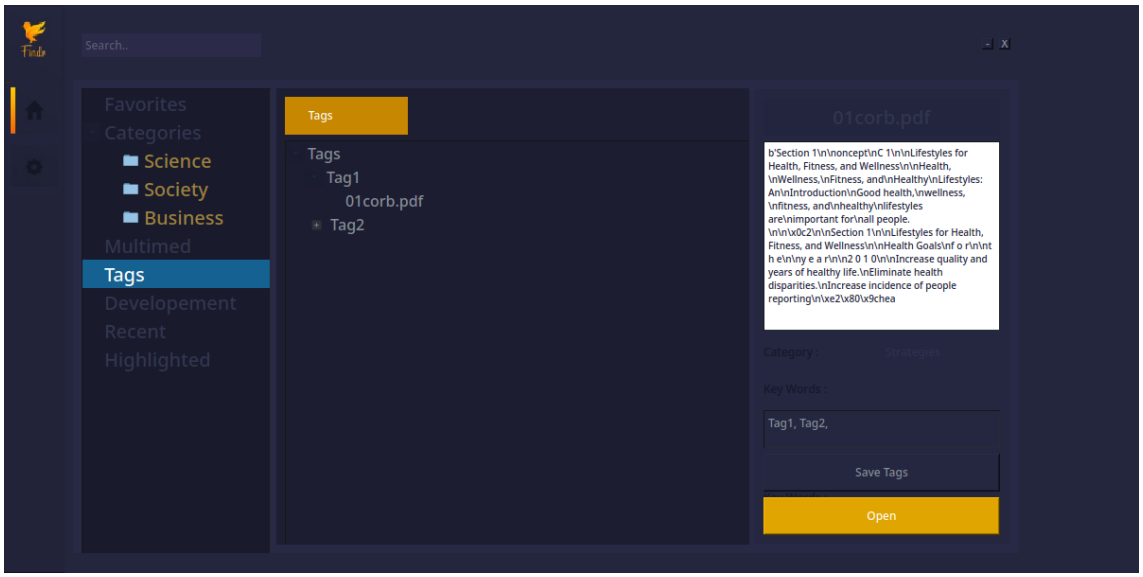


FIGURE 6.4 – Visualisation d'un Fichier

Résumé

La croissance exponentielle des données numériques et l'expansion de la capacité de stockage ont permis certes à l'utilisateur d'accumuler un nombre impressionnant d'informations utiles sur son ordinateur mais cela a créé un problème d'un nouveau genre : comment trouver l'information dont on a besoin, parmi toutes celles qui nous sont accessibles ?

Dans ce mémoire nous proposons notre solution qui est un système intelligent de gestion de fichiers sur son ordinateur. Après une première étape où nous procèderons a une critique des systèmes d'organisation classiques, nous nous efforcerons de combler les lacunes de ces derniers en proposant une nouvelle approche d'organisation entièrement automatisée, qui se base sur l'apprentissage automatique et plus précisément sur le domaine de la catégorisation automatique de textes. Ainsi, tout au le long de ce mémoire, nous détaillerons les différentes étapes de la conception à la réalisation de notre système intelligent de gestion de fichiers.

Mots-clés : Traitement Automatique du Langage Naturel ; apprentissage automatique ; classification ; indexation de fichiers.

Abstract

The exponential growth of digital data and the expansion of the storage capacity have certainly allowed the user to accumulate an impressive amount of useful information on his computer but this has created a problem of a new kind : How to find the information that one needs from amongst all those accessible ?

In this memoir we offer our solution which is an intelligent file management system on one computer. After a first step where we will proceed to a criticism of the classical organization systems, we will strive to fill the gaps of the latter by proposing a new approach of a fully automated organization based on machine learning and more precisely on the field of automated text categorization. Therefore, all along this memoir, we will detail all the different steps of the design to the realization of our intelligent file management system.

key words : Automatic Processing of Natural Language ; machine learning ; classification ; file indexing.