Utilisation du corpus de l'Internet pour l'aide à la rédaction

Aurélie Zara

Juin 2005

Table des matières

Introduction

Avec la mondialisation et l'explosion des nouvelles technologies de l'information et de la communication, il est devenu indispensable de maîtriser une autre langue que notre langue maternelle. Aujourd'hui on compte plus de 934 millions d'internautes dans le monde dont 25,47 millions de français. 35 % des utilisateurs de l'internet sont des anglophones contre 4% de francophones, et 65% des pages web sont rédigées dans la langue de Shakespeare. Ces chiffres démontrent le caractère indispensable de la maîtrise de l'anglais, tant pour la compréhension des informations disponibles sur le web que pour la rédaction de documents destinés à un lecteur international (pages web, mails, forums, articles scientifiques ect.). Or,dans bien des cas, la rédaction dans une langue étrangère s'avère être une tache laborieuse, même pour ceux qui en maîtrisent la grammaire et l'orthographe. En effet, il n'est pas rare de trouver dans des textes écrits par des apprenants non débutants, des tournures de phrases, correctes d'un point de vue grammatical , mais maladroites d'un point de vue stylistique ou pragmatique. C'est pourquoi, il semble propice de s'intéresser à un outil d'aide à la rédaction capable de mesurer le caractère naturel d'un énoncé. Cet outil devra fournir à son utilisateur une mesure précise, et proposer par exemple différents contextes d'utilisations du fragment concerné. Il pourra aussi être utilisé en tache de fond sur le texte produit et indiquer au rédacteur les énoncés qui, d'après les calculs effectués, sont incorrects.

Le but de ce présent rapport est de décrire les différentes phases du développement d'un outil d'aide à la rédaction destinés aux apprenants de la langue anglaise.

Chapitre 1

Analyse

1.1 définition du caractère naturel d'un énoncé

La langue maternelle est celle que l'enfant acquiert au contact du milieu familial ou il est élevé.L'apprentissage qui se très tôt dans la vie de l'enfant, se fait, de façon intuitive, par un lent processus d'imprégnation, d'imitation et la construction à partir de ce qui a été entendu(la grammaire d'aujourd'hui Michel Arrivé,F Gadet,M Galmiche). C'est cet apprentissage qui permet à un locuteur de reconnaître une production dans sa langue maternelle comme étant naturelle ou non. On dira qu'un énoncé est naturel si il peut être identifié en tant que tel par un locuteur dont la langue maternelle est celle qui a été utilisée pour sa production.

Cette reconnaissance se fait de façon intuitive, et il rare que les critères qui ont permis d'aboutir à cette conclusion, puissent être définis. C'est pourquoi, une simple analyse syntaxique ou sémantique du fragment concerné ne suffit pas. Mais l'hypothèse peut être faite qu'un énoncé est naturel si et seulement si on peut en trouver un certain nombre d'occurrences dans des productions en langue maternelle. C'est pourquoi, l'utilisation d'un corpus comme le corpus de l'Internet semble appropriée.

1.2 utilisation du corpus de l'internet

1.2.1 avantages de l'utilisation d'un tel corpus

Aujourd'hui, plus de 6 milliards de pages web sont référencées par les moteurs de recherche, ce qui fait de l'internet un corpus immense. Outre sa taille, ce corpus possède aussi bien des textes litteraires ,journalistiques ou scientifiques, que des textes proches du langage parlé, comme des discussions provenant des forums. Il rend compte de l'hétérogénéité de la langue, tant pas la diversités des origines sociales et territoriales des rédacteurs (par exemple américain-anglais,parisien-marseillais, chef d'entreprise - employé) que part la diversité des générations (adultes, adolescents ect...),ainsi que de la mutabilité de la langue. La prise en compte de la variation dans le temps des langues est aussi l' un des atouts majeurs du corpus de l'Internet .

1.2.2 un exemple de moteur de recherche : Google

Le meilleur moyen d'accéder aux contenus des pages web est l'utilisation d'un moteur de recherche qui indexe les données contenus sur le web de façon automatique. Il existe actuellement plusieurs moteurs de recherche comme Altavista , Yahoo ect. . Nous parlerons ici de Google. Cet outil, né à la mi-98 s'est imposé, au fil des années, comme le premier moteur au monde. Avec ses 8 milliards de pages - 8 058 044 651 exactement(Source Google : www.google.com), Google détient l'index le plus important devant MSN Search et ses 5 milliards de documents (Source Microsoft : www.microsoft.com), Le moteur de Google effectue une analyse du contenu des sites web repérés sur l'Internet. C'est ainsi que chaque page récupérée est scrutée par le robot à travers des éléments tels que les mots-clés, les occurrences, etc.Mais Google s'appuie aussi sur le PageRank qui apparaît comme un système permettant de mesurer la popularité d'une page sur le web (nombre de fois où la page est citée par un lien depuis un autre site). Plus le PageRank est élevé,

plus la page en question sera considérée comme populaire et par conséquent pertinente par Google.Google propose plusieurs fonctionnalités intéressantes pour lancer des recherches plus précises, comme par exemple la possibilité de délimiter la recherche aux pages contenant : tous les mots demandés, une expression, au moins un des mots demandés. Il est également possible d'exclure des termes grâce à l'option « aucun des mots suivants ». Google permet aussi de limiter la recherche aux pages rédigées dans une langue donnée. Le système d'index de Google nous permet de faire l'hypothèse que les premières pages citées par Google sont écrites en langage naturel du fait qu'elles sont les plus visitées.

1.2.3 Limites de l'utilisation de ce corpus

Divers problèmes se présentent quant à l'utilisation du corpus de l'internet. Le premier est de juger de la qualité des informations disponibles. En effet, il est par exemple difficile de savoir si les pages ont été rédigées en langue maternelle, même si quelques indices peuvent parfois permettre de répondre à cette question (par exemple des sites d'universités américaines ou de journaux). Des erreurs d'ordre lexicale,syntaxique ou stylistique peuvent aussi apparaître dans des textes rédigés par des non apprenants (.On trouve aussi , souvent, en particulier sur les forums, des formes particulières d'écriture basées sur une phonétique approximative, appelé langage SMS(études faites par l'université de Louvain en TAL). Par exemple la phrase "Okay, 1st of all, u hav 2 go 2 my computer" provient d'un forum de jeux videos (http ://forum.onekit.com/) Un autre problème provient des sites commerciaux, qui pour apparaître dans les premiers résultats retournés par les moteurs de recherche, entrent dans leur pages, de grandes listes de fragment de phrases,comme par exemple des noms de produits, ou des mots faisant référence au produits vendus.

1.3 typologie des erreurs commises par un apprenant

1.3.1 erreurs lexicales

Une erreur lexicale est la production d'un mot qui ne fait pas partie du vocabulaire d'une langue. Les erreurs lexicales sont facilement détectables avec un correcteur orthographique ou un étiqueteur par la seule utilisation du dictionnaire. Les moteurs de recherche comme Google, proposent aussi des corrections quand l'énoncé semble faux.

1. Erreurs liées à la forme :
   (a) I would leave scool if I could
   (b) I would like to become teatcher
2. Le pérégrinisme "consiste à utiliser des mots de la langue maternelle en leur appliquant les règles de la morphologie et de la phonologie de la langue étrangère, ce qui aboutit à la création de mots inexistants ou impropres dans le contexte de la langue étrangère ".
   (a) It is a merveillous movie
   (b) A dictatory has taken power
3. La translittération "correspond à une traduction littéraire d'un mot de la langue maternelle résultant en la création d'un barbarisme ".
   (a) i like wrong-fur because animals do not suffe
   (b) I could not support to see my mother wear a fur-coat
   (c) I like clothes who are in the wind

1.3.2 erreurs syntaxiques

Une erreur grammaticale (environ 50% des erreurs) survient lorsque le locuteur ne respecte pas les accords de nombre avec les noms ou encore lorsqu'il fait une erreur au niveau de la morphosyntaxe verbale. voici un exemple d'erreur de temps : "if i will become..." ou encore "Cat are black"

1.3.3 erreurs liées au style

Ce sont les erreurs liées aux répétions , à l'utilisation de la forme passive ou à la production de chaînes trop longues.

1.3.4 autres erreurs

Ce sont les erreurs qui ne peuvent pas être classées dans l'une des catégories précédentes.

1. Chez les apprenants anglais, les erreurs sur de la phonétique quasi similaire sont fréquentes : "there are ships in the field" au lieu de "there are sheeps in the field" est une faute indétectable par un correcteur, car il s'agit d'une erreur de sémantique.
2. On trouve aussi l'utilisation de faux amis : par exemple "it was a good distraction" au lieu de "it was fun" n'ont pas du tout le même sens et le premier fragment est faux.
3. I y a aussi des erreurs de prépositions que les correcteurs ne sont pas capable de détecter :
   (a) segment correct :"i m interested to hear"
   (b) i m interested in hearing
4. Les apprenants font des erreurs sur la position des mots :
   (a) segment correct : "speaks english very well"
   (b) segment incorrect : "speaks very well english"
   (c) segment correct : "long black hair"
   (d) segment incorrect : "black long hair"
5. Mais aussi des traductions littérales français-anglais : l'important est de :
   (a) segment correct :"the important thing is to"
   (b) segment incorrect :"the important is to"
6. Une des erreurs fréquentes faites par des apprenants sont sur verbe + to + action suivant le verbe et verbe + ing + action précédant le verbe, ici il s'agit encore d'une erreur de sémantique qui dépend du contexte :
   (a) segment correct :"i regret saying what i said"
   (b) segment incorrect : "i regret to say what i said"

1.4 outils d'aide à la rédaction disponibles à l'heure actuelle

A l'heure actuelle, il existe un certain nombre d'outils d'aide à la rédaction. Ces outils permettent d'indiquer à son utilisateur des erreurs de type lexical(correcteur orthographique) ou syntaxique (correcteur grammatical). On trouve aussi des thesaurus (dictionnaire de synonymes) ou des concordanciers qui permettent de voir des mots ou des groupes de mots en contexte. Il existe aussi des traducteurs d'une langue vers une autre.

1.4.1 correcteurs orthographiques

Aujourd'hui, il est facile de trouver un correcteur orthographique. Les systèmes d'exploitation et les traitements de texte en sont généralement équipés. La vérification lexicale consiste à s'assurer que chacun des mots d'un texte ou d'une phrase correspond à une forme réellement existante dans la langue cible. Pour y arriver, le logiciel doit comparer chacune des graphies aux chaînes de caractères que contient son lexique interne, c'est-à-dire à toutes les variations possibles des mots en genre et en nombre ainsi qu'à toutes les conjugaisons des verbes. Lorsqu'une graphie ne correspond à aucune forme du lexique interne, le logiciel ne peut poursuivre le traitement syntaxique de la phrase tant que l'erreur n'est pas corrigée. Lorsqu'il rencontre une forme absente de son lexique, le logiciel correcteur fournit habituellement à l'utilisateur une liste de graphies possibles, susceptibles de remplacer la forme erronée. En général, cette liste de graphies possibles sera établie par ressemblance alphabétique, combinée à des éléments de ressemblance phonétique. Par Exemple, en français, en présence de la graphie " chapo ", le logiciel va proposer " chapeau " (par ressemblance phonétique), " chape ", " chapé ", " chapon ", " chap .", " chopa " (par ressemblance alphabétique).

1.4.2 correcteurs grammaticaux

Le logiciel de traitement syntaxique d'une phrase (laquelle est délimitée par une majuscule initiale et un point final, d'interrogation ou d'exclamation) fonctionne lui aussi par comparaison. Il construit le schéma syntaxique de la phrase, en vérifiant la compatibilité des catégories syntaxiques des mots dans l'ordre dans lequel ils sont placés, et en s'assurant que cet ordre correspond à l'un des schémas syntaxiques qu'il connaît parce qu'il a été placé comme référence dans sa base de données . Comme les mots peuvent appartenir à différentes catégories syntaxiques, le correcticiel va s'efforcer de diviser l'énoncé en plusieurs segments en se basant sur la ponctuation, les prépositions ou les subordonnants. Il s'ensuit que toute phrase mal ponctuée ou construite suivant un schéma syntaxique inconnu du correcticiel sera signalée comme étant incorrecte.

1.4.3 correcteurs stylistiques

La correction du style d'un texte consiste à traiter la chaîne de caractères que forme la phrase pour, entre autres,détecter les répétitions, repérer certaines tournures syntaxiques comme la voix passive, signaler des chaînes trop longues ou détecter l'usage de mots "douteux" comme les pléonasmes ou les mots vulgaires.

1.4.4 limites de ces outils

Aujourd'hui ces outils ne permettent pas de corriger des erreurs de sémantique . Par exemple "le cheval rie" est correct du point de vue grammatical mais n'a aucun sens car l'action de rire ne se rapporte qu'à l' être humain. Des concordanciers (donne le contexte gauche et droit d'un énoncé) existent mais ils sont limités par la taille de leur corpus et ne sont donc pas exhaustifs. Quant aux traducteurs automatiques, les textes qu'ils produisent sont souvent peu naturels, et ne peuvent aider qu'à la compréhension de la langue étrangère, et non à sa rédaction . Cependant une traduction de l'énoncé produit par un apprenant, de la langue utilisée vers sa langue maternelle, lui permettrait de vérifier son sens. Par exemple, l'emploi d'un faux ami peut donner une phrase correcte à tout point de vue mais différer du sens que le rédacteur a voulu donner à celle ci.

1.5 méthodes de mesure du caractère naturel d'un énoncé

Nous allons décrire ici deux méthodes statistiques pour la mesure du caractère d'un énoncé.

1.5.1 comparaison avec les moyennes associées aux n-grammes

Supposons, que pour chaque bi-grammes tri-grammes n-grammes ont ait le nombre d'occurrences dans un corpus donné (en langue correcte) alors on peut calculer le nombre moyen d'occurrences pour un bi-grammes tri-grammes ... n-grammes bien formé quelconque. Il suffit alors de comparer chaque nombre d'occurrences associée au n-gramme que l'on veut tester avec la moyenne correspondante . On peut penser que si l'écart entre le nombre d'occurrences du fragment et la moyenne associée est assez petit alors le fragment est naturel. Par contre si l'écart est grand on ne peut rien décider, car le nombre d'occurrences d'un fragment dépend de plusieurs facteurs comme par exemple la fréquence de chaque mots. En effet un mot rare fera baisser le nombre d'apparitions d'un fragment dans un corpus, ce qui ne veut pas dire que ce fragment n'est pas naturel.

1.5.2 chaînes de Markov

En mathématiques, une chaîne de Markov est un processus stochastique possédant la propriété markovienne. Dans un tel processus, la prédiction du futur à partir du présent ne nécessite pas la connaissance du passé. Elles ont pris le nom de leur découvreur, Andrei Markov. Une chaîne de Markov en temps discret est une séquence X1, X2, X3, ... de variables aléatoires. L'ensemble de leurs valeurs possibles est appelé l'espace d'états, la valeur Xn étant l'état du processus au moment n. Si la distribution de probabilité conditionnelle de Xn+1 sur les états passés est une fonction de Xn seul, alors :

$$P(X_{n+1} = x | X_0, X_1, X_2, \ldots, X_n) = P(X_{n+1} = x | X_n) \tag{1.1}$$

où x est un état quelconque du processus. L'identité ci-dessus identifie la probabilité markovienne. Prenons le cas des tri-grammes. On cherche à évaluer la probabilité que le bi-gramme soit suivi du mot W. Pour les débuts de texte on créé des pseudo mots . Pour créer le modèle, c'est à dire (l'ensemble de probabilités) on compte le nombre occurrences du préfixe (ensemble des mots qui apparaissent avant W) ainsi que le nombre d'occurrence du tri-gramme . On obtient la formule :

$$P_e(w_i, w_{i-2,i-1}) = \frac{C(w_{i-2,i})}{C(w_{i-2,i-1})} \tag{1.2}$$

On divise ensuite l'un part l'autre. Le problème majeur avec le modèle du tri-grammes, est dans le cas ou il n'y a pas d'occurrence du tri-grammes dans le corpus. Une solution est de prendre trois coefficients $\lambda_1, \lambda_2, \lambda_3$ tels que leur somme soit égale à 1. Et on applique la formule :

$$P_e(w_n, w_{n-2,n-1}) = \lambda_1 P_e(W_n) + \lambda_2 P_e(W_n|W_{n-1}) + \lambda_3 P_e(W_n|W_{n-1,n-1}) \tag{1.3}$$

On généralise ensuite pour n-grammes. On peut appliquer cette méthode en calculant la probabilité que le dernier mot du fragment testé soit dans la continuation logique de son préfixe, avec la formule si dessus. Si la probabilité est mauvaise, on peut pensé que le fragment n'est pas naturel.

1.5.3 autres algorithmes permettant de maximiser les mesures

Certaines substitutions peuvent permettre d'améliorer les mesures. Par exemple un mot rare peut être remplacé par un synonyme plus fréquent ou encore par un hyperonyme. Pour cela on peut utiliser Wordnet (voir le rapport de Svetlana). Mais on peut aussi faire une substitution par un mot ayant le même lien d'hyperonymie. Par exemple "cow" et "yak" ont le même lien d'hypéronymie. si on substitue "yaks" dans "yaks eat grass" par "cow" on obtient "cows eat grass". Pour le premier, on trouve 4 occurrences, pour le deuxième on en trouve plus de 3000 sur google. Si le fragment testé contient un nom propre alors on peut le remplacer par un pronom personnel : par exemple pour "John is eating" on trouve 428 occurrences . Si on substitue "John" par "he" on obtient 46000 résultats. Si l énoncé contient un chiffre comme "first" on peut faire les différentes substitutions avec les autres chiffres et additionner les résultats, ou prendre le maximum. On peut utiliser la même approche pour les déterminants posséssifs (my, your , her , his , their). Le problème est de s'assurer qu'il y a alors bien équivalence entre les deux fragments.

Chapitre 2

Spécification et implémentation de l'outil d'aide à la rédaction

2.1 Spécification informelle

Voici les spécifications qui ont été faites au début du stage :

1. dans un premier temps l'utilisateur devra pouvoir tester le caractère naturel d'un énoncé donné en entrée
2. dans un deuxième temps cette vérification pourra être exécutée en tache de fond à la demande de l'utilisateur et lui indiquer quels fragments semblent faux à la manière des correcteurs orthographiques et grammaticaux actuels.
3. l'utilisateur pourra aussi voir différents exemples de contextes dans lesquels apparaît le fragment qu'il veut tester.
4. le contexte gauche de l'énoncé pourra dans un second temps être pris en compte pour l'évaluation du fragment
5. Le système pourra proposer à l'utilisateur différentes reformulations de l'énoncé si celui ci n'est pas correct.
6. Le système pourra proposer une traduction dans la langue du rédacteur afin que celui ci puisse vérifier le sens du fragment de texte qu'il veut tester.
7. Le système devra utiliser différents moteurs de recherche afin d'utiliser au mieux chacune de leurs capacités respectives (par exemple utiliser Google ou Yahoo).
8. Le système devra effectuer différentes mesures sur les fragments dont un doute subsiste dans le but d'affirmer ou d'infirmer les résultats donnés par une première mesure.
9. Ces différentes mesures de l'énoncé devront être associées à des mesures faites sur l'énoncé transformé par exemple par une substitution de mots (voir le rapport de Svetlana).

2.2 schéma général du système

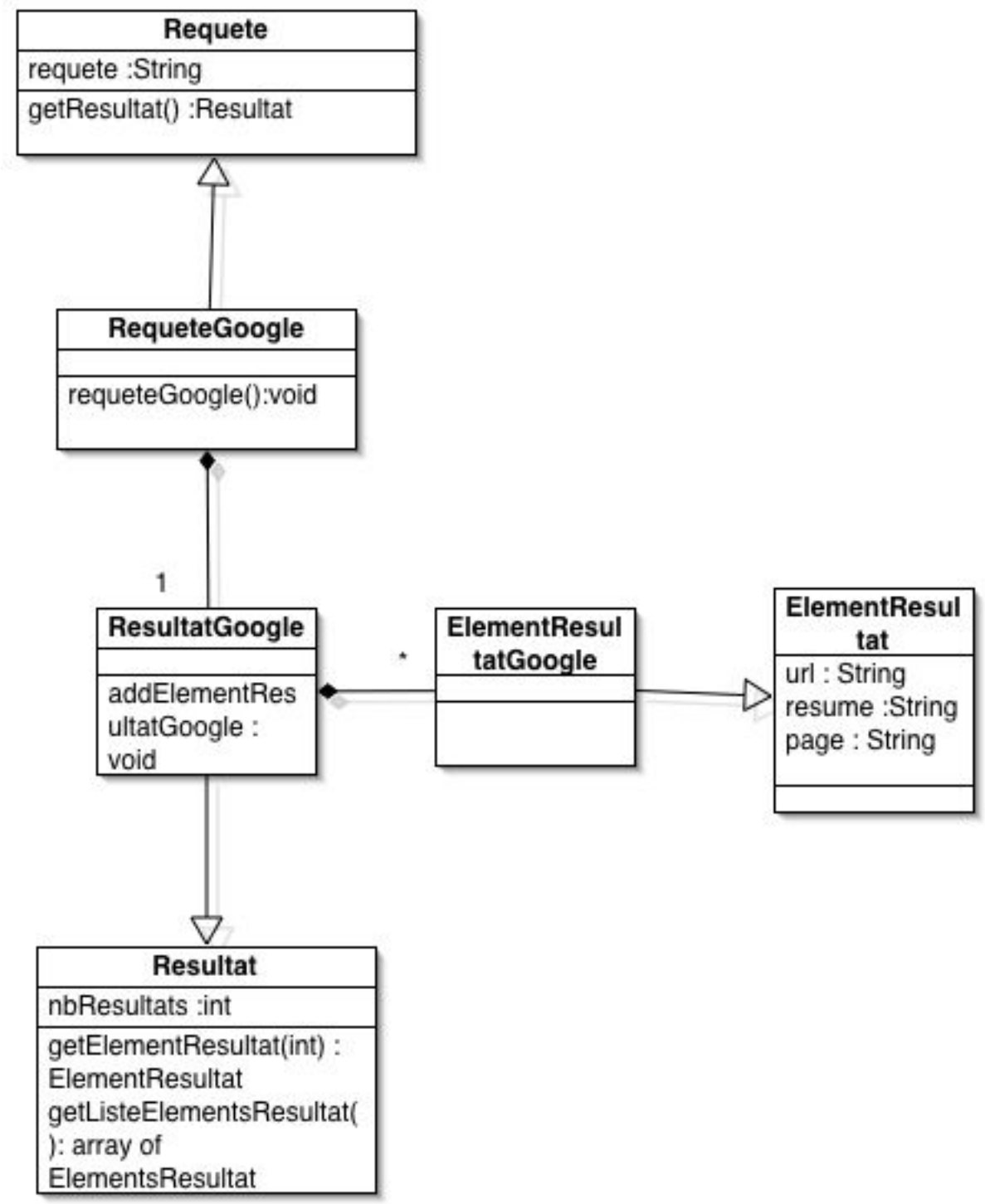
Le langage choisi pour l'implémentation de l'outil est Java. Nous avons choisi ce langage non seulement parce qu'il est multiplateformes, mais aussi parce qu'il existe de nombreux packages utiles au traitement des langues comme par exemple les expressions régulières, les tokenizers , les outils pour le XML ect.... Le projet contient 17 classes. On peut regroupé les différentes classes en quatre grands groupes :

1. classes d'interface avec un moteur de recherche
2. classes de traitement des données résultats
3. classes liées au score
4. classes implémentant l'interface graphique Le diagramme de classe est donné en annexe.

2.2.1 les classes d'interface avec un moteur de recherche

Elle sont au nombre de 6 dont 3 abstraites Cette structure suit la structure de l'API Google mais aussi celle de l'API Yahoo. Requête,Résultat, ElementResultat sont des classes abstraites . Les 3 autres classes (liées à l'API Google),étendent ces trois classes. En cas d' utilisation d'un autre moteur de recherche, on pourra utiliser les résultats provenant des différents moteur de recherche de façon confondue, sauf dans certain cas comme le calcul du score par exemple.

1. Requête : est associé à un résultat. Elle implémente des méthodes d'accès à la requête et au résultat
2. Résultat : un résultat est lié à un ensemble d'éléments résultat. Des méthodes d'accès à l'élément sont implémentées dans cette classe.
3. ElementResultat : possède des méthodes d'accès à l'URL des pages résultats, ainsi qu'à leur résumé et aux différents contextes extraits.
4. RequeteGoogle : implémente des méthodes permettant de faire des requêtes sur google
5. ResultatGoogle : donne le nombre de résultats pour une requête et possède une liste des éléments résultats
6. ElementsResultatGoogle : donne le résumé ainsi que l'url de la page résultat, il nous donne aussi le contexte.



2.2.2 les classes de traitement des données résultats

1. classe OutilsTexte : fournit diverses outils pour le traitement du texte, comme par exemple une méthode permettant de récupérer les contextes à partir de l'énoncé..
2. classe fichierHTML : permet de récupérer les différents contextes en parallèle
3. classe InterfaceTagger : tagge les mots de la requête
4. classe fichier XML : fournit des méthodes pour créer un fichier XML à partir des résultats d'une requête

2.2.3 les classes liées au score

Elles sont au nombre de deux :

1. ModelGoogle : cette classe permet de faire l'apprentissage des moyennes pour chaque n-grammes (n allant de 2 à 8)
2. BasicScoreGoogle : cette classe, en fonction des résultats fournit par le modèle, donne un score au fragment à tester.

2.2.4 les classes de l'interface graphique

Elles sont au nombre de deux. L'une est l'interface graphique à proprement dit, l'autre est l'interface entre l'interface graphique et le système permettant de faire des requêtes et de donner un score . On a donc :

1. InterfaceGraphique
2. InterfaceRequete

2.3 requêtes sur un moteur de recherche (classes RequeteGoogle, Resultatgoogle et elementResultatGoogle)

Pour les requêtes, j'ai choisi d'utiliser l'API Google, Il s'agit d'un kit de développement logiciel disponible librement, qui permet de créer de nouvelles applications utilisant directement la base de donnée des pages indexées par Google, par le biais d'un service web. Afin d'utiliser l'API Google, il faut se procurer une clef d'identification (une suite de chiffres et de lettres) qui lui est propre. Pour ne pas nuite au bon fonctionnement de Google, chaque clef est limitée à 1000 interrogations serveur par jour. L'API est basée sur le protocole SOAP et le langage WDSL. SOAP (Simple Object Access Protocol) est un protocole d'échange de données et de services entre applicatifs et sites Web. Il s'appuie sur le protocole HTTP pour le transport et sur le langage XML pour la description des données et des messages transmis. Le langage WSDL (Web Services Description Language) sert quant à lui à décrire la façon d'appeler et d'utiliser les services Web distants.

L'API se compose de cinq classes :

1. GoogleSearch
2. GoogleSearchDirectory
3. GoogleSearchFault
4. GoogleSearchResult
5. GoogleSearchResultElement

Pour faire une requête et récupérer le nombre d'occurrences d'un fragment de texte, on utilise essentiellement deux classes : GoogleSearch et GoogleSearchResult. La première permet de faire une requête dans la base de données de Google .Cette classe permet de définir le nombre de résultats qui doivent être retournés (0 à 10 résultats), mais l'utilisateur peut aussi définir des restrictions sur la langue, ce qui dans notre cas sera l'anglais. La deuxième classe permet d'avoir accès à la listes des éléments résultats ainsi qu' au nombre total de résultats. Nous nous intéresserons à la classe GoogleSearchResultElement dans un deuxième temps, pour récupérer les contextes liés au fragment testé.

Voici un exemple de code Java pour faire une requête sur Google :

```
public void requeteGoogle(String langueRestrict,int maxResult, String req){
    GoogleSearchResult resultatGoogle;
 //initialisation du nombre maximum de résultats retournés
     search.setMaxResults(maxResult);
//initialisation de la chaine de caractère représentant la requête
     search.setQueryString("\""+req+"\"");
     try {
//lancement de la recherche
        resultatGoogle=search.doSearch();
        resultat = new ResultatGoogle(maxResult);
```

```
  //initialisation du nombre de résultats obtenus
        resultat.setNbResultat(resultatGoogle.getEstimatedTotalResultsCount());
   //initialisation des éléments résultats retournés par Google
        GoogleSearchResultElement[] elements = resultatGoogle.getResultElements();
        for (int i=0;i<elements.length;i++){
            resultat.addElementResultatGoogle(elements[i],i);
        }
                    } catch (GoogleSearchFault e) {
                     e.printStackTrace();
                }
                //on garde une trace de la requête
     this.requete=req;
//nombre de mots dans la requête
     this.nbMots=OutilsTexte.countWord(req);
    }
```

C'est cette méthode qui sera au centre de tout le projet. Nous l'utiliserons dans divers contextes.

2.4 récupérer les contextes

2.4.1 récuperer le code sources des pages résultats (class PageHTML)

Pour pouvoir proposer à l'utilisateur de cet outil divers contextes d'utilisation de l'énoncé dont il veut tester le caractère naturel, nous utilisons les éléments résultats de l'API google. La classe *GoogleSearchResult* nous fournit l'adresse ainsi que le résumé de la page résultat. L'adresse nous permet de récupérer le code html de la page grâce à la classe URL. Cette classe possède une méthode *openStream()* qui ouvre une connexion avec le serveur correspondant à l'url et retourne le code HTML dans un InputStream.

Pour pouvoir récupérer rapidement toutes les pages HTML données en résultats et les traiter, les connexions avec la méthode *openStream()* ainsi que le traitement des pages se font en parallèle. Chaque connexion est un Thread et tous les Threads sont lancés simultanément. Le programme principal est alors bloqué jusqu'à ce que tous les Threads aient finis leur traitement. Ceci implique que le temps d'exécution dépend du traitement le plus long (connexion la plus lente, ou traitement du code html le plus long) et non de la somme des temps de traitement. Voici les deux méthodes qui permettent l'exécution en parallèle :

```
//c'est le Thread
public  void run() {
    statut=en_cours;
    try {
    //on recupere le code html de la page
       codeHTML=getHTMLcode();
       //on enlève les balises HTML pour ne récupérer que le texte
       codeHTML=OutilsTexte.getTexteFromHtml(codeHTML);
       //on extrait tous les contextes du fragment  contenu dans la page
       eR.contexte=OutilsTexte.getContext(req,codeHTML);
    } catch (IOException e) {
     e.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    } catch (InitOutilsTexteException e) {
e.printStackTrace();
}
    statut=finis;
    //il s'agit d'un compteur statique initialisé par le nombre de thread et qui est décrémenté
    //à la fin du traitement
    compteur--;
```

```
   }

   //méthode qui initialise tous les threads
    public static void getAllContexts(Resultat r,String reqRegex){
 ElementResultat[] elemResult=r.getListeElementsResultat();
compteur=elemResult.length;
//pour chaque éléments résultat on lance un thread
for (int i=0;i<elemResult.length;i++){
PageHTML p=new PageHTML(elemResult[i],reqRegex);
p.start();
}
//tant que le compteur n'est pas nulle alors le programme principal est bloqué
while(compteur!=0);
}
```

2.4.2 traiter le code source des éléments résultats (class OutilsTexte)

Dans un premier temps nous devons supprimer les balises HTML qui nous empêchent de récupérer les contextes facilement. En effet on trouve dans les balises du code avec des symboles qui peuvent être confondu avec des fins de phrases. Mais on veut aussi pouvoir fournir à l'utilisateur des phrases "saines", c'est à dire sans balises ou morceaux de code . Pour cela on utilise des expressions régulières à l'aide de l'API Jregex (http ://jregex.sourceforge.net/) . Pour supprimer les tags dans le code HTML on utilise l'expression régulière suivante :

```
 (<[^>]*>)|(\\[[^\\]]*\\])
```

On parcourt le code jusqu'au caractère de fin et à chaque fois que l'expression régulière est "matchée" on supprime le texte correspondant. Le code HTML introduit des codes de symbole que java n'est pas capable d'interpréter, comme par exemple

```
&#171  ou encore &Oslash.
```

Il faut donc les remplacer par les symboles correspondant. Pour cela nous avons un fichier contenant tous les symboles et leur codes. A l'initialisation de l'outil, on lit ce fichier et chaque code est rangé dans une table de hachage et la clé pour ce symbole est le code correspondant. On utilise une expression régulière pour repérer le code de ces symboles :

```
 (&#(\\d)*;)|&(\\w)*;
```

et à chaque fois que l'on trouve un fragment de texte correspondant à l'expression régulière on fait une recherche dans la table de hachage et on remplace par le symbole correspondant. Maintenant, nous voulons récupérer les phrases dans lesquelles est contenu l'énoncé que l'on veut testé. Pour cela nous avons besoin de transformer la requête en expression régulière. En effet Google omet les marques de ponctuation contenues dans les requêtes, et ne tient pas compte du nombre de blancs entre les mots. Donc les fragments contenus dans les pages ne sont pas forcément exactement les même que l'énoncé initial. Il nous faut donc d'abord découper le fragment en mots puis introduire entre chaque mot une expression régulière. Pour cela nous utilisons les méthodes de la classe StringTokenizer. Voila le code de la méthode qui transforme une requête en expression régulière :

```
\\prend en entree la requête et l'expression régulière à insérer entre chaque mot
public static String transRequeteRegex(String requete,String regex){
StringBuffer sb = new StringBuffer();
StringTokenizer st = new StringTokenizer(requete);
\\on découpe la requête en tokens
while (st.hasMoreTokens()){
\\on ajoute derriere le token l'expression régulière
```

```
sb.append(st.nextToken()+regex);
}
return sb.toString();
}
```

L'avantage de prendre en entrée l'expression régulière à placer entre chaque mot, est que l'on peut introduire la possibilité de faire une recherche où il peut y avoir des termes qui apparaissent entre les différents mots. Par exemple, si on veut faire une requête "the * cat is sleeping" , on pourra aisément récupérer le fragment avec le mot "matché" par l'étoile dans la requête, par exemple "the black cat is sleeping". La transformation de la requête en expression régulière nous permet maintenant de récupérer la position du fragment dans la page résultat. Il nous faut alors récupérer le contexte gauche et le contexte droit de l'énoncé. Nous voulons nous arrêter à la fin de la phrase précédente, et au début de la phrase suivante, si le fragment n'est pas en fin de phrase, sinon une phrase plus loin.

Pour cela nous définissons trois méthodes :

1. getStartIndexOfSentence
2. getEndIndexOfSentence
3. analyseVoisinnage

La première nous permet de récupérer l'indice du début du contexte à l'aide d'analyseVoisinnage et la deuxième méthode nous permet de récuperer l'indice de fin du contexte toujours à l'aide de la méthode AnalyseVoisinnage. AnalyseVoisinnage observe une suite de symboles sans blanc. Si cette suite contient un "." ou un " ?" ou un " !" alors si c'est un " !" ou " ?" alors on considère qu'il s'agit d'un symbole de fin de phrase, sinon si elle contient un "." alors

1. Si la suite de caractère correspond à une abréviation alors ce n'est pas une fin de phrase (on a préalablement chargé une liste d'abréviations que l'on a rangé dans une table de hachage.). Sinon :
2. Si la suite de caractère est un réel, c'est à dire matchée par l'expression régulière :

   `(([A-Z0-9]\\.)+)|([a-z0-9]\\.)+`

   alors il ne s'agit pas d'une fin de phrase. Sinon :
3. Si le point est positionné à la fin de la suite de caractère, alors il s'agit d'une phrase. Sinon :
4. Si la partie droite du point contient plusieurs points alors on regarde le suffixe (.*.suffixe). Si le suffixe est un point alors c'est une fin de phrase. Sinon, si il s'agit d'un suffixe connu (com, net, edu...) alors il s'agit d'une adresse mail ou d'une adresse internet et ce n'est pas une fin de phrase. Sinon :
5. On considère qu'il s'agit d'une ponctuation de fin de phrase.

Voici l'implémentation des trois méthodes :

```
public static int getStartIndexOfSentence(int indice,String texte){
int ind=indice;
//on se de déplace dans la chaîne de caractère de droite à gauche
while(ind>0){
char current=texte.charAt(ind);
//si le caractère courant est un ! ou ? ou un . alors
if (current=='!' || current=='?' || current=='.'){
//on récupère la chaîne de caractère sans blanc qui contient ! . ou ?
String candidat=texte.substring(getStartIndex(ind,texte),getEndIndex(ind,texte));
//si le candidat contient une fin de phrase alors on retourne l'indice
if( OutilsTexte.analyseVoisinnageBis(candidat))
return ind+1;
}
//sinon on continue
ind--;
}
return ind;
}
```

```
public static int getEndIndexOfSentence(int indice,String texte){
int ind=indice;
//on se de déplace dans la chaîne de caractère de droite à gauche
while(ind<texte.length()){
char current=texte.charAt(ind);
//si le caractère courant est un ! ou ? ou un . alors
if (current=='!' || current=='?' || current=='.'){
//on récupère la chaîne de caractère sans blanc qui contient ! . ou ?
String candidat=texte.substring(getStartIndex(ind,texte),getEndIndex(ind,texte));
//si le candidat contient une fin de phrase alors on retourne l'indice
if( OutilsTexte.analyseVoisinnageBis(candidat))
return ind+1;
}
//sinon on continue
ind++;
}
return ind;
}


private static boolean analyseVoisinnageBis(String candidat){
    Pattern p=new Pattern("[.!?]");
    Matcher m=p.matcher(candidat);
    String prefixe=null;
    String suffixe=null;
    StringBuffer sb=new StringBuffer(candidat);
    int index;
    //si il n'y a pas de .!? alors ce n'est pas une fin de phrase
    if (!m.find()){
        return false;
    }
    //si il appartient à la liste des abréviations alors ce n'est pas une fin de phrase
    if (abbrev.get(candidat)!=null){
   System.out.println(candidat+" : est une abbreviation");
         return false;
    }
    //si c'est un réel alors ce n'est pas une fin de phrase
    if (candidat.matches("(([A-Z0-9]\\.)+)|([a-z0-9]\\.)+")) {
     return false;
    }
    index=m.start();
    //si le dernier caractère est un point alors c'est une fin de phrase
    if (index==(candidat.length()-1)){
        return true;
    }
    if (index>0)
        prefixe=candidat.substring(0,index);
    else
        prefixe="";
    suffixe=candidat.substring(index+1,candidat.length());
    String ls=OutilsTexte.getLastSuffix(suffixe);
    //si le suffixe est un point alors c'est une fin de phrase
```

```
    if (ls=="."){
        return true;
    }
    //si le suffixe est dans la liste des suffixes (com net edu...) alors ce n'est pas une fin de phrase
    if (suffix.get(ls)!=null){
        return false;
    }
    //sinon on considère que c'est une fin de phrase
     return true;
}
```

Nous obtenons donc les contextes d'un énoncé de cette façon.

2.4.3 construction du fichier XML associé au fragment testé (classes fichierXML InterfaceTagger)

pour garder une trace des résultats obtenu pour un fragment, on construit un fichier XML contenant diverses informations :

1. l'énoncé lui même
2. les mots taggés avec leur catégories grammaticales
3. les différents contextes liés au fragment

Pour construire le fichier XML, on utilise l'API JDOM.

DOM est l'acronyme de Document Object Model. C'est une spécification du W3C pour proposer une API qui permet de modéliser, de parcourir et de manipuler un document XML. Le principal rôle de DOM est de fournir une représentation mémoire d'un document XML sous la forme d'un arbre d'objets et d'en permettre la manipulation (parcours, recherche et mise à jour). A partir de cette représentation (le modèle), DOM propose de parcourir le document mais aussi de pouvoir le modifier. Ce dernier aspect est l'un des aspect les plus intéressant de DOM. DOM est défini pour être indépendant du langage dans lequel il sera implémenté. DOM n'est qu'une spécification qui, pour être utilisée, doit être implémentée par un éditeur tiers. DOM n'est donc pas spécifique à Java. Le parseur DOM pour JAVA le plus répandu est Xerces. JDOM utilise DOM pour manipuler les éléments d'un Document Object Model spécifique (créé grâce à un constructeur basé sur SAX). JDOM permet donc de construire des documents, de naviguer dans leur structure, d'ajouter, de modifier, ou de supprimer leur contenu.

La création d'un fichier XML en partant de zéro est des plus simple. Il suffit de construire chaque élément puis de les ajouter les uns aux autres de façon logique. Un noeud est une instance de org.jdom.Element. voici l'implémentation de la construction du fichier XML grâce à JDOM :

```
public static void generateXMLfile(String requete,Resultat resultat)
throws FileNotFoundException, IOException{
//l'élément racine est la requête
Element racine = new Element("requete");
//on lui ajoute comme attribut l'énoncé en chaîne de caractère
Attribute enonce=new Attribute("enonce",requete);
racine.setAttribute(enonce);
org.jdom.Document document = new Document(racine);
//le premier fils de la racine sont les tokens
Element tokens = new Element("tokens");
racine.addContent(tokens);
//on tags les differents mots de la requête
ArrayList toks=OutilsTexte.segmenter(requete);
String[] tags=null;
try {
tags = InterfaceTagger.tag(toks);
```

```
} catch (InitTaggerException e) {
e.printStackTrace();
}
// on les ajoute au noeud tokens
for (int i=0;i<toks.size();i++){
Element token =new Element("token");
//on ajoute les tags en attribut de chaque mot
Attribute tag=new Attribute("tag",tags[i]);
token.setAttribute(tag);
token.setText((String)toks.get(i));
tokens.addContent(token);
}
//on ajoute à la racine le noeud contextes
Element contextes=new Element("contextes");
racine.addContent(contextes);
ElementResultat[] elems=resultat.getListeElementsResultat();
//on ajoute au noeud contextes tous les contextes du fragment concerné
for (int i=0;i<elems.length;i++){
for(int j=0;j<elems[i].contexte.size();j++){
Element contexte= new Element("contexte");
Attribute numElem=new Attribute("numELem",String.valueOf(i));
contexte.setAttribute(numElem);
contexte.setText((String)elems[i].contexte.get(j));
contextes.addContent(contexte);
}
}
//on crée le fichier de sortie
 XMLOutputter sortie = new XMLOutputter(Format.getPrettyFormat());
 sortie.output(document, new FileOutputStream("requete.xml"));
}
```

Pour étiqueter les mots avec leur catégories grammaticales on utilise Qtag .Qtag donne les informations sur les parties du discours de façon probabiliste. Voici la méthode qui permet d'étiqueter un texte dont les mots sont rangés dans une liste.

```
public static String[] tag(ArrayList enonce) throws InitTaggerException{
String[] s=null;
if (!init)
//erreur si le tagger n'a pas été initialisé
 throw new InitTaggerException("il faut d'abord lancer la methode initTagger de la classe InterfaceTagger
else {
//tags les mots de l'énoncé
s=tagger.tag(enonce);
}
return s;
}
```

2.5 implémentation des mesures (classe ModelGoogle et BasicScoreGoogle)

Je n'ai eu le temps d'implémenter qu'une seule mesure, la plus simple. Cette mesure utilise les requêtes sur Google pour obtenir les nombres moyens d'occurrences sur les bi-grammes, tri-grammes ect.

2.5.1 apprentissage du nombre moyen d'occurrences pour chaque n-grammes

Pour chaque n (n allant de 2 à 8) , on crée un thread qui va parcourir de n en n le fichier source. Pour chaque n-grammes on lance une requête Google, et le couple (n, nombre de résultats) est rangé à la fin d'un fichier texte. Pour éviter d'avoir des résultats trop élevés lorsque tous les mots sont vides, on se sert d'une méthode qui utilise des listes de mots vides et qui renvoie vrai si tout les mots appartiennent à cette liste. Comme l'API de Google nous limite à 1000 résultats par jour, nous avons un compteur partagé par les threads, qui est décrémenté à chaque requête. On l'initialise à 900 pour avoir encore faire la possibilité de faire des requêtes si besoin.

```
public static void constructModel(int n_grammes,StringBuffer sbIn,StringBuffer sbOut)
 throws InitOutilsTexteException{
//expression régulière qui permet de matcher n mots
Pattern p=new Pattern("(\\w+\\W+){"+n_grammes+"}");
Matcher m=p.matcher(sbIn.toString());
String tmp=null;
try{
//tant qu'on a pas atteint la fin du texte et qu'on à pas atteint les 900 requêtes
while (m.find() && compteur_req<900) {
tmp=m.group(0);
//si les mots de la requête ne sont pas vides et ne contient pas de guillemets alors
if (!OutilsTexte.contientQueMotsVides(tmp) && (!OutilsTexte.contientDesGuillements(tmp))) {
RequeteGoogle rg=new RequeteGoogle("lan_en",1,tmp);
//on lance la requête
rg.requeteGoogle();
synchronized (sbOut) {
//on écrit dans un stringBuffer le résultat
  sbOut.append(n_grammes+" ");
sbOut.append(rg.resultat.nbResultats+"\n");
compteur_req++;
}
System.out.println(compteur_req+" "+n_grammes+" "+rg.resultat.nbResultats+" "+tmp);
}
}
} catch (Exception e){
System.out.println("problemes google");
}
}
```

2.5.2 utilisation des moyennes pour la mesure du caractère naturel d'un énoncé

Pour calculer les moyennes, on lit le fichier d'échantillons, pour chaque n (de 2 à 8) on range à l'indice n le résultat associé puis on fait la moyenne pour chaque n-grammes que l'on range ensuite dans un tableau des moyennes.

Ensuite pour chaque requête de l'utilisateur, on calcule ((nb-resultats*100)/[moyenne des n-grammes], n étant le nombre de mots dans l'énoncé donné en entrée.

2.6 interface graphique (classe interfaceGraphique

L'interface graphique a été implémentée avec le package swing appartenant à la librairie standart de java. Elle est composée d'un champ texte qui permet à l'utilisateur d'entrer l'énoncé dont il veut tester le caractère naturel, d'un bouton "ok" pour lancer la requête, de boutons de choix entre Google et Yahoo, d'un champs de texte permettant de choisir le nombre d'éléments résultats que l'on veut (1 à 10) , et enfin une zone de texte, où s'affiche pour une requête le nombre de résultats, les différents contextes, le résumé de la page fournir pas Google et à la fin la mesure, en pourcentage, effectuée sur ce fragment.

Voici une image de l'interface graphique permettant à un utilisateur de tester le caractère naturel d'un énoncé

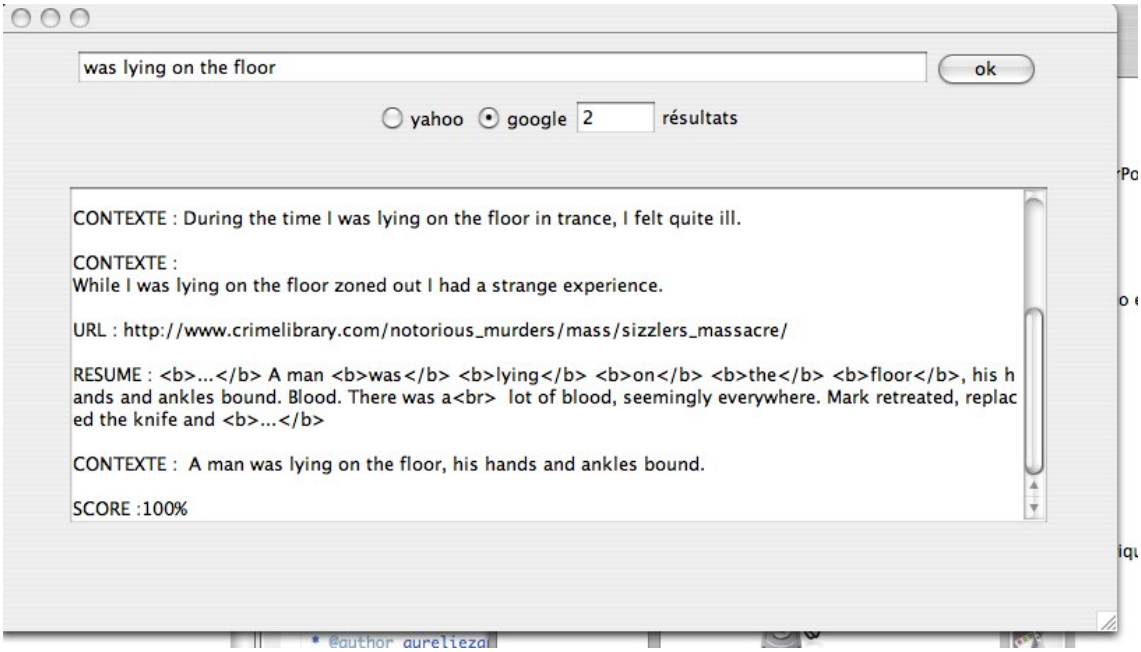


FIG. 2.1 – interface graphique

Chapitre 3

Jeux de tests et rapport de tests

3.1 résultats sur l'utilisation de l'API google

On constate que l'API Google ne renvoie pas les mêmes résultats que le moteur de recherche (www.google.fr). Aucune explication n'a été donné par Google quand à ce phénomène. Ceci montre la nécessité de donner des mesures en fonction des moteurs de recherche. Voici quelques exemples :

requête	résultats API	résultats site Google
was lying on the floor	7090	37500
how do you do	129000	695000
was entirely hidden	372	607

3.2 résultats du traitement des données fournies par Google

3.2.1 tests de la génération de contextes

La plupart des contextes issus des sites autres que les forums et les sites commerciaux, sont corrects. Voici des exemples de contexte utilisable : Pour l'énoncé "how do you do" :

```
" They then shake hands saying, "How do you do, how do you do, how do you do?"
Each then runs in opposite directions around the circle saying
"How do you do's" when they meet.
```

Pour l'énoncé "i apologise for all the stuff" :

```
 I apologise for all the stuff I actually remember doing,
 and trust all photographic evidence of anything
  I've forgotten will be destroyed immediately!
```

Ou encore pour le fragment "made up is mind" :

```
It seemed clear that Bush had made up his mind to take military action,
 even if the timing was not yet decided.
```

Voici des contextes inutilisables : pour l'énoncé "how do you do" :

```
 Fine How-Do-You-Do Link Text ++++++++++++++++++++++++++++++++++++++++++++++++++ -->del.
```

Certains contextes issus de forums, donnent des informations sur la date l'auteur ect...ce qui n'est pas souhaitable :

```
DanCoolidge2456 Posts Posted-05/24/2005: 12:57:12   quote:Originally posted by dprorok
Though it was entirely hidden under my orange velvet-lined hood,
 I proudly wore the purple Carlo Franco tie my mom bought me as a graduation
 gift (S23 if you're wondering).
```

ou encore des morceaux de code comme pour la requête "i apologize for all the stuff" :

```
 write('');//-->
= 11) OAS_RICH(pos); else OAS_NORMAL(pos);
}
//-->OAS_AD('Top');GANTLEY-L ArchivesArchiver > GANTLEY > 2000-08 > 0965568381
From: "Gantly Family" DisplayMail('','');
Subject: Re: More Gantleys
Date: Sun, 6 Aug 2000 14:26:21 +0100
References: 20000806082127.22602.qmail@web3204.mail.yahoo.com
Dear tigress-your name will be taken off our list immediately and
I apologise for all the stuff
you have received.
```

3.2.2 tests de la génération du fichier XML

Voici le fichier XML pour la requête "was lying on the floor" pour 2 résultats.

```
<?xml version="1.0" encoding="UTF-8"?>
<requete enonce="was lying on the floor">
  <tokens>
    <token tag="BEDZ">was</token>
    <token tag="JJ">lying</token>
    <token tag="IN">on</token>
    <token tag="DT">the</token>
    <token tag="NN">floor</token>
  </tokens>
  <contextes>
    <contexte numELem="0">During the time I was lying on the floor in trance,
     I felt quite ill.</contexte>
    <contexte numELem="0">While I was lying on the floor zoned
    out I had a strange experience.</contexte>
    <contexte numELem="1">A man was lying on the floor,
    his hands and ankles bound.</contexte>
  </contextes>
</requete>
```

On constate que les mots sont correctement taggés :

1. was est bien la forme de be au passé
2. lying est bien un adjectif (JJ adjective, general (near))
3. on est bien une préposition (IN preposition (on, of))
4. the est bien un déterminant (DT determiner, general (a, the, this, that))
5. floor est bien un nom commun au singulier (NN : noun, common singular (action))

On constate de plus que le fichier est bien formé.

3.3 résultats de l'apprentissage des moyennes

Les textes dont proviennent les échantillons, ont été notamment trouvés sur le site du projet Gutenberg ainsi que sur des sites de news comme YAhoo news. J'ai essayé de prendre des textes de style différent (article

de journaux,articles scientifiques, romans ect.). Pour l'instant ma base d'apprentissage contient environ 10000 échantillons, ce qui n'est pas suffisant. Voici le résultat obtenu pour l'apprentissage des moyennes pour chaque bi-grammes, tri-grammes ect. :

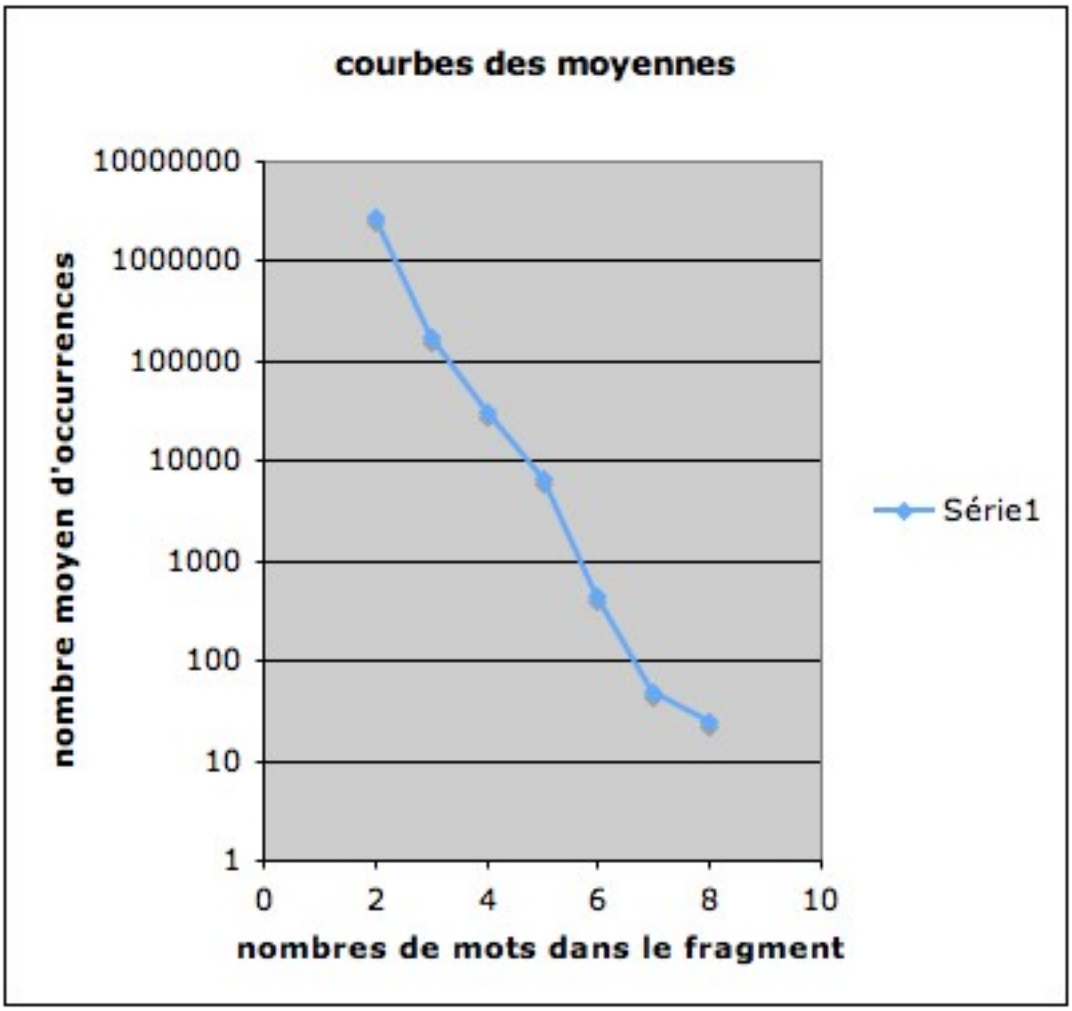


FIG. 3.1 – moyennes des n-grammes

3.4 résultats obtenus pour la mesure du caractère naturel d'un énoncé

voila quelques exemples de résultats :

fragment correct	certitude(%)	fragment incorrect	certitude(%)
”there was no reply”	100%		
”to come down”	100%		
”i made a mistake”	100%	”i did a mistake”	2%
”i m looking at her”	1%	”i m looking her”	1%
”he is looking at her”	8%	”he is looking her”	0%
”the important thing is to”	100%	”the important is to”	8%
”made up his mind”	100%		
”there are five of us in the house”	36%	”we are five in the house”	0%
”was lying on the floor”	100%		
”they sprinted in their socks”	0%		
”the bell rang at that moment”	15%		
”who speaks english”	7%	”that speaks english”	4%
”long black hair”	19%	”black long hair”	1%
”the last three days”	100%	”the three last days”	0%
”i answered your question”	12%	”i answered to your question”	1%

Ces résultat nous permettent d’affirmer qu’au dela d’un certain pourcentage, par exemple 50 % les phrases sont justes. Par contre on ne peut pas décidé pour les phrases fausses, car des phrases justes peuvent avoir des pourcentages très bas. Il nous faut donc d’autres algorithmes pour décider si l’énoncé est naturel ou non. Mais il nous permet au moins de décider immédiatement pour un certain nombre de phrases correctes.

Chapitre 4

Perspectives et Conclusion

4.1 Perspectives

Dans un premier temps il serait bien d'implémenter les autres mesures et d'améliorer l'interface graphique, en ajoutant par exemple des barres de progression représentant les différentes mesures. On pourrait aussi donner des coefficients à chaque mesure et donner une mesure globale utilisant les autres. Dans un second temps, il serait intéressant d'intégrer l'outil dans une Applet, et le mettre à disposition sur internet. Ainsi les gens pourraient tester leur énoncé directement sur internet. On pourrait aussi permettre à l'utilisateur de choisir un style (soutenu,relâché,scientifique,rapport ect...). On limiterait alors les recherches aux sites correspondant au style, comme par exemple les textes présents sur le site du projet Gutenberg, pour un style littéraire, ou le site d'un journal pour un style plutôt journalistique.

4.2 conclusion

Ce stage m'a intéressé, dans le sens ou il m'a permis de découvrir le TAL et d'approfondir mes connaissances en linguistique. J'ai pu m'intéresser notamment aux algorithmes d'apprentissage comme les chaines de markov ou les classifieurs Bayesiens, et j'ai ainsi pu découvrir l'utilisation des statistiques et des probabilités en intelligence artificielle. J'ai aussi pu appliquer plusieurs techniques que j'ai apprises au cours de la licence et de la maîtrise comme l'utilisation des expressions régulières, ou encore le parallélisme, mais aussi les techniques de modélisation enseignées en génie logiciel.

Bibliographie

1. La grammaire d'aujourd'hui (M. Arrivé. , F. Gadet, M. Galmiche)
2. Pratique de l'anglais de A à Z (M. Swan,F. Houdart)
3. Statistical language Learning (E. Charniak)
4. Mémoire : Erreurs lexicales : tentative d'analyse des systèmes compensatoires (N Becker)
5. Mémoire : Etude sur la modélisation stylistique destinée à la spécification d'un outil d'aide à la rédaction en langue étrangère (Catalina Chircu)
6. fr.wikipedia.org pour les chaînes de Markov