Rapport de projet

« L'Analyse des Sentiments au travers du discours »

Introduction

Pour conclure la première année de Master Informatique, nous avions le choix entre la réalisation d'un projet, à choisir parmi une liste de sujets proposés, et un stage.
Notre équipe est composée de Kevin Garabedian et Julian Hurst.

Nous avons choisi plusieurs sujets qui nous semblaient intéressants. Nous avons eu la chance de tomber sur l'un des sujets que nous avions sélectionnés, à savoir : *« Analyse des sentiments à travers le discours ».*

Malheureusement, dû à des soucis d'emplois du temps qui ne concordaient pas, nous n'avons pu obtenir un rendez-vous avec les experts concernés qu'après la fin des cours et des examens. Une fois le premier rendez-vous effectué, nous avons pu démarrer le projet.

Nos encadrants, Mme Line Jakubiec, MM. Vincent Risch et Jean-Jacques Pinto nous ont communiqué leur enthousiasme. La méthode de l'ALS (Analyse des Logiques Subjectives), initiée par M. Jean-Jacques Pinto nous a fait découvrir un aspect intéressant et une application originale de l'informatique.

La première étape a été de nous mettre d'accord sur les standards utilisés lors du projet. Nous avons décidé d'utiliser le langage de programmation Java. Pour le nommage nous avons choisi le standard « Lower Camel Case ». Pour les commentaires, en plus des commentaires à proximité de certaines actions, nous avons choisi de prévoir des commentaires compatibles avec Doxygen afin de pouvoir générer une documentation.

Pour l'arborescence, nous sommes partis d'un fichier Main.java, qui va appeler un fichier qui gère l'affichage, et qui va à son tour utiliser les autres modules de notre application. Pour le moment, seul un fichier Als.java se trouve dans le package affichage. Les classes appartenant au package affichage gèrent l'affichage du menu contextuel, et des résultats du traitement.

Chaque module possède un package qui lui est propre.

Nous avons identifié trois modules :

- le lemmatiseur qui permet de faire la lemmatisation du texte,
- le Tokemiseur qui s'occupe de créer un arbre de tokem afin d'optimiser la recherche de mots pour certaines étapes,
- Synonyme qui permet de trouver les synonymes d'un mot qui n'est pas encore classifié.

Afin de disposer du schéma complet, reportez-vous à l'annexe 1.

Nous avons décidé de nous répartir le travail.

Kevin s'est concentré au départ sur la lemmatisation des verbes , le menu d'affichage et la création du tokemiseur, ainsi que sur la suppression des expressions neutres.
Julian a travaillé sur la recherche et l'utilisation d'une API pour les synonymes, ainsi que sur la lemmatisation des noms propres/communs, adjectifs, et sur la suppression des mots outils et de la ponctuation.

Plusieurs réunions ont été organisées à Luminy pour vérifier l'avancement des travaux et obtenir de nouvelles informations. De nombreux changements ont été apportés dans la stratégie à appliquer et de nouveaux éléments et outils sont apparus au fur et à mesure.
Le projet a donc progressé plutôt de façon itérative selon le feedback donné lors des réunions à Luminy.
Ces réunions ont été l'occasion de soulever des problèmes de traitement dû à l'informatique, et de réfléchir à des solutions pour les contourner.

Description du sujet

Notre sujet était l'Analyse des Sentiments au travers du discours. Ce sujet se base sur la méthode de l'ALS (Analyse des Logiques Subjectives) initiée par M. Jean-Jacques Pinto, psychanalyste.

Cela consiste à identifier des sentiments ou une personnalité au travers du discours d'une personne, au fil d'un processus en plusieurs étapes. Ce processus a été établi par M. Pinto et est en constante évolution. Il vise à identifier si un point de vue est plutôt extraverti ou introverti. Le processus se base sur la manière d'employer certains mots, qui vont permettre de reconnaître le trait de personnalité Ces mots ou groupes de mots (lexèmes), appelés atomes, figurent sur deux listes, appelées Série A et Série B. Les mots complexes, ou molécules quant à eux sont des adjectif complexes, des noms, des verbes, et des adverbes dont le sens peut se décomposer en atome A ou B.

Un atome, est un adjectif qui exprime des propriétés simples (ex : ouvert/fermé) que l'on peut classer selon deux séries opposées :

- La série A concerne l'extérieur, le changement, le désordre, la destruction de l'ancien. Elle se compose d'adjectifs simples comme : ouvert, souple, varié, changeant, nouveau, libre…
-
- La série B concerne au contraire l'intérieur, le non-changement, l'ordre, la conservation. Elle se compose d'adjectifs simples comme : sérieux, ferme, stable, ancien, solide, durable…

Ces listes ne sont pas exhaustives et sont en expansion permanente. Pour schématiser la méthode de M. Pinto, si on trouve une majorité d'atomes de la série A employés d'une façon positive, et/ou une majorité d'atomes de la série B employés d'une façon négative, le point de vue de l'auteur pour ce texte donné sera extraverti.

Et inversement, si on trouve une majorité d'atomes de la série A employés de façon négative, et/ou une majorité d'atomes de la série B employés de façon positive, le point de vue de l'auteur pour ce texte donné sera introverti.

Les différentes étapes nous ont été communiquées, et après étude, ont été modifiées.

Les étapes sont les suivantes

- Suppression de la ponctuation
- Mise en minuscules du texte
- Lemmatisation (normalisation) des verbes (remplacer les formes conjuguées par l'infinitif)
- Suppression des expressions neutres (d'abord, au nom de, à coups d'…)
- Suppression des mots outils (de, du, le, je, tu, mon, car)
  Lemmatisation des noms et adjectifs (remplacer genre et nombre par masculin singulier)

Après toutes ces opérations, le texte est réduit, et on obtient en gros
la liste du vocabulaire pertinent pour l'A.L.S. Le but ultime sera de classifier les mots de cette liste en fonction de leur interprétation.
Cependant il restera des mots à classifier, car comme dit plus haut, les listes des séries A et B ne sont pas exhaustives. Il faut donc les enrichir au fur et à mesure d'une façon aussi peu subjective que possible.

Pour ce faire, il a fallu rajouter une étape supplémentaire :
classifier les synonymes d'un mot afin de déterminer son appartenance à la série A ou B (liste de proxémie)

Une méthode de calcul a été établie par M. Pinto permettant de pondérer les atomes en fonction de leur appartenance. Le résultat du calcul permettra de déterminer si le mot nouveau, pas encore classifié, devra être rajouté à la série A ou B.

On calcule la somme de ses atomes avec +1 pour la série B, et -1 pour la série A. Si le résultat de cette somme est de 20% ou plus du total des mots de la proxémie, alors on peut le classer. Dans le cas contraire, on classe le mot en tant que neutre.

Dans notre cas nous avons travaillé sur des textes, mais il est possible d'étendre ce procédé au discours oral car celui-ci selon l'ALS répond aux mêmes règles.

Cette classification ne permet pas encore de donner un point de vue car les mots ne sont pas valorisés.

Le point de vue

Une personne extravertie utilisera les mots de la série A pour parler de ce qu'il aime, et de la série B pour parler de ce qu'il critique. Au contraire, une personne introvertie choisira des mots de la série B pour parler de ce qu'il aime, et des mots de la série A pour critiquer ce qu'il n'aime pas.

Les parlers

Les parlers est l'extension à l'échelle d'une vie du point de vue, et permet d'en déduire quatre parlers différents.

- Le premier est le parler **« conservateur » (I → I)**, correspondant à la personnalité obsessionnelle « introvertie incorruptible » nostalgique du paradis perdu, qui commence introvertie, et finit introvertie.
- Le deuxième est le parler **« changement/destruction » (E → E)**, correspondant à la personnalité hystérique : « extraverti incorrigible, tenté par l'enfer, qui commence Extravertie, et finit Extravertie.
- Le troisième est le parler **« du progrès » ou « constructeur » (E → I)**, sans équivalent sémiologique : « extravertie repentie » transitant par le purgatoire, qui commence Extravertie, et finit Introvertie.
- Le quatrième est le parler **« hésitant » (I ou E)**, en gros la personnalité phobique : « l'éternel indécis », oscillant toute sa vie entre Extravertie, et Introvertie.

Hypothèse de l'ALS

C'est le discours parental qui détermine, non de façon linéaire mais avec des transformations elles-mêmes « programmées », le discours fantasmatique de l'enfant, différemment selon qu'il est idéalisé ou rejeté (cas extrêmes). L'enfant, une fois identifié au texte du désir parental, qualifiera et traitera désormais tout objet (y compris lui-même et son parent) comme le parent l'a qualifié et a souhaité le traiter. C'est la satisfaction du parent, et non la sienne, qu'il exprime et recherche sans le savoir. Les adjectifs extraits des appréciations du parent sur lui, et les verbes décrivant le sort qu'il lui souhaite, fourniront les atomes valorisés dans les énoncés fantasmatiques, et constitutifs des séries.

Traiter un texte – Exemple :

Concentrons-nous sur le texte de Witold Gombrowicz suivant : *« Ma morale, c'est d'abord de protester au nom de mon humanité personnelle, d'exprimer ma révolte à coup d'ironie et de sarcasmes. Je suis un humoriste, un plaisantin, je suis un acrobate et un provocateur, je suis cirque, poésie, horreur, bagarre, jeu, que voulez-vous de plus ? »*

Pour appliquer l'ALS il faudra respecter les étapes susmentionnées afin de pouvoir analyser les lexèmes. A noter que cet exemple comporte quelques exceptions, dûes aux imperfections du lemmatiseur et aux listes non exhaustives. Pour obtenir un résultat parfait, il aurait fallu analyser scrupuleusement la fonction de chaque mot dans chaque phrase. Cette opération, très complexe, n'a pas été requise et aurait nécessité davantage de temps.

Suppression de la ponctuation et mise en minuscules

ma morale c'est d'abord de protester au nom de mon humanité personnelle d'exprimer ma révolte à coup d'ironie et de sarcasmes je suis un humoriste un plaisantin je suis un acrobate et un provocateur je suis cirque poésie horreur bagarre jeu que voulez-vous de plus

Lemmatisation des verbes

Tout d'abord, il faut remplacer les verbes conjugués par leur infinitif. Cette étape traitera les verbes est, été suis, suis, suis, voulez. Cela donnera le texte suivant :

« ma morale c'être d'abord de protester au nom de mon humanité personnelle d'exprimer ma révolte à coups d'ironie et de sarcasmes je être un humoriste un plaisantin je être un acrobate et un provocateur je être cirque poésie horreur bagarre jeu que se vouloir-vous de plus »

Suppression des locutions neutres

Ensuite, il faut supprimer les locutions neutres à l'aide d'une liste. Dans notre texte, il faudra supprimer d'abord, au nom de, à coup d'. On obtiendra le texte suivant :

« ma morale c'être de protester mon humanité personnelle d'exprimer ma révolte ironie et de sarcasmes je être un humoriste un plaisantin je être un acrobate et un provocateur je être cirque poésie horreur bagarre jeu que se vouloir-vous de plus »

Recherche des expressions figées

Ensuite, il faut chercher les expressions figées afin de les traiter dans leur ensemble, et de les classer dans la série A ou B. Dans ce texte, il n'y en a pas, par conséquent, il reste le même que précédent.

Suppression des mots outils

Maintenant, il faut supprimer tous les mots outils et la ponctuation qui n'apportent rien à l'analyse afin qu'ils ne soient pas traités. On obtient au final le texte suivant :

« moral protester humanité personnel exprimer révolte ironie sarcasme cirque poésie horreur bagarre jeu vouloir plus »

Lemmatisation des noms et adjectifs

Dans cette étape, il faudra mettre sous leur forme lemmatisée les noms et adjectifs afin de pouvoir les analyser.

Cela donnera le texte suivant :

« moral protester humanité personnel exprimer révolte ironie sarcasme cirque poésie horreur bagarre jeu vouloir plus »

Analyse mot à mot

Pour l'analyse de chaque mot, si le mot n'est pas classé dans une série, on utilise la méthode de la classification des synonymes.
Par exemple :
« beau, bon, chaste, convenable, correct, digne, droit, décent, exact, honnête, honorable, int ègre, juste, parfait, probe, propre, pur, respectable, régulier, sage, sérieux, vertueux ». Tous ces mots étant de série B, on a un total de +22, ce qui représente 100%, moral est donc de série B.

Ensuite, il faudra valoriser le résultat. Par exemple, pour moral on se retrouve avec un mot de série B valorisé, donc B+, il impliquera donc l'introversion.

En conclusion, ne fois tous les mots traités un à un, il sera possible de dégager le point de vue général de l'auteur pour ce texte.

Description du travail réalisé.

Le développement

Nous sommes les premiers à travailler sur ce projet. Par conséquent, il n'y avait pas de ressources existantes en termes de développement. Pour ce qui est des autres ressources, nous avions accès à un résumé de ce qu'est l'ALS, rédigé par Monsieur Pinto.

Pour le traitement de texte, nous avions aussi à notre disposition des listes de mots pour les mots outils, les locutions neutres, les expressions figées. Pour finir, afin de lemmatiser les textes nous avions aussi accès à un fichier de lemmatisation mis à disposition par Mme Line Jakubiec.

Le programme comporte actuellement la lemmatisation du texte, et son traitement afin de pouvoir en faire ressortir les mots intéressants à analyser pour l'ALS ainsi que l'analyse des mots grâce à leurs synonymes.

Général

Comme expliqué plus haut, nous avons choisi une arborescence simple sous forme de modules qui sont tous gérés par le Package affichage (voir annexe 1).

Une fois le programme lancé, tout est géré dans « package/Als.java ». Pour ce faire, pour le moment, il y a un menu contextuel (sans interface graphique) fait via une boucle. Pour quitter le programme, il suffit de taper « quit » lors du premier message, à savoir de l'action à effectuer.

Pour le moment, le menu permet de traiter un texte pré-chargé, ou de rentrer le chemin d'accès d'un texte au format txt.

Ce choix par module a été fait car il simplifie les ajouts, et permet au programme d'être évolutif, facilement améliorable et modifiable.

Affichage

Le package affichage s'occupe de l'affichage qui pour le moment n'est que via console, ainsi que de la gestion du processus selon les choix de l'utilisateur.

Lors de son lancement, il s'occupe de charger tous les fichiers texte utiles au traitement afin de ne pas avoir à les recharger à chaque texte. Tant que le programme est lancé, les fichiers textes pour le traitement ne seront donc pas rechargés.

Une fois tous les fichiers nécessaires chargés, il affiche le menu textuel à l'utilisateur. Le premier choix est l'action à faire, soit traiter un texte, soit quitter l'application. Dans le cas où l'utilisateur choisit de poursuivre, il pourra choisir d'entrer le chemin d'accès de son propre texte, ou de choisir un texte dans un choix de 4 textes. Afin de faciliter l'ajout de texte dans cette liste, pour le choix, le programme fait appel à une méthode choix Texte. Les textes doivent être placés dans res/txt/.

Tokemisation

Le package tokemisation s'occupe de stocker les éléments des différentes listes, mais aussi de rechercher un mot de façon optimisée dans ces listes. Il utilise les connaissances acquises lors la matière FIL avec Monsieur Ramisch.

Le programme crée un arbre de tokem dans lequel il ajoute un mot à chaque utilisation de la méthode createVthree (String line, String inf). Bien qu'elle soit à l'origine créée pour traiter les verbes, elle peut être utilisée dans de nombreux cas.

Pour ce qui est de l'arbre, chaque feuille représente une lettre. Lors de l'ajout de la première branche, il suffit de mettre les lettres du mot courant une à une en tant que fils de la lettre qui la précède, et de terminer par une valeur de fin. Dans le cas des verbes, on termine en stockant le verbe à l'infinitif.

Lorsque ce n'est pas le premier mot, tant qu'on a une concordance de lettre, on descend dans l'arbre. Lorsque la lettre diffère, on cherche dans les frères si la lettre courante est stockée. Si c'est le cas, on continue en cherchant les fils ou frère jusqu'à trouver le mot complet, sinon on crée un frère, et on crée ses fils.

Dans le cas où le mot courant serait un mot similaire à un précédent mot, mais plus court (exemple : morale et moral), une fois à la fin du mot, on vérifie de bien avoir la valeur de fin. Si on n'a pas la valeur de fin, on parcourt les frères jusqu'à en trouver un qui n'a pas de frère, et on crée un frère contenant la valeur de fin. (Voir annexe 2 pour l'arbre)

Afin de créer cet arbre, nous avons mis en place une classe qui joue le rôle de structure de feuille. La classe Listchar permet donc de stocker une lettre, un fils, un frère, et une valeur finale de type String. A part cela, elle contient juste des accesseurs, et des mutateurs, ainsi que 2 constructeurs avec paramètre : un pour créer une feuille caractère, et un pour créer une feuille finale.

Pour ce qui est de la recherche, le principe est assez similaire à celui de l'ajout d'un mot. On descend donc dans l'arbre tant que le caractère courant est identique dans l'arbre. Si le caractère est différent, on cherche un frère identique, et dans le cas où l'on trouve un frère, on recommence à descendre de fils en fils comme précédemment.

Si on arrive à la fin du mot courant avant d'atteindre une valeur finale de l'arbre, on vérifie dans les frères s'il n'y a pas une valeur finale.

Dans le cas où on ne trouve plus de frère qui concorde avec la lettre suivante, alors, on renvoie null en signe de négation à la recherche, sinon, on renvoie la valeur finale trouvée.

Ce type d'arbre permet une recherche rapide et efficace, et ce même dans listes de plusieurs centaines de milliers de mots. Le seul inconvénient c'est que pour l'heure, sa création est un peu plus coûteuse que celle d'une liste, mais lors du chargement d'un grand texte, ou de plusieurs textes à la suite, le temps gagné dans la recherche de mots est largement supérieur à la perte lors de la création.

Synonymes

Pour les synonymes, il fallait trouver une base de donnée comportant l'ensemble des synonymes d'un mot donné.

Au début du projet, M. Pinto avait l'intention de récupérer les listes de proxémie depuis le site du CNRTL (Centre National de Ressources Textuelles et Lexicales, créée par le CNRS) Cependant, le site ne proposait pas d'outil développeur pour récupérer ces informations facilement à partir d'une application.
Le site utilise les ressources d'un autre fournisseur, à savoir l'API (APITAL – traitement automatique du langage), sans aucune indication quant aux modalités d'accès. Nous avons signalé l'existence de ce fournisseur aux encadrants. A noter que M. Risch essaie actuellement de joindre l'un des développeurs pour obtenir l'accès à cette API.

Entre temps, nous avons trouvé une autre API fournissant les synonymes des mots, un peu moins précis que les listes de proxémie originales, mais facilement accessibles et disponibles en plusieurs langues. M. Pinto nous a donné son accord pour utiliser cette API en attendant l'accès à APITAL.

Cette API proposée par le site thesaurus.altervista.org fonctionne par requête http GET au site http://thesaurus.altervista.org/thesaurus/v1, avec les paramètres suivants :
word qui est le mot concerné,
language, qui est la langue choisie
Output, qui est soit xml ou json
Key, qui est la clé d'accès à l'API

Il fallait donc pouvoir envoyer des requêtes http GET à partir de java pour un mot donné, et de parser le XML reçu afin d'en extraire la liste de synonymes et de la stocker, afin de pouvoir la classifier.

Pour ce faire, nous avons créé les méthodes suivantes sachant que nous disposons de deux fichiers – sérieA et sérieB- qui contiennent les listes initiales d'atomes de série A et de série B :

- Synonymes.java qui comporte les méthodes findSynonymes (String mot) qui permet de récupérer les synonymes d'un mot si ce mot est un adjectif. On reconnaît que le mot est un adjectif grâce à l'arbre des adjectifs, récupéré à partir de l'objet Lefff.
- Une surcharge de la méthode findSynonymes qui ajoute un paramètre String lang qui permet de choisir la langue.
- La méthode classification qui prend en paramètre deux tokemiseurs, représentant les listes de série A et série B créés à partir des fichiers sus mentionnés, et permet de classifier le mot selon ses synonymes et l'exporter/ajouter à un fichier sérieA_maj.txt s'il appartient à la série A, et sérieB_maj.txt s'il appartient à la série B.
- Plusieurs wrappers : classifierMot, classifierListe, classifierTableau, qui permettent respectivement de trouver les synonymes d'un mot, ou de tous les mots d'une liste ou tableau et de les classifier.
- Plusieurs accesseurs et mutateurs, afin de changer ou récupérer le mot référence, la langue, ou l'arbre d'adjectifs, et de récupérer le tableau de synonymes.
- Cette classe comporte deux constructeurs : le premier permet de créer un objet synonyme avec une langue choisie, et le deuxième permet de créer un objet synonyme en langue française par défaut.

  Grâce à cette classe, nous avons pu étendre les listes de série A et de série B de façon significative, et ce processus pourra continuer dans le temps.

Lemmatisation

Cette classe nous permet de lemmatiser un texte, c'est-à-dire :

- suppression de la ponctuation, des mots outils et des expressions neutres. Cette étape pour la suite de la lemmatisation :
- transposer les verbes conjugués à leur forme infinitive
- supprimer les genres et nombres des noms et adjectifs
- Cette classe nous permet également d'écrire le texte généré dans un fichier et d'ouvrir/lire un texte.

Quatre arbres de tokemisation sont créés grâce à la méthode readLefff() qui va lire dans le fichier de lemmatisation et récupérer les données nécessaires pour chaque arbre : le premier arbre, arbreVerbe contient les verbes conjugués et leur infinitif, le deuxième arbreNomsP contient les noms propres, le troisième arbreNoms contient les noms communs, le quatrième arbreAdj contient les adjectifs.
Compte tenu de la quantité de mots, nous utilisons pour les stocker le tokemiseur afin d'optimiser la recherche. Cette action n'est effectuée qu'une fois au lancement du programme. Une fois le programme lancé, les listes ne sont plus rechargées.

Un cinquième arbre, arbreOutil est chargé grâce à la méthode readOutil et contiendra tous les mots outils.
Pour le moment ils utilisent eux aussi le système de tokemiseur dans le cas où le nombre de mots augmenterait, mais dans l'immédiat, il n'y aurait aucune contradiction à le stocker dans une liste.

Pour finir, on charge la liste des expressions figées. Pour les expressions figées, nous utilisons une liste car le traitement fait dessus est plus simple avec ce genre de structure.

Lorsque l'on veut traiter un texte, on appelle traiteTexte(String p) qui s'occupe d'appliquer tout le traitement afin d'obtenir la liste de mots à analyser. Tout au long du processus, il y a une sortie console afin de prévenir l'utilisateur où en est le programme. Cela permet aussi de vérifier que le programme ne bloque pas lors d'une action.

On commence par ouvrir le texte choisi avec openTexte(String) qui retourne le texte sous la forme d'un String pour faciliter la suite de traitement. La fonction openTexte utilise une ouverture classique du texte qu'elle lit ligne par ligne afin de les stocker dans un String qui est retourné.

Ponctuation :

Une fois le texte récupéré, on commence par supprimer toute la ponctuation afin qu'elle n'interfère pas avec le traitement vu qu'elle n'a pas d'utilité pour l'analyse. Pour ce faire, la fonction supprPonctuation(String) prend en entrée le texte, applique un replaceAll sur toute les ponctuations connues, et retourne le texte modifié.
On remplace aussi les espaces multiples par un seul espace et les retours chariot par rien.

Verbes :

Grâce à la méthode findExp on peut empêcher la lemmatisation de certains mots composés, qui ne doivent pas être considérés ou évalués comme des verbes. Ces mots/expressions sont rassemblés dans un fichier.
Ensuite, comme indiqué dans la méthode de traitement pour l'ALS, il faut lemmatiser tous les verbes, c'est-à-dire les passer à leur forme à l'infinitif. Pour cela, la fonction

traiteVerbe(String) prend en entrée le texte une fois la ponctuation enlevée. Dans un premier temps, le texte est séparé en mot stocké dans un tableau.

Une fois ce tableau créé, on vérifie si le mot contient un ' ou un -. Dans ces deux cas particuliers, on commence par faire un traitement afin de séparer le verbe de la particule qui le compose.

Une fois ce traitement fini, on cherche si le mot est dans l'arbre de verbe, s'il y est, on remplace le verbe par son infinitif dans le texte avec replaceFirst. Le processus reste encore à peaufiner car certains cas ne sont pas traités encore comme ils le devraient, mais la marge d'erreur reste assez faible.

Expressions neutres :

Une fois les verbes traités, on s'occupe des expressions figées neutres. Pour cela la fonction supprExpNeutre(String) applique un replaceAll sur toute les expressions neutres en les remplaçant par la chaine vide afin de les supprimer. Ce processus étant facilité avec une liste, c'est pour cela que les expressions n'ont pas été stockées dans un arbre.

Mot outils :

Ensuite, on supprime les mots outils car ils ne sont plus utiles. Pour le moment, on utilise l'arbre, mais, pour des raisons de simplicité le processus a de grande chances d'être modifié pour un système similaire à celui des expressions neutres.

Nous avons été confrontés à un problème : les mots outils font parfois indirectement partie d'un autre mot, par exemple « le » apparaît dans « indécrottab*le* ».
Nous avons donc adopté la méthode suivante : si le mot est en début de ligne, suivi d'un espace, ou en fin de ligne, précédé d'un espace, ou entre deux espaces, on le remplace par un espace.

Noms propres/noms communs/adjectifs

Ensuite, on traite les noms propres et les noms communs. Pour cela, dès qu'on trouve un mot qui appartient à l'arbre des noms propres, ou des noms communs ou des adjectifs, on le lemmatise en lui donnant sa forme au masculin singulier.

Une fois fini, on écrit le résultat dans un fichier texte afin d'avoir un retour en dur que l'on peut vérifier. A la fin de cette étape, le texte est prêt à être analysé.

Mots potentiellement non identifiés par le lemmatiseur :
Ces mots ne seront pas modifiés, mais pourraient être exportés dans un fichier afin de potentiellement les intégrer au lemmatiseur

Dans tous les cas, les méthodes de modification vont renvoyer le texte modifié.

Perspectives d'avenir

Comme expliqué plus tôt, le projet n'en est qu'à ses débuts d'un point de vue développement, et il reste beaucoup à faire. D'autres améliorations seront faites entre la rédaction de ce rapport et la soutenance.

Pour l'heure, le texte est préparé pour pouvoir être analysé, mais l'analyse n'est pas encore en place. Par conséquent, les prochaines améliorations seront l'analyse, et la valorisation des mots. Nous avons trouvé des pistes intéressantes pour la valorisation des mots, à savoir les travaux de Mme Nuria Gala, plus précisément polarimot, qui est une base de données de mots auxquels a été appliquée une valorisation suivant un système de pondération basé sur la subjectivité.

Une fois ces étapes obligatoires passées, on peut imaginer un grand nombre d'améliorations autour de l'ALS. La première est l'apprentissage automatique ou assisté afin d'augmenter la liste d'expressions figées pouvant être traitées, et de diminuer les erreurs de traitement dues à certaines expressions.

On peut imaginer aussi l'ajout d'une interface graphique permettant de sélectionner un texte de façon plus intuitive, et d'avoir plusieurs retours visuels sous forme de liste, mais aussi de courbes.

Une idée complémentaire consisterait à exporter les données sous une forme qui serait facile à intégrer dans un site web, par exemple xml.

Bien évidemment, par la suite, on peut continuer l'analyse de ces résultats en les couplant a d'autres méthodes pour en ressortir des informations sur l'auteur, et son caractère.

On peut imaginer aussi une possibilité d'adaptations au traitement via l'audio pour un discours parlé ou autre.

Vous l'aurez compris, le sujet est vaste, et propose de réelles possibilités d'évolution pour certaines obligatoires compte-tenu de l'existant que nous allons laisser, mais aussi des améliorations quasiment sans limite si ce n'est l'imagination, et les idées.

Conclusion :

Après un début laborieux, le projet a évolué assez rapidement.

Ce projet s'articulant autour d'une recherche, la méthode de développement s'est faite plus à tâtons en mettant en place des solutions de traitement, et en les améliorant après avoir détecté des erreurs aussi bien dans l'ordre du traitement que dans le traitement lui-même.

Motivés par cette contrainte, nous avons choisi une arborescence sous forme de module ce qui permettait de pouvoir modifier rapidement une méthode dans l'un des modules sans avoir à réadapter tout le programme.

La durée de ce projet nous a semblée trop courte malheureusement compte-tenu de l'étendue de ce qui peut être fait. Cela reste pour nous une expérience toute particulière. Le projet nous donnait vraiment l'impression de mettre les pieds dans un domaine ésotérique, nous demandant de faire preuve de réflexion et d'ingéniosité. Nous avons particulièrement apprécié les discussions lors des rendez-vous, du fait que nous pouvions partager des idées avec le créateur de la théorie.

Ce projet en équipe nous a permis de mettre en pratique des techniques de communication, aussi bien entre étudiants qu'avec les encadrants, sur un projet en constante évolution du point de vue des étudiants et des experts impliqués.
Cette expérience –basée sur la recherche- était totalement nouvelle pour nous.

Ce projet nous a permis d'apercevoir les difficultés inhérentes au langage humain, et combien il est difficile et subtil d'essayer de les appréhender à l'aide d'un langage informatique.
Nous avons découvert des notions propres à l'étude et l'analyse d'une langue, telles que la lemmatisation, les listes de proxémie, et avons compris combien ce domaine est fascinant et source de multiples applications informatiques.

En conclusion, ce projet a été une expérience rare et motivante, car il nous a permis d'évoluer dans un domaine original et très intéressant. Nous n'avons pu cependant qu'effleurer le sujet et nous sommes conscients qu'il reste beaucoup d'étapes supplémentaires à effectuer pour perfectionner le programme.