Université de Lorraine

7 Juin 2020

Projet PII : ANALYSE DES SENTIMENTS À PARTIR DE CRITIQUES DE FILMS

Auteurs :

Romain SCHALLER
Mohammed KRIMI
Julien GIET
Groupe 26

Superviseurs :

Sébastien DA SILVA
Gérald OSTER
Olivier FESTOR

Table des matières

Chapitre 1

Introduction

1.1 Contexe du projet

Le projet nous est donné dans le cadre de notre formation d'ingénieur de Télécom Nancy. Le sujet principal est l'analyse des sentiments, en s'appuyant sur une base de données, il faut déterminer si un commentaire donné est positif ou négatif.
Il nous est demandé de proposer des méthodes capables de répondre à certains besoins, en lien avec les problématiques sous-jacentes aux manipulations de l'analyse des sentiments.

1.2 Précisions importantes

Étant le premier véritable projet réalisé par les membres de l'équipe, nous l'avons mené avec une vigilance particulière pour les outils de gestion de projet.
Le plagiat est prohibé par l'article L335-3 du Code de la propriété Intellectuelle :*"Est également un délit de contrefaçon toute reproduction, représentation ou diffusion, par quelque moyen que ce soit, d'une oeuvre de l'esprit en violation des droits de l'auteur, tels qu'ils sont définis et réglementés par la loi."*.

Chapitre 2

Etat de l'art

2.1 Introduction

2.1.1 Analyse des sentiments

Histoire

Considéré comme l'un des domaines les plus riches en traitement automatique du langage naturel : "*TAL*" depuis les années 2000, l'analyse des sentiments était souvent restée attachée à une psychologie, individuelle ou collective, qui percevait plus ou moins l'émotion comme une perturbation de l'âme et du corps, opposée binairement à la raison. Ce psychologisme réducteur éloigna durablement les sciences sociales de son étude, d'autant que ce «mythe des passions» tendit à véhiculer une conception «hydraulique» des affects, mécaniquement subis par l'individu. [5]

Définition

Dans la littérature, l'analyse des sentiments est également appelée opinion Mining, opinion extraction, sentiment mining, subjectivity analysis, affect analysis, emotion analysis, review mining, appraisal extraction, est un domaine de recherche qui consiste à analyser les sensations, les attitudes et les émotions des individus vis-à-vis d'un produit, un service ou bien une organisation économique. L'analyse des sentiments est en effet un processus qui se base sur un grand nombre de données, d'un domaine précis, et les lier à un système de traitement de langage naturel afin de construire une idées sur des nouveaux objets de la même nature. [3]

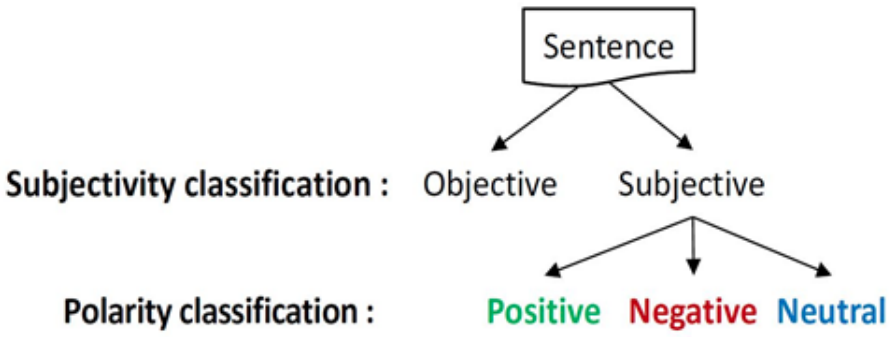

Figure 2.1 – Flux de l'analyse des sentiments

Les niveaux d'analyses

- **Niveau du document :** détermine la polarité d'un texte entier. L'hypothèse est que le texte n'exprime qu'une seule opinion sur une seule entité.
- **Niveau de la phrase :** détermine la polarité de chaque phrase contenue dans un texte. L'hypothèse est que chaque phrase dans le texte exprime une opinion unique sur une entité unique.
- **Niveau des aspects :** Effectue une analyse plus fine que les autres niveaux. Il est basé sur l'idée qu'une opinion consiste d'un sentiment et une cible (d'opinion). [3]

Algorithme d'analyse des sentiments

Il existe trois approche pour mettre en œuvre des systèmes d'analyse des sentiments : approche hybride, approche automatique, approche à base de règles. La dernière étant celle utilisée dans le cadre de ce projet on va la détailler par la suite.
Approche à base de règles : Cette méthode se base sur l'analyse sémantique des mots présents dans un texte indentifiant la polarité ou le sujet d'une opinion. Ce processus peut prendre en entrée deux types de données :

- Les entrées provenant du *TAL* et de ses techniques classiques : analyse syntaxique, morphologie, racinisation, délimitation de la phrase, racinisation...
- Les autres options se basant sur les dictionnaires du lexique et des mots de sentiments et d'opinions. Ainsi, ces algorithme essaye de faire correspondance avec les données du produit afin de determiner leur polarité.

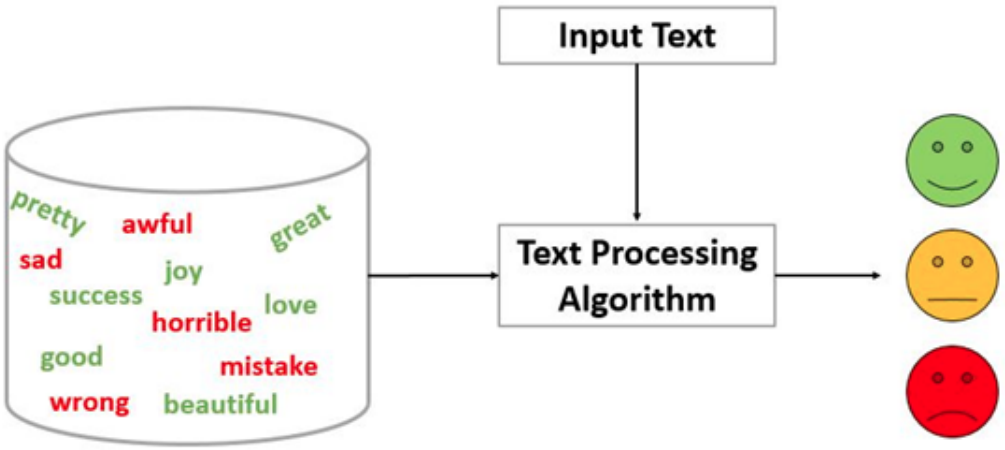

FIGURE 2.2 – Illustration de l'algorithmique de l'analyse des sentiments

2.2 Les tables de hachage

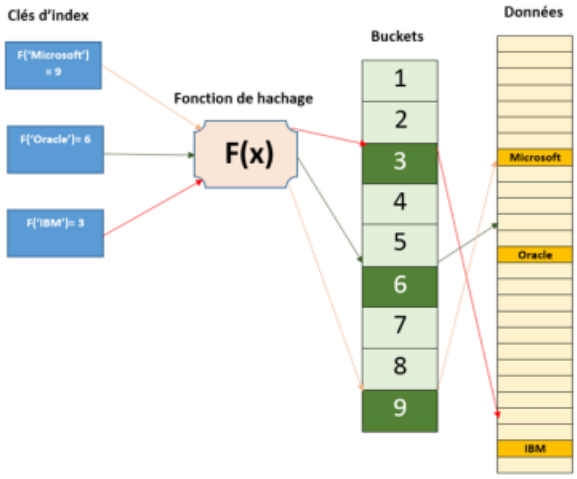

FIGURE 2.3 – Illustration de la structure

2.2.1 Definition

Un table de hachage est une structure de données informatique qui permet de définir un association **clé-valeur** pour chaque éléments afin de l'intégrer à la table.
Contrairemengt à un tableau ordinaire, une table de hachage ne conçoit pas un orde précis pour ses éléments. En effet, pour remplir la table et donner à chaque élément sa place dans la table, on fait appel à un outil dit **fonction de hachage**.

2.2.2 Fonction de hachage

Une fonction de hachage est une fonction qui prend en entrée un donnée fournie, et calcul en sortie un nombre qui correspondant à la case de l'élément dans le tableau. Ce nombre est appelé **hashcode**, il dépend de la fonction de hachage et de la donnée fournie.
Une fonction de hachage parfaite retournerait un **hashcode** unique pour chaque clé.

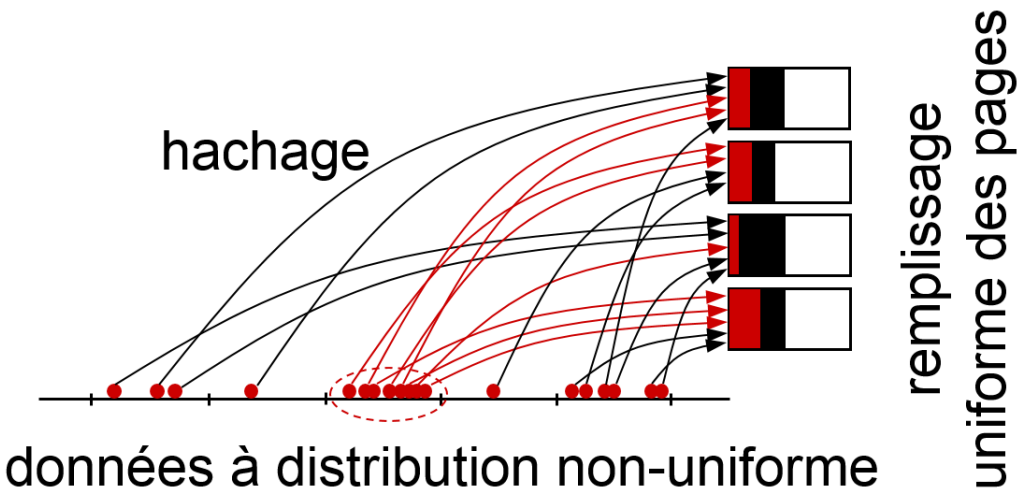

FIGURE 2.4 – Illustration du remplissage de la table en utilisant des hashcode

2.2.3 Pourquoi un table de hachage ?

Une table de hachage est une structure de type *Dictionnaire*, mais la compléxité de recherche dans une structure de dictionnaire classique est en $\boxed{\Theta(n)}$ si on considère que n est la taille du dictionnaire. En revanche, une table de hachage est une structure de données permettant d'effectuer des recherche en $\boxed{\Theta(1)}$. On peut epliquer la correspondance entre les deux structure de la façon suivante :

FIGURE 2.5 – D'un dictionnaire à une table de hachage

2.3 Liste chaînée

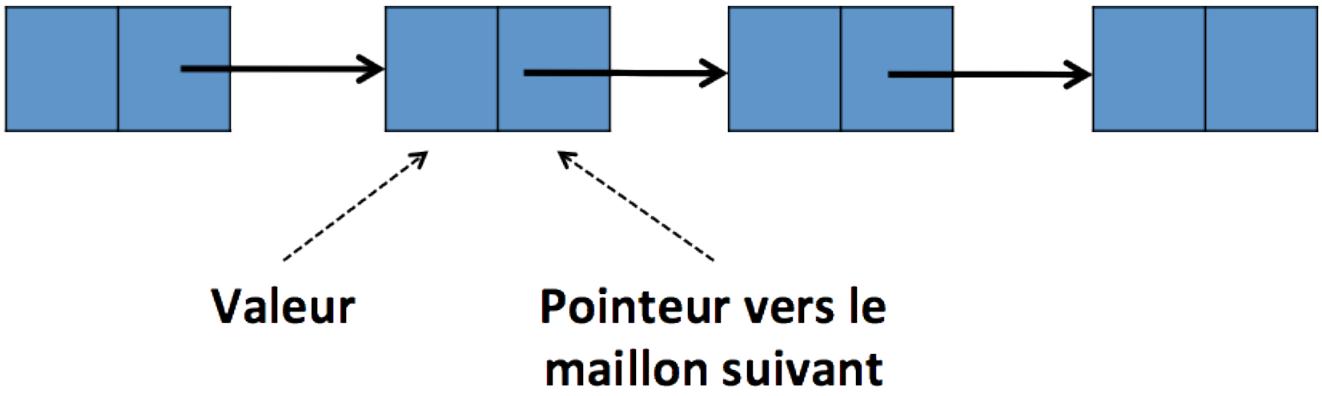

FIGURE 2.6 – Illustration de la structure

2.3.1 Définition

Une liste chaînée désigne en informatique une structure de données représentant une collection ordonnée et de taille arbitraire d'éléments de même type, dont la représentation en mémoire de l'ordinateur est une succession de cellules faites d'un contenu et d'un pointeur vers une autre cellule.

2.3.2 Liste chainée vs liste classique

Contrairement à une liste ordinaire, l'accès à un élément d'un liste chainée se fait d'un manière séquetielle en parcourant un partie de la liste, alors que l'accès à un élément pour une liste classique se fait d'un manière directe par le biais de son indice. [4]

	Avantages	Inconvéniants
Liste classiques	-Place en mémoire : éléments seulement -Accès aux données	-Place en mémoire : allocation initiale -Insertion/Suppression
Liste chaînées	-Place en mémoire : allocation à la volée -Insertion/Suppression	-Place en mémoire : utilisation des cellules -Accès aux données

2.4 Algorithme

2.4.1 Différentes approches

L'analyse des sentiments peut se faire grâce à deux méthodes principales :

Analyse sémantique

L'analyse sémantique consiste à étudier le sens des mots. Dans le cas d'analyse de critiques de films, on veut savoir si le mot a un sens positif ou négatif. Pour ce faire, il faut avoir à disposition une base de données dans laquelle le sens de mots ou de phrases est quantifié (par une note entre 0 et 4 dans notre cas).

Apprentissage automatique

Le but est içi d'utiliser le machine learning. Plusieurs possibilités existent alors :
-classification naîve bayésienne
-méthode des réseaux de neurones
-machine à vecteurs de support... [1]

2.4.2 Calcul du poids

Données

Lors de ce projet, un fichier de 8529 lignes nous a été fourni. La corpus du fichier se compose de plusieurs commentaires sur des films dont la forme est la suivante :

— Une note au début de chaque ligne du fichier décrivant le ressenti ***général*** du spectateur vis-à-vis du film. On distingue 4 catégories :

Catégorie	Descriptif
0	négatitif
1	quelque peu négatitif
2	neutre
3	quelque peu positif
4	positif

— Du texte explicitant les détails de la note et décrivant le ressenti du spectateur sous form d'un commentaire. Exemple :
" Like a less dizzily gorgeous companion to Mr. Wong 's In the Mood for Love – very much a Hong Kong movie despite its mainland setting ."

L'algorithme de calcul du poids

Afin de calculer le poids d'un mot du fichier texte donné comme support dans le projet, nous avons opté pour un algorithme qui parcourt le fichier une et une seule fois et pour chaque mot du fichier :

- **Si le mot n'a pas encore été rencontré :** on l'ajoute à la table et on incrémente son *nb* avec 1 et on donne à son *sum* la note du commentaire associé, **(cas 1)**
- **Si le mot a déjà été rencontré :** on le cherche et et on incrémente son *nb* avec 1 et on augmente son *sum* de la note du commentaire associé **(cas 2)**

En d'autres termes :

$t \leftarrow TableHachage$
$e \leftarrow ELEMENT$
$key(e) \leftarrow mot$
$temp \leftarrow note(commentaire)$
if $e \notin t$ **then** ▷ cf. cas 1
    $nb(e) \leftarrow 1$
    $sum(e) \leftarrow temp$
else ▷ cf. cas 2
    $nb(e) \leftarrow nb(e) + 1$
    $sum(e) \leftarrow sum(e) + temp$
end if

2.4.3 Mesure de la fiabilité de notre algorithme

Fiabilité du modèle

L'évaluation de la fiabilité de notre programme est une chose assez difficile étant donné qu'en théorie, nous ne connaissons pas la note réelle d'un commentaire. La seule chose qu'on peut faire est d'essayer de la calculer à l'aide de notre algorithme. Bien evidemment, on peut tout de même s'en sortir en evaluant la fiabilité de notre programme sur un jeu de commentaire pour lesquelles on connaît la note réelle. On peut alors comparer la note réelle d'un commentaire avec la note calculée par notre programme.

Outil de mesure : méthode EQM

La méthode EQM pour erreur quadratique moyenne ou MSE pour Mean Square Error, est une approche naïve d'évaluation de qualité pour notre modèle. Elle consiste à comparer la note calculée d'un commentaire avec la note réelle de celui-ci. En divisant par le nombre total de commentaire, on obtient une erreur moyenne pour la totalité du fichier
Formulation rigoureuse :

$$EQM(X,Y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2$$

Où :

- N est le nombre de commentaire pris en compte
- xi est la note calculée par notre algorithme
- yi est la note réelle

Plus l'EQM sera petit sur un jeu de commentaires, plus notre modèle sera bon.

2.5 Les tests

Les tests unitaires

Le test unitaire consiste à isoler une partie du code et à verifier qu'il fonctionne comme on l'espère. Ils ne permettent pas de s'assurer qu'il n'y a aucuns bugs, mais plutôt de valider la logique d'une unité de code. Très concrètement, on va isoler une fonction ou une partie d'une fonction. Puis on va proposer des valeurs à évaluer. Et on s'assure qu'on récupère en sortie ce à quoi on s'attendait.

Répertoire snow

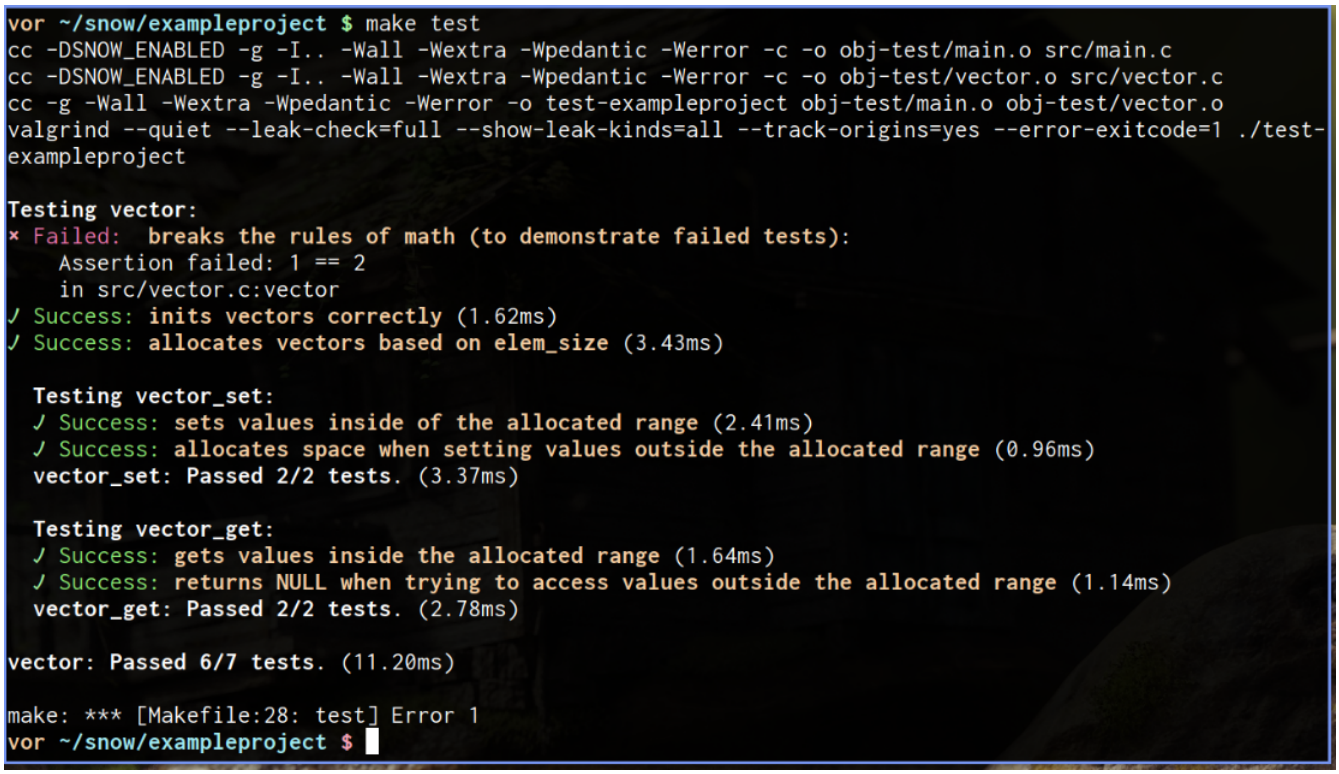

FIGURE 2.7 – Illustration du frameWork snow

Pour effectuer nos tests unitaires, on utilisera le répertoire snow. [2] Snow est très pratique à utiliser car ce n'est pas une libraire. Il suffit juste d'inclure un header dans notre .c et on peut écrire tous nos tests unitaires puis créer un exécutable. Ce répertoire offre en plus un frameWork très simple à comprendre et très ergonomique.

Chapitre 3

Gestion de projet

3.1 Equipe de projet

L'équipe projet se compose de trois étudiants en première année à Telecom Nancy :

Romain SCHALLER

Julien GIET

Mohammed KRIMI

Le chef de projet est Romain SCHALLER.

3.2 Organisation de l'équipe projet

3.2.1 Organisation immatérielle

L'équipe projet s'est réunie toutes les semaines à travers des appels Discord sur les serveurs de Télécom Nancy . La communication s'est également faite en continu à travers un groupe Messenger constitué de l'ensemble des membres de l'équipe projet.

3.2.2 Organisation matérielle

Afin d'assurer un bon encadrement et un suivi permanant du projet, différents outils ont été déployés :

Ordinateurs personnels : pour travailler sur le projet de chez nous.

Messenger : Afin de communiquer et transmettre les informations rapidement.

Le serveur Overleaf de Télécom Nancy : Afin de réaliser le rapport du projet ainsi que différents documents de Gestion de Projet.

Les tp proposés et réalisés auparavant : Ces tps nous ont permit de définir plusieurs structures de données qui nous ont été utiles, ainsi que l'écriture des tests unitaires, la gestion de la mémoire et du débuggage des programme et la découverte du langage C.

Gitlab de Télécom Nancy : Afin de partager les fichiers du code et le maintenir à jour pour l'ensemble de l'équipe projet.

3.3 Analyse du projet

3.3.1 Matrice RACI

	Julien GIET	Mohammed KRIMI	Romain SCHALLER
Cahier des charges	R	RA	R
Etat des arts	R	RA	R
Diagramme Gantt		RA	
Matrice SWOT			RA
Matrice RACI	RA	R	
Implémentaton des structures de données		RA	RA
Création des librairies	RA	R	
création des tests sur librairies			RA
fonctions interagissant avec l'utilisateur	RA	R	
pretraitement des données			RA
passage de .txt à table de hachage		RA	R
Utilisation de la table	RA		
infos sur les tables (stats)	RA		
test des fonctionalités	A	A	RA
Rédaction du rapport	R	RA	R
README		RA	

R : Réalise A : Autorité C : Conseil I : Informe

3.3.2 Matrice SWOT

STRENGTHS

1. Compétences techniques.
2. Bonne entente de l'équipe.
3. Ce n'est pas le premier projet des membres du groupe.

WEAKNESSES

1. Peu d'expérience en C .
2. Manque d'expérience dans la manipulation des structures de données dans un grand projet.
3. Découverte de l'analyse des sentiments.

OPPORTUNITIES

1. Confinement : On peut se concentrer sur le projet.
2. Acquérir de nouvelles compétences en C et gestion de projet.
3. Manipulation de nouvelles structures de données.

THREATS

1. Manque de temps avec les partiels.
2. Manque de motivation avec le confinement.
3. Cadre parfois compliqué avec le confinement .
4. Manque de contact avec les profs à cause du confinement.
5. Chanboulement organisationel avec le déconfinement.

3.3.3 Compte-rendus des réunions

Compte-rendu du 24 Avril 2020

Présents :
Romain SCHALLER — Durée : 1h
Mohammed KRIMI — Heure : 15h
Julien GIET — Lieu : Serveur discord de l'école

Nous avons tout d'abord déterminé les rôles des membres de l'équipe :

- Romain SCHALLER : Chef de projet
- Mohammed KRIMI : Secrétaire

Nous avons ensuite enchaîné sur le partage des tâches concernant la gestion du projet : matrice RACI, matrice SWOT, diagramme Gantt et charte de projet.
Ensuite nous avons discuté sur la disposition de notre projet sur la plateforme gitlab :

- Ajout des branches nécessaires
- Choix de la bibliothèque ***Snow*** pour faire les tests unitaires
- Faire des recherches concernant le sujet de l'analyse des sentiments afin d'élaborer un plan lors de la prochaine réunion.

Compte-rendu du 01 Mai 2020

Présents :
Romain SCHALLER — Durée : 2h
Mohammed KRIMI — Heure : 15h
Julien GIET — Lieu : Serveur discord de l'école

Discussion entre les membres du groupe sur le plan à adopter ainsi que la structure de données adéquate :

- **Calcul du poids de chaque mots :** deux méthodes ont été proposées. La première se base sur le codage d'une fonction qui calcule chaque fois en parcourant tous le fichier plusieurs fois le poids d'un mot. La deuxième méthodes étant plus optimale en terme de complexité **(*cf. Chaptre 4*)** qui parcours une seule fois le fichier et attribut vers la fin à chaque mot une somme de notes et un nombre d'apparence.
- **Structure de données :** décision d'utiliser une table de hachage, où chaque case de la table va pointer sur une liste et chaque liste va contenir des éléments ;ainsi chaque mot sera associé à deux flottants : sum et nb **(*cf. Chaptre 4*)**.

Compte-rendu du 08 Mai 2020

Présents :
Romain SCHALLER — Durée : 30min
Mohammed KRIMI — Heure : 15h
Julien GIET — Lieu : Serveur discord de l'école

Lors de cette réunion, on a fait le point sur l'avancement du codage des fonctions fondamentales pour l'élaboration de la table de hachage. Puis nous avons distribuer les autres taches nécessaires pour terminer l'étape minimale. Ensuite, nous avons fait le point sur l'avancement des outils de gestion de projet et la rédaction du rapport. Enfin, on a fixé un jalon pour le 16 Mai afin de finir cette étape et commencer le prétraitement additionnel du fichier.

Compte-rendu du 16 Mai 2020

Présents :
Romain SCHALLER — Durée : 30min
Mohammed KRIMI — Heure : 14h
Julien GIET — Lieu : Serveur discord de l'école

Une réunion d'avancement qui a porté sur le jalon fixé auparavant dans la réunion du 08 Mai 2020. Nous avons ensuite fait le point sur la façon dont on va tester notre code. Enfin, nous avons distibuer les tâches concernant le prétraitement additionnel du fchier texte donné afin d'améliorer les résultats de notre programme et augmenter sa précision.

Compte-rendu du 20 Mai 2020

Présents :
Romain SCHALLER — Durée : 2h30
Mohammed KRIMI — Heure : 14h
Julien GIET — Lieu : Serveur discord de l'école

Cette réunion -urgente- a porté sur un problème concernant notre structure de données et son implémentation ainsi que le code de la première partie. On s'est réuni pendant une longue durée pour essayer de debugguer ensemble notre implémentation et refaire toutes les fonctions et les structures mais sans résultat concluant. Il a alors été décidé de travailler chacun de son côté pour résoudre le problème en restant en contact sur le groupe messenger.

Compte-rendu du 27 Mai 2020

Présents :
Romain SCHALLER — Durée : 45min
Mohammed KRIMI — Heure : 16h
Julien GIET — Lieu : Serveur discord de l'école

Cette réunion a porté principalement sur les tests à effectuer sur notre table de hachage et les résultats obtebus ainsi que la partie statistique.
D'une part, pour le test de la table de hachage il a été décidé d'utiliser la méthode EQM décrite dans l'état de l'art.
D'autre part, pour l'étude statistique on a choisit principalement de déterminer la fréquence de chaque mots, ceux les plus fréquents, ceux les moins fréquents, ceux les mieux notés, ceux les moins notés, la densité des mots dans chaque intervalle de critique ainsi que la moyenne et la variance des notes de chaque mot.

Compte-rendu du 2 Juin 2020

Présents :
Romain SCHALLER
Mohammed KRIMI
Julien GIET

Durée : 45min
Heure : 14h
Lieu : Serveur discord de l'école

Lors de cette réunion on a fait le point, sur l'avancement de toutes les tâches non encore finalisées y compris le rapport du projet. Ensuite, on a enchaîné sur le choix de l'organisation de notre menu final et toutes les options qu'il fallait inclure afin de permettre à l'utilisateur la meilleure expérience. Enfin, le groupe projet a choisit les parties qui vont constituer le **README** qui va accompagner notre projet.

Compte-rendu du 6 Juin 2020

Présents :
Romain SCHALLER
Mohammed KRIMI
Julien GIET

Durée : 45min
Heure : 14h
Lieu : Serveur discord de l'école

La dernière réunion de notre projet avait pour but de mettre le point sur toutes les tâches fixées par le groupe projet dès le début et vérifier leurs avancement. Ensuite, nous avons évalué notre position par rapport aux objectifs du projet afin de décrire parfaitement cette expérience dans le rapport. Enfin, nous avons déterminé la date de fin de notre rapport vis-à-vis de la date du rendu.

3.3.4 Diagramme Gantt

Le diagramme représenté dans la figure suivante est le digramme Gantt du projet.
A noter : Dans ce diagramme nous avons représenter juste les grandes taches et parties du projets, afin de prendre une meilleure idée sur toute les tâches réalisées, vous pouvez voir la ***RACI*** **(cf *Chapitre 3.3.1*)**

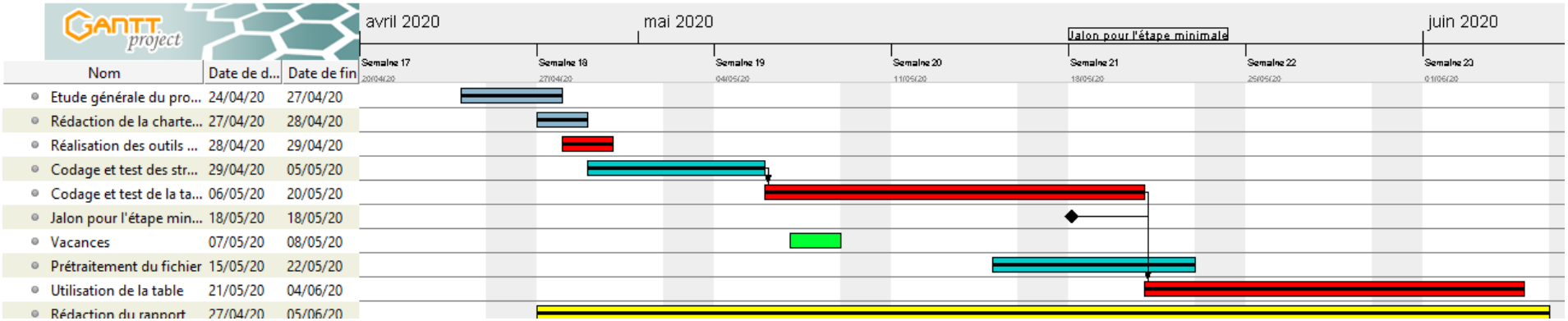

FIGURE 3.1 – Caption

3.4 Charte du projet

Présentation du groupe L'équipe est composée de trois élèves de première année du cursus ingénieur de Télécom Nancy.

— M. Mohammed Krimi : Secrétaire

— M. Julien Jiet

— M. Romain Schaller : Coordinateur

3.4.1 Description du client

Le projet est effectué à la demande des membres de l'équipe éducative de Télécom Nancy.

— M. Sébastien Da Silva, enseignant en informatique à l'école.

— M. Gérald Oster, enseignant en informatique à l'école.

— M. Olivier Festor, directeur, enseignant en informatique à l'école.

3.4.2 Problématique

Problématique principale et prioritaire :

— Comment traiter et interpréter de manière informatique les commentaire ou avis écrits ?

Problématiques secondaires :

— Comment récupérer les données contenues dans la base de données fournie ?

— Quelle est l'implémentation de structure de données la plus adaptée à notre problème ?

— Comment utiliser les données récupérées pour interpréter un commentaire ?

3.4.3 Objectifs

— Exploitation correcte des outils de gestion de projet (Gantt, Charte projet, RACI, SWOT)

— Maîtriser les outils mis à notre disposition (gitlab, sharelatex, outils de debug : gdb ; valgrind ; … , bibliothèque de test snow, outil makefile).

— Maîtriser le thème de l'analyse de sentiments et faire de nos connaissances un état de l'art.

— Proposer un algorithme répondant aux problématiques soulevées en essayant de le rendre le plus performant possible.

— Maîtriser la gestion de notre base de donnée et de structures de données adaptées au problème.

3.4.4 Utilisation

L'utilisateur peut taper make en ligne de commande à la racine du projet pour ouvrir le menu, où différentes options s'offrent à lui.

Chapitre 4

Code

4.1 Description

Notre code a pour but de permettre à l'utilisateur de :

- Estimer une critique **(*cf. Chapitre 1 partie 2.2*)** sur un commentaire saisi en ligne de commande.
- Effectuer un traitement sur le fichier contenant la base de donnée afin d'optimiser les réultats.
- Faire des études statistiques sur les données du fichier.
- Mesurer la fiabilité de notre modèle de prédiction.

4.2 Menu

Option 1 : Afficher la critique associée à votre commentaire

Option 2 : Tester le menu avec la méthode EQM

Option 3 : Récupérer ddes données statistique sur le fichier des commentaires

Option 4 : Quitter le programme.

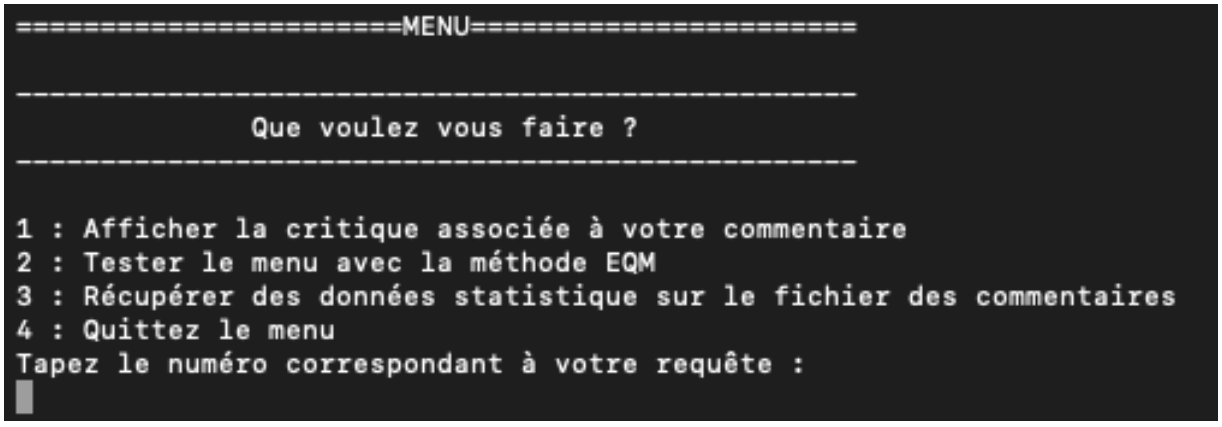

FIGURE 4.1 – La forme du menu dans une ligne de commande

Voir ci-après pour le détail de chaque option :

4.2.1 Option 1

Description

En choisissant l'option 1 du menu, l'utilisateur est amené à saisir le commentaire qu'il veut évaluer et afficher sa critique.
Après cet affichage, la console demande à l'utilisateur s'il veut effectuer une autre requête, s'il tape 1, il va être envoyer directement au menu principale, sinon il va quitter le menu.

Illustration

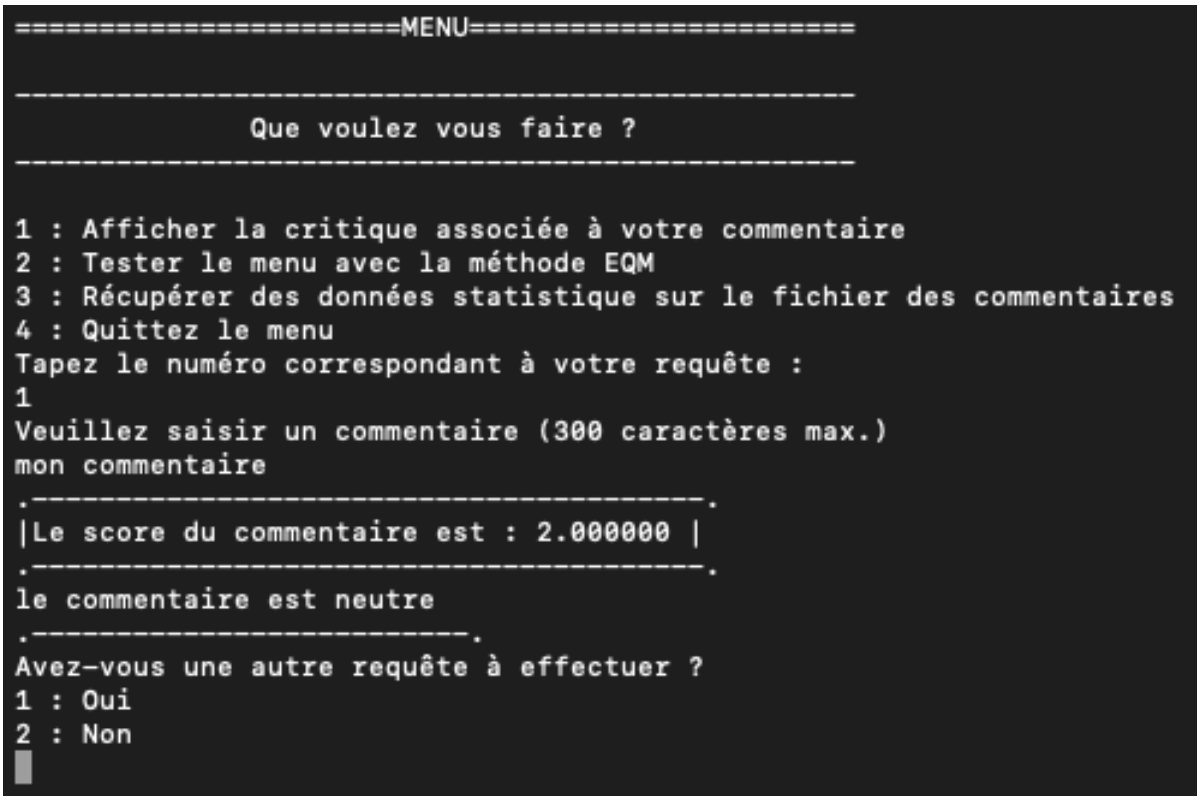

FIGURE 4.2 – L'option 1 du menu pricipal

4.2.2 Option 2

Description

L'option 2 du menu nous permet de tester le menu.
Une fois selectionnée, l'utilisateur est amené á saisir le path du fichier de données en premier lieu. Après, l'utilisateur à plusieurs options à choisir nécessairement pour effectuer le test :

- La sensibilité à la casse du fichier,
- Le nombre de lignes sur lesquel on veut tester le fichier,
- Le nombre de lettres composant les mots du fichier et leur fréquence d'apparition comme paramètres de l'évaluation.

De même que l'**option 1**, après l'affichage du résultat, la console va demander à l'utilisateur s'il veut effectuer une autre tâche, et exécuter le programme correspondant en fonction de sa réponse.

Illustration

FIGURE 4.3 – L'option 2 du menu pricipal

4.2.3 Option 3

Description

L'option 3 du menu nous permet de faire une étude statistique élargie sur le fichier des données. Quand elle est séléctionnée, l'utilisateur est renvoyé directement vers un sous-menu permettant de :

- Calculer le nombre total des mots du fichier,
- Calculer le nombre de mots différents du fichier,
- Calculer les n nombres les plus fréquents,
- Calculer le nombre des mots dans n intervalles critiques entre 0 et 4,
- Calculer les n mots les mieux notés,
- Calculer les n mots les moins bien notés,
- Calculer la moyenne et la variance des notes de la table.
- Retour au menu principal,

— Quitter le menu.

Une fois le résultat affiché, il sera automatiquement proposé à l'utilisateur de choisir s'il souhaite effectuer une autre requête statistique ou bien s'il désir être amener au menu principal.

Illustration

FIGURE 4.4 – L'option 1 du menu pricipal

4.3 Codage de la table de hachage

4.3.1 Structure associée

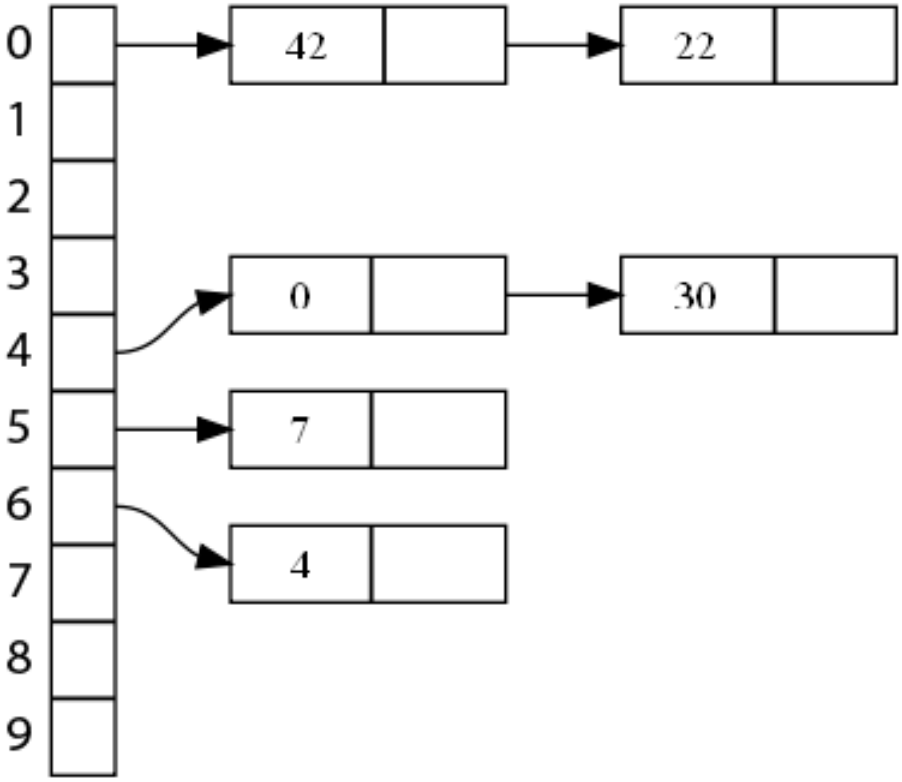

FIGURE 4.5 – Illustartion de la structure de notre table

Notre structure de données se compose de trois éléments essentiels :

ELEMENT : C'est un pointeur vers une structure qui se compose d'un pointeur vers une chaîne de caractère : **Clef**, et de deux entiers : le premier ***sum*** va contenir la somme des notes des commentaires où la **Clef** est présente et le deuxième ***nb*** le nombre d'apparence de la **Clef** dans tout le fichier.

```
1 typedef char* Clef;
2 struct ELEMENT
3 {
4     Clef key;
5     int sum;
6     int nb;
7 };
8 typedef struct ELEMENT *ELEMENT;
```

Liste : C'est un pointeur vers une autre structure appelée **cellule** dans notre programme. Cette dernière se composant principalement d'un **ELEMENT** ***elm*** et d'un pointeur vers une autre **cellule** ***suiv***.

```
typedef struct cellule cellule;
struct cellule {
    ELEMENT elm;
    cellule* suiv;
};
typedef cellule* Liste;
```

TableHachage : c'est un pointeur vers une structure appelée **tableHachage**. Cette dernière contenant un pointeur vers une **Liste *tableauListe*** et un entier décrivant la taille de la table ***tableauListe***.

```
typedef struct table tableHachage;
struct table
{
    Liste* tableauListe;
    int tailleTableau;
};
typedef tableHachage* TableHachage;
```

4.3.2 Codage

Tableau des fonctions utilisées sur les listes chainées

Fonctions	Description	Complexité
list_creer()	Elle permet de créer une liste chainée	**O(1)**
list_estVide()	Elle permet de tester si une liste est vide	**O(1)**
list_ajoutFin()	Elle permet d'ajouter un élément à la fin d'une liste	**O(n)**
list_rech()	Elle permet de chercher un élément dans une liste	**O(n)**
list_visualiser()	Elle permet d'afficher le contenu d'une liste	**O(n)**
list_incrementation()	Elle incrémente le sum et le nb d'un élément précis de la liste	**O(n)**

n : la taille d'une liste

Tableau des fonctions utilisées sur les tables de hachage

Fonctions	Description	Complexité
tableHachage_creer()	Cette fonction créer une table de hachage de taille t	**O(n)**
tableHachage_get()	Cette fonction revoie l'élément de clé c s'il est dans la table t	**O(n)**
tableHachage_incrementation()	Cette fonction mets à jour les valeurs de l'élément de clé c dans la table t (sum<-sum+temp et nb<-nb+1)	**O(n)**
hachage()	La fonctrion calculant le hash d'une chaine de caractères	**O(m)**
tableHachage_ajout()	Cette fonction ajoute l'élément e à la table t	**O(n)**

n : la taille de la table de hachage

m : la longueur d'un mot

Tableau des fonctions utilisées pour remplir la table

Fonctions	Description	Complexité
file_open()	Les paramètres de la fonction : char* s. Cette fonction nous permet d'ouvrir le fichier et tester s'il est bien ouvert ou pas.	$O(1)$
remplissage()	Les paramètres de la fonction : char* s, TableHachage t, int temp Cette fonction ajoute le mot s à la table s'il n'existe pas, sinon elle incrémente nb et sum correspondant à cet élément.	$O(m)$
commentaire_fichier()	Les paramètres de la fonction : FILE* filep. Cette fonction permet de calculer le nombre des commentaire d'un fichier	$O(n)$
traitement_ligne()	Les paramètres de la fonction : FILE* filep, char tab[][MAXLIGNE]. Cette fonction stocke chaque ligne fichier dans une case du tableau tab	$O(n)$
gestion()	Les paramètres de la fonction : FILE* filep. Cette fonction permet de stocker chaque mot du fichier et le mettre dans la table de hachage en utilisant le processus de remplissage()	$O(n^2m)$

n : le nombre des lignes d'un fichier

m : la taille de la table de hachage

4.4 Codage de l'évaluation d'un commentaire

4.4.1 Description

La fonction evaluer sert à attribuer une note à un commentaire donné. Elle prend en entrée la table de hachage **t** contenant les mots de la base de données, leur nombre d'occurences et leur score. Elle prend aussi un entier **nbl** et un flottant **f**. La fonction détermine la note du commentaire en calculant la moyenne des scores de chaque mot (en se basant dans la table). Si le mot n'est pas dans la table, son score associé est 2 (score neutre). De plus, la fonction ignore les mots de nbl lettres ou moins (en effet les mots courts sont souvent des mots syntaxiques et n'apportent donc aucun renseignements sur l'avis de l'auteur du commentaire). La fonction ignore aussi les mots qu'elle "détecte" comme étant neutre. Cette détection regarde les mots qui ont une note proche de 2 avec une fréquence importante (fréquence **f**). La fréquence importante permet d'assurer la fiabilité du score associé au mot, celui-ci étant présent dans de nombreux commentaires de la base de données.

4.4.2 Complexité

Une première boucle while parcourt les mots du commentaire. Dans cette boucle, on fait appel à tableHachage_get qui parcourt dans le pire des cas toute la table. Le reste des opérations se fait en **O(1)**. On a donc une complexité totale en **O(mn)** avec **m** le nombre de mots du commentaire traité et **n** la taille de la table de hachage fois le nombre moyen de collision.

4.5 Codage de l'obtention de statistiques

n : la taille de la table de hachage

m : le nombre moyen de collision

Fonctions	Description	Complexité
motsDifs(TableHachage t, int n)	Cette fonction renvoie et affiche le nombre de mots total de la base de donnée si **d**=0 et le nombre de mots différents sinon	$O(nm)$
freq(TableHachage t, int n)	Cette fonction renvoie et affiche les **n** mots les plus fréquents dans la base de données	$O(nm)$
repartition(TableHachage t, int n)	Cette fonction renvoie la répartition des mots selon leur note en **n** intervalles	$O(nm)$
motsPos(TableHachage t, int n)	Cette fonction renvoie les **n** mots les mieux notés de la base de données	$O(nm)$
motsNeg(TableHachage t, int n)	Cette fonction renvoie les **n** mots les moins bien notés de la base de données	$O(nm)$

4.6 Codage des prétraitements additionnels des données

4.6.1 Description

Ce dossier sert exclusivement à réaliser un traitement additionnel sur un jeu de données. En effet le but va être de passer tous les caractères en majuscules. On va aussi chercher à remplacer les n't par not et à supprimer les 's. Enfin, on sépare les mots par des points.
De manière pratique, on appelle la fonction principale fileTreatment qui va générer un nouveau fichier : "traited.txt" qui sera un copié collé du fichier d'entrée mais avec les modifications faîtes.

4.6.2 Codage

Fonctions	Description	Complexité
char* ligne_treatment(const char* ligne);	Cette fonction va prendre en paramètres une ligne et va : tout mettre en majuscule mettre des points comme séparateur des mots remplacer n't par not supprimer les 's	$O(nb\_caractres^2)$
char* str_replace(...);	Cette fonction va prendre en paramètres trois Const Char* : Le premier est une ligne Le deuxième est un motif à remplacer Le troisièrme est le motif remplacant Elle va donc enlever les motifs avant par les motifs nouveaux.	$O(nb\_caractres)$
char* casse(const char* mot);	Cette fonction va passer toutes les lettres d'un mot en MAJUSCULE.	$O(nb\_lettres)$
void file_treatment(char *s);	Cette fonction va récrire un nouveau fichier nommés "traited.txt" qui est un copié collé du fichier s mais avec toutes les modifications décrites appliquées	$O(nbl * nbcar^2)$

4.6.3 Bug

Nous avons découvert un bug à la toute fin sans qu'on puisse le régler. En effet la fonction fileTreatment ne fonctionne pas sur movie$_r eviews.txt$.

4.7 Codage de la mesure de la fiabilité de notre algorithme

4.7.1 Description

L'idée fut de proposer un sous-menu qu'on puisse appeler directement dans un autre menu principal. Ce menu se veut le plus généraliste possible. En effet, il va demander à l'utilisateur un maximum de paramêtres sur les tests effectués pour qu'on puisse faire ce que l'on souhaite. De manière générale, le but va être d'évaluer la fiabilité de notre programme en entraînant l'algorithme sur une partie d'un fichier de données et en évaluant l'eqm sur le reste du jeu de donnée.
Il se présente de la manière suivante :

- On demande à l'utilisateur le path du fichier de données.
- On va lui demander s'il veut appliquer un prétraitement additionnel des données.
- On va lui demander sur combien de lignes il veut évaluer l'algorithme.
- On va lui demander les paramêtres qu'il souhaite donner à la fonction évaluer.
- Puis pour chaque lignes testé, on va afficher la note réelle et la note calculée.
- Enfin on affiche le calcul de l'eqm pour ce test.

4.7.2 codage

Fonctions	Description	Complexité
void testing();	Cette fonction contient un sous-menu clé en main que l'on a juste à appeler dans le menu principale.	variable
double eqm(char* fa, char*fb);	Cette fonction permet de remplir une table de hachage avec les commentaires de fa, puis elle va mesurer l'erreur sur les commentaires de fb. Elle retourne un double qui correspond à l'eqm	$O(n * m^2(i))$
void split(char*s, int nb);	Cette fonction va générer deux nouveaux fichiers : fileA.txt et fileB.txt. Le premier contient les nb premières lignes de s. L'autre contient le reste des lignes.	$O(nb\_lignes)$

(1) On définit n=nombre de lignes dans fa,
et m la taille de la table de hachage.
On suppose que n >>nombre de lignes dans fb.

4.7.3 bug

Nous avons conscience du bug, il nous est apparus à la toute fin et nous n'avons pas put le régler.
Le menu n'est donc pas operationnel.

Chapitre 5

Bilan global du projet

5.1 Bilan de Julien

Ce projet a été l'occasion une fois de plus de travailler en équipe dans des conditions plus proches de ce qu'on peut trouver dans le monde professionnel. Le sujet traité était interessant puisqu'il dérivait d'un procédé très en vogue à notre époque : l'opinion mining. Il m'a de plus permis, à travers la pratique et l'utilisation des connaissances acquises pendant l'année, de bien me familiariser avec le language C mais aussi avec ce qui gravite autour de la réalisation d'un projet en C tel que git, les Makefiles ou encore les tests. Il a été très plaisant de travailler en équipe puisque chacun des membres de l'équipe s'est montré motivé et concerné, et l'entraide a été centrale dans l'avancée du projet.

5.2 Bilan de Mohammed

Mon attribution de ce projet m'a permis de découvrir le déroulement d'un travail en équipe. En effet, d'un côté pédagogique, j'avais la chance de coder parallèlement avec mes collègues des algorithmes de taille plus grande et de nature différentes que ceux connus précédemment en langage *C*. De plus, j'ai pu appliquer la plupart des outils de gestion de projet acquis dans le module du MOOC. D'une autre perspective, j'ai très bien découvert l'importance de la communication et du partage de l'information entre les membres d'une équipe de projet afin que le travail puisse avancer d'une manière équilibrée. Ainsi, j'ai bien aimé le fait que tous les membres de cette équipe soient autonomes en respectant leurs tâches et intervenant directement dans les cas de pannes de leurs coéquipiers.
Je pense que nous étions assez corrects et réguliers dans notre travail, même si nous avons un peu tardé sur des points moins importants spécialement vers le début du projet.

5.3 Bilan de Romain

Ce projet fut une bonne application des différents outils vues en cours, avec par exemple les structures de données. S'adapter à des jeux de données plus vaste fut un réel challenge qui fut très formateur. Il a fallut garder une cohésion au sein du projet tout en cherchant à se rapprocher d'une architecture type logiciel sans réelles connaissances au préalable.
Malgrès un contexte difficile, le projet à avancer et ce en gardant une cohésion de groupe. Même si nous avons eu du mal au lancement, chacun des membres a réussi à réaliser les tâches demandées et à avancer de manière très autonome ce qui était nécessaire malgrès une communication intense dans le groupe.

Malheuresement, ma partie supplémentaire sur fileTreatment et menu testing a été un échec puisqu'un bug est apparu à la toute fin sans qu'on puisse le régler. Et je regrette de ne pas avoir sut anticiper et gérer ce problème.

5.4 Bilan du groupe

Le groupe a eu une bonne cohésion globale durant toute la durée du projet . La communication et l'entraide entre les membres du groupe ont été efficaces dès le début du projet. Nous avons su faire face aux différents problèmes qui se sont posés avec cohérence et pragmatisme. Nous aurions cependant pu être plus efficients dans la mise en route du projet lors de l'implémentation de toutes les structures de données utilisées ainsi que dans la répartition de notre temps du travail pendant la période des partiels. Nous avons souffert d'un bug de dernière minute que nous n'avons pas réussi à régler. Il souligne peut-être un problème d'organisation sur la fin du projet car nous aurions dû anticiper un délais pour régler ce genre de chose.

5.5 Temps de travail de chaque membre groupe

	Julien GIET	**Mohammed KRIMI**	**Romain SCHALLER**
Etat des arts	5h	10h	2h
Implémentation des structures de données	3h	5h	10h
Création des librairies	2h	1h	-
création des tests sur librairies	-	-	5h
fonctions interagissant avec l'utilisateur	-	4h	5h
pretraitement des données	-	-	10h
passage de .txt à table de hachage	5h	15h	5h
Utilisation de la table	6h	-	-
infos sur les tables (stats)	7h	-	-
test des fonctionalités	4h	-	10h
Rédaction du rapport	12h	20h	7h
README	-	1h	-
Temps total de chaque membre	**44h**	**56h**	**54h**

Bibliographie

[1] Audrey BOULLIER, Dominique ; LOHARD. *Opinion mining et Sentiment analysis : Méthodes et outils*. http ://books.openedition.org/oep/214, 2012.

[2] Martin Dørum. A testing library for c, Mars 2018. https ://github.com/mortie/snow.

[3] Mehdi Hadji. *Analyse des sentiments : Généralités*. https ://medium.com/@mehdihadji/analyse-des-sentiments-g, 2019.

[4] TELECOM NANCY. *structure liste et table de hachage*. https ://arche.univ-lorraine.fr/pluginfile.php/1973356/mod$_resource/content/2/Diaporama$, 2020.

[5] Hervé Mazurel et M'hamed Oualdi Quentin Deluermoz, Emmanuel Fureix. *Écrire l'histoire des émotions : de l'objet à la catégorie d'analyse*. https ://journals.openedition.org/rh19/4573, 2013.