Guillaume Pinot
GI06
Suiveur : Philippe Trigano
Printemps 2005

Rapport de stage TN10

Correction orthographique en contexte

LINA
2, rue de la Houssinière
BP 92208
44322 Nantes Cedex 03
Responsable : Chantal Enguehard

Remerciements

Merci à Chantal Enguehard pour avoir proposé ce sujet et m'avoir suivi pendant ce stage de recherche.

Je remercie également Patrice Simard et Irena Rusu sans lesquels je n'aurais pas pu suivre le Master recherche en informatique « Système d'aide à la décision », dans le cadre duquel j'effectue ce stage de recherche.

Je remercie Philippe Trigano pour m'avoir suivi ainsi que Noëlle Trigano pour son travail de supervision des stages de GI à l'UTC.

Je tiens à remercier l'INRIA pour son formidable travail sur le langage OCaml, ainsi que les lecteurs de `fr.comp.lang.caml` pour leur aide dans son utilisation.

Je ne remercierai jamais assez Richard M. Stallman, Linus Torvald et tous les développeurs du logiciel libre pour avoir créé un système d'exploitation et des applications fiables sans lesquels je n'aurais pas pu réaliser ce travail dans d'aussi bonnes conditions.

Je remercie enfin Donald Knuth, Leslie Lamport et les développeurs du projet LaTeX3 pour le développement de LaTeX, logiciel avec lequel j'ai rédigé ce rapport.

Table des matières

Résumé Technique

Les erreurs cachées surviennent lorsqu'une ou plusieurs modifications d'un mot le transforment en un autre mot du dictionnaire.

exemple : *This chocolate cake is a famous* desert.

L'omission d'un *s* dans *dessert* fait apparaître le mot *desert*. Cette erreur n'est pas détectée par un correcteur orthographique classique car *desert* figure dans le dictionnaire.

Notre but est de trouver un algorithme permettant de corriger ces fautes de manière interactive. De plus, nous voulons que l'apprentissage se fasse automatiquement en utilisant uniquement des ressources faciles à obtenir comme du texte brut. De cette façon, notre correcteur peut facilement être adapté à toute langue.

Première partie

Contexte

Chapitre 1

L'Entreprise

1.1 Présentation

Le LINA (Laboratoire d'Informatique de Nantes Atlantique) est un laboratoire de recherche en informatique (LINA CNRS FRE 2729).

Il a été créé en 2004 et regroupe 140 personnes réparties en 9 équipes. Associé au CNRS, à l'Université de Nantes et à l'École des Mines de Nantes, le LINA compte également deux projets INRIA.

Le projet scientifique du LINA est de développer un pôle de compétences international en « sciences du logiciel » spécialisé sur deux principaux axes de recherche :

- Architectures logicielles distribuées ;
- Systèmes d'aide à la décision.

Présent dans plusieurs projets européens et fort de nombreuses relations internationales, le LINA a pour vocation de participer activement à l'établissement d'un pôle STIC (Sciences et Technologie de l'Information et de la Communication) à Nantes et, au-delà, dans l'ouest de la France.

1.2 Organisation

Le LINA est divisé en 9 équipes de recherche :

CD (Contraintes Discrètes) : Ce groupe de recherche se focalise sur la programmation par contraintes discrètes. Ses axes de recherches sont la classification des contraintes, l'explication et les algorithmes de filtrage.

COCOA (Contraintes Continues et Applications) : Son orientation est la résolution des formules du premier ordre sur les nombres entiers et réels, avec des contraintes non-linéaires. Il traite également de l'incertitude ou des préférences sur les données et des critères de performance.

COMBI (Combinatoire et Bio-informatique) : Leur but est de développer des modèles et des méthodes (principalement) combinatoires adaptés

aux traitements des objets et phénomènes biologiques. Leurs moyens sont des compétences en algorithmique discrète, en théorie des graphes, en complexité algorithmique, en probabilités et statistiques.

COD (Connaissances — Optimisation — Décision) : Avec comme objectif opérationnel *in fine* la diffusion de connaissances intelligibles pour l'aide à la décision, les travaux de recherche de l'équipe s'inscrivent dans un continuum « modélisation, extraction et gestion des connaissances » centré sur l'extraction de connaissances des données.

GDD (Gestion de Données Distribuées) : Les travaux de cette équipe s'organisent dans deux directions : les réseaux « *peer to peer* » et les clusters de PC.

GRIM (Gestion et Recherche d'Information Multimédia) : Cette équipe cherche des solutions pour gérer le nombre croissant de données que nous produisons, et plus particulièrement les données multimédia. Deux axes de recherche sont présents : la gestion de l'information multimédia et la génération de résumés à partir de bases volumineuses.

MOC (Modélisation par Objets et par Composants d'architectures logicielles) : L'équipe a pour objectif l'élaboration de nouveaux langages de modélisation d'architectures logicielles. Elle fédère ses activités autour de trois thèmes : Langages de conception de nouvelles architectures logicielles ; Spécification, validation et vérification d'architectures logicielles ; et Architectures logicielles et architectures de systèmes à base de connaissances.

OBASCO (Objets, Aspects et Composants) : Ce groupe de recherche possède trois grands axes sur le thème du génie logiciel : La programmation orientée composant, la programmation orientée aspect et l'extension de la programmation objet par la programmation orientée composant.

TALN (Traitement Automatique du Langage Naturel) : L'équipe possède deux thématiques de recherche : la fouille de textes et analyse de documents ainsi que la modélisation formelle du Langage Naturel.

C'est dans cette dernière équipe que j'effectue ce stage, c'est pourquoi sa présentation est développée ci-dessous.

1.3 L'équipe TALN

1.3.1 Présentation de l'équipe

L'équipe TALN (Traitement Automatique du Langage Naturel) s'est constituée lors de la création de l'IRIN [1] en 1992. L'équipe menait les re-

[1] Institut de Recherche en Informatique de Nantes. Laboratoire fondateur du LINA

cherches sur la découverte d'informations linguistiques dans les textes à l'aide de méthodes d'analyse partielle à base d'automates finis.

Cette thématique, si elle est toujours centrale, s'est élargie pour prendre en compte la complexité toujours croissante des documents textuels au sein du thème actuel fouille et analyse de documents.

En 1998, l'équipe TALN a intégré une nouvelle thématique concentrée sur les modèles formels de la syntaxe qui constitue le thème Modèles formels du langage naturel avec trois aspects particuliers : syntaxe, sémantique et inférence.

1.3.2 Axes de recherche

Fouille de textes et analyse de documents

La fouille de documents est une problématique de recherche apparue au début des années 1980. Elle concerne d'une part, l'exploration des données textuelles pour en extraire des informations linguistiques à tous les niveaux du traitement de la langue : morphologique, lexical, syntaxique, sémantique, et pragmatique, et d'autre part, leur analyse pour permettre d'accéder à l'information qu'elles véhiculent.

Modèles formels du Langage Naturel

Le traitement automatique des ressources du langage naturel n'est possible qu'avec une formalisation préalable. La modélisation formelle du langage naturel s'appuie sur quatre grands piliers : analyse, interprétation, acquisition et génération. L'équipe se focalise sur les trois premiers avec un accent plus particulier sur l'acquisition. Les deux premiers représentent les deux aspects intrinsèques de l'apprentissage automatique des ressources du langage naturel : l'aspect syntaxique et l'aspect sémantique.

1.3.3 Effectifs de l'équipe

Béatrice Daille, Professeur, est responsable de l'équipe.

L'équipe est composée de 2 Professeurs, 6 Maîtres de Conférences et 6 Doctorants.

Deuxième partie

Le Stage

Chapitre 2

Organisation du stage

2.1 Sujet de stage

Mon sujet consiste en l'élaboration d'un correcteur orthographique en contexte, et ce, indépendamment de la langue. Un tel correcteur utilise les mots situés autour du mot à corriger pour vérifier sa correction. Un exemple d'un tel correcteur est la correction grammaticale de MS Word.

Le but de ce stage est de réussir à détecter des « erreurs cachées », c'est-à-dire des erreurs transformant un mot en un autre un mot existant.

Par exemple, prenons la phrase « Ce *désert* au chocolat est très bon. ». Le mot « désert » est ici mal employé, nous devinons assez facilement grâce au contexte que l'auteur voulait utiliser le mot « dessert », notamment grâce à la présence de « chocolat » et « bon ».

Ce genre de problème a été traité pendant la deuxième partie des années 90, notamment par Andrew R. Golding dans [Gol95] et [GS96]. Sa technique consiste à prendre des ensembles de mots confondables (comme $\{\text{désert}, \text{dessert}\}$ par exemple) et ensuite à calculer le candidat le plus probable dans le texte. Cette méthode a été reprise dans d'autres publications ([JM97] par exemple). Nous ne souhaitons pas utiliser cette technique car elle nécessite une ressource linguistique ne pouvant pas être apprise automatiquement : les ensembles de mots confondables.

2.2 Déroulement du travail

2.2.1 Conditions de travail

Pour mon stage, je partage une salle de 12 ordinateurs avec les autres stagiaires en master de recherche ayant un responsable au LINA, pôle Faculté des Sciences et des Techniques de l'Université de Nantes. Nous sommes une quinzaine de stagiaires pour 12 ordinateurs, mais comme nous avons la possibilité de travailler chez nous, il y a la plupart du temps un ordinateur

de libre.

Les ordinateurs sont de marque Dell, avec un processeur à 2 GHz, 256 Mo de mémoire vive et un écran 17 pouces TFT. Ils sont installés sous Fedora Core 3. Ce matériel est confortable pour travailler correctement, mais peu pratique pour faire tourner des calculs (d'autant plus que ces machines ne supportent pas très bien une charge importante de travail).

Nous disposons d'un accès au Web confortable.

2.2.2 Planning

1–18 mars : Étude bibliographique sur la correction en contexte ;

21 mars au 1er avril : Développement de l'apprentissage en contexte ;

4–11 avril : Développement du correcteur en contexte ;

12–29 avril : Tests de l'application, améliorations et description de son mode de fonctionnement.

2–6 mai : Nettoyage du code pour le rendre plus simple et plus modulaire.

9 mai au 1er juin : Rédaction d'un article en anglais sur le correcteur pour la conférence RANLP (*Recent Advances in Natural Language Processing*) 2005 [1].

30–31 mai : Modification de l'algorithme de proposition du correcteur en contexte.

2–7 juin : Développement de l'apprentissage n-grammes.

8–15 juin : Développement de la correction n-grammes centrée.

18–24 juin : Développement de la correction n-grammes sur les extrémités.

27 juin au 13 juillet : Développement des sorties statistiques pour le correcteur en contexte.

18–29 juillet : Rédaction du rapport.

1er au 12 août : Fermeture du LINA.

15–29 août : Rédaction du rapport.

30 août au 5 septembre : Préparation de la soutenance orale.

6 septembre : Soutenance orale.

18–22 septembre : Conférence RANLP 2005 en Bulgarie.

2.3 Choix techniques

Le programme que je dois développer n'est pas un programme de production, mais un programme de recherche. Il n'est pas soumis aux exigences

[1] http://www.lml.bas.bg/ranlp2005/

de fonctionnalité, efficacité, interface d'un programme de production. Son but est de montrer des résultats, non d'être utilisé quotidiennement.

Le choix de l'implémentation n'est donc pas important, et je suis libre de faire ce que bon me semble.

J'ai donc choisi un langage portable, efficace, puissant et facile à utiliser. Mon choix s'est porté sur Objective Caml car les styles fonctionnel et impératif peuvent être utilisés. OCaml peut être compilé en natif, en *byte-code* ou être exécuté dans un interpréteur. Il possède de nombreux outils de développement ainsi qu'un *garbage collector*.

Pour pouvoir stocker et manier les données extraites des corpus, j'utilise SQLite, une base de données embarquée. Elle possède l'intérêt d'être légère, portable, efficace, de ne pas nécessiter de serveur (une base est tout simplement un fichier) et de posséder une interface pour OCaml.

2.4 Publication

Durant mon stage, j'ai rédigé un article en anglais pour la conférence RANLP 2005 (*Recent Advances in Natural Language Processing*) se déroulant en Bulgarie. Cet article est co-écrit avec Chantal Enguehard, ma suiveuse.

La rédaction de cet article s'est déroulé en plusieurs étapes :

- Rédaction de l'article en anglais.
- Soumission de l'article à la conférence.
- Réception de l'admission pour la conférence en tant que poster avec les notes des relecteurs de l'article.
- Modification de l'article.
- Envoi de la version finale de l'article pour impression.
- Participation à la conférence.

Cet article me permet d'entrer dans le monde de la recherche en obtenant ma première publication. Il me permet également de découvrir ce monde de l'intérieur en allant et participant à une conférence internationale scientifique.

Troisième partie

La correction des erreurs cachées

Chapitre 3

Description du problème

Les correcteurs orthographiques distribués avec les traitements de textes courants tels que MS Word ou OpenOffice sont basés sur l'utilisation d'un dictionnaire. Le texte est analysé mot à mot : chaque mot ne figurant pas dans le dictionnaire est supposé erroné, des propositions pour la correction sont alors proposées à l'utilisateur. Paradoxalement, ces correcteurs voient leurs performances en détection d'erreurs se dégrader avec l'augmentation de la taille du dictionnaire car ils sont incapables de détecter les erreurs cachées.

Les erreurs cachées surviennent lorsqu'une ou plusieurs modifications d'un mot le transforment en un autre mot du dictionnaire.

exemple : *This chocolate cake is a famous* desert.

L'omission d'un *s* dans *dessert* fait apparaître le mot *desert*. Cette erreur n'est pas détectée car *desert* figure dans le dictionnaire.

Ce problème a été abordé pendant la deuxième partie des années 90, notamment par Andrew R. Golding dans [Gol95] et [GS96]. Il définit des ensembles de mots de confusions (comme {désert, dessert} par exemple) et ensuite détermine, en examinant le texte, lequel de ces mots est le meilleur candidat. Cette méthode a été reprise dans d'autres papiers comme [JM97, GS96, MB97].

Notre but est de trouver un algorithme permettant de corriger ces fautes dans le cadre d'un correcteur orthographique interactif. De plus, nous voulons que l'apprentissage se fasse automatiquement en utilisant uniquement des ressources faciles à obtenir comme du texte brut. De cette façon, notre correcteur peut facilement être adapté à toute langue [1].

Nous nous fixons de telles exigences car ce projet a pour but d'être utilisé sur des langues peu dotées en ressources linguistiques comme le bambara et d'autres langues africaines. Ces langues ne possèdent pas de ressources linguistiques autre que des textes bruts.

[1] La seule restriction est la possibilité de définir des mots dans la langue traitée.

Chapitre 4

État de l'art

La détection et la correction des erreurs cachées est un thème de recherche qui fut étudié principalement dans la fin des années 90. Les travaux de cette période ont été initiés par Andrew R. Golding dans [Gol95]. Ses recherches reposent sur des ensembles confondables.

Par la suite, quelques autres publications non fondées sur les ensembles de confusions ont traité ce sujet.

4.1 Méthodes fondées sur les ensembles de confusions

Un ensemble de confusion est un ensemble de mots pouvant être confondus entre eux, en raison de leur orthographe proche ($\{\text{dessert}, \text{desert}\}$) ou parce qu'ils sont souvent confondus dans leur usage ($\{\text{between}, \text{among}\}$). Ces ensembles sont une ressource linguistique indispensable dans ce type de détection.

4.1.1 *A Bayesian hybrid method for context-sensitive spelling correction* [Gol95]

Cet article est le premier d'une série d'articles sur le sujet. Il introduit la correction des erreurs cachées en utilisant les ensembles confondables.

Cet article présente deux méthodes (*context words* et *collocations*) et deux façons de les combiner.

Méthode *context words*

Lors de l'apprentissage sur un texte brut, il est créé un ensemble de $(w_c, w_i, P(w_c|w_i))$, avec w_c un mot appartenant aux ensembles de confusions, w_i un mot quelconque et $P(w_c|w_i)$ la probabilité d'avoir le mot w_c sachant que w_i est dans son contexte. Pour corriger un mot, les probabilités de tous

les mots de chaque ensemble de confusions sont calculées, la probabilité la plus élevée est proposée en correction.

D'après l'auteur, cette méthode donne des résultats satisfaisants lorsque l'erreur est d'ordre sémantique (comme {dessert, desert}). Il a donc conçu un autre algorithme se basant plus sur la syntaxe pour corriger les erreurs restantes : la méthode *collocations*.

Méthode *collocations*

L'approche pratique est très proche de *context words*. Théoriquement, Golding utilise des n-grammes [1] sur des étiquettes grammaticales [2] des mots ou sur des mots.

Par exemple, pour le mot *desert* et pour la suite de 3 mots « *in the desert* », nous pouvons avoir comme n-gramme :

- `in the desert` (les 3 mots directements).
- `PREP the desert` (une étiquette pour le mot *in*).
- `PREP DET desert` (des étiquettes pour les mots *in* et *the*).
- `in DET desert` (une étiquette pour le mot *the*).

Cette technique permet d'utiliser les étiquettes pour un meilleur résultat dans le cas général ainsi que de garder les mots exacts pour les cas particuliers (*in the desert* est une suite courante de termes).

En pratique, des 3-grammes sont utilisés. Le fait d'utiliser des n-grammes et des étiquettes permet de corriger les erreurs de syntaxe.

Avant la correction, le mot à corriger est remplacé par l'ensemble contenant le mot lui-même et l'ensemble de ses étiquettes possibles [3]. De cette façon, aucun étiqueteur [4] n'est utilisé durant la correction. En effet, la présence de fautes dans la phrase à étiqueter risque de rendre l'étiquetage non valide.

Méthodes hybrides

Ces deux méthodes sont complémentaires entre elles car l'une donne de meilleurs résultats sur les erreurs sémantiques et l'autre sur les erreurs syntaxiques. Golding propose donc deux méthodes permettant de les combiner pour obtenir un correcteur polyvalent.

La première méthode est basée sur des listes de décisions. Les différentes données (3-grammes et probabilités contextuelles) sont ordonnées par ordre

[1] Un n-gramme est un n-uplet de mots. Ils sont généralement utilisés pour la description d'une grammaire basique.

[2] Une étiquette grammaticale représente la classe grammaticale d'un mot (comme adverbe, adjectif féminin, etc.). Le nombre d'étiquettes est fixé, et varie selon les implémentations et les langues.

[3] Un mot peu avoir plusieurs étiquettes selon le contexte. Par exemple, « poli » peut être un nom (le poli d'un galet) ou un adjectif (un enfant poli, un diamant poli).

[4] Programme assignant une étiquette aux mots d'une phrase.

de fréquence. Lors de la correction, la première donnée concordante est utilisé.

La deuxième sur un classificateur Bayesiens (appelé par la suite « méthode Bayes »). Cette méthode fonctionne comme la méthode des listes de décisions sauf que, durant la correction, toutes les données concordantes sont utilisées. Cette dernière méthode semble être la meilleure.

4.1.2 Autres articles utilisant les ensembles de confusions

D'autres articles ont été écrits par la suite en se basant sur les ensembles de confusions. On peut différencier deux groupes : les articles écrits par Golding et les autres.

Articles de Golding

Golding a écrit deux articles dans la prolongation de [Gol95] : [GS96] avec Yves Schabes et [GR99] avec Dan Roth. Les méthodes utilisées dans ces articles sont comparées entre elles. [GR99] semble être la meilleure.

[GS96] expose une méthode fondée sur des trigrammes d'étiquettes. Cette méthode fonctionne bien lorsque les mots de l'ensemble de confusions ont des étiquettes différentes. Lors de la correction, la méthode trigrammes est appelée si l'ensemble de confusions s'y prête ; sinon, la méthode Bayes de [Gol95] est utilisée. Cette hybridation donne de bons résultats en général et de meilleurs que la méthode de Bayes utilisée seule.

[GR99] utilise un réseau de neurones type « winnow » pour utiliser les données de la méthode *context words* (un mot est situé à proximité d'un autre) et de la méthode *collocations* (n-grammes d'étiquettes et de mots). Cet article est le dernier de Golding sur ce sujet et semble exposer l'algorithme le plus efficace, surtout pour les apprentissages volumineux.

Autres articles

D'autres articles fondés sur les travaux de Golding ont été publiés. Ils décrivent de nouveaux algorithmes basés sur les ensembles de confusions.

Michael P. Jones et James H. Martin écrivent [JM97] en 1997. Leur algorithme utilise le *latent semantic analysis* [5] adapté pour la correction en contexte. Cette méthode semble donner des résultats comparables à [GS96].

Lidia Mangu et Eric Brill décrivent dans [MB97] un algorithme fondé sur des règles de transformation. Les résultats sont comparables à [GS96] mais inférieurs à [GR99]. Par contre, cette méthode a l'avantage d'utiliser des données directement compréhensibles par un humain (en moyenne une vingtaine de règles pour un ensemble de confusions).

[5] Technique utilisant une représentation vectorielle des mots en fonction de leur contexte.

4.2 Autres méthodes

Par la suite sont apparus quelques articles traitant de la correction en contexte, mais les ensembles de confusions de Golding ne sont plus utilisés. Les recherches sur le sujet semblent plus rares mais toujours existantes comme nous le montrent ces quelques articles plus récents.

G. Hirst et A. Budanitsky décrivent dans [HB05] un algorithme de correction basé sur une distance sémantique. L'idée est de chercher les mots n'ayant pas de relation sémantique dans le paragraphe courant, puis de proposer un mot possédant une telle relation à la place. Le thésaurus hiérarchique de WordNet est utilisé comme donnée.

En 2005, Chiraz Ben Othmane Zribi, Fériel Ben Fraj et Mohamed Ben Ahmed exposent dans [ZFA05] un système multi-agent pour la langue arabe. Il se décompose en un groupe d'agents syntaxiques et un groupe d'agents sémantiques tout deux coordonnés, ainsi que des agents de correction et de génération de propositions.

4.3 Bilan

Le sujet de la correction des erreurs cachées n'est pas un nouveau problème. Différents algorithmes ont été proposés, notamment dans les années 90.

La recherche sur ce sujet n'est plus très active, mais est toujours présente. L'inconvénient des différentes approches proposées jusqu'ici est qu'elles sont, d'une manière ou d'une autre, fortement liées à des ressources linguistiques comme les ensembles de confusions, un étiqueteur ou un thésaurus.

Nous pouvons également constater que de tels correcteurs ne sont pas présents dans les applications grand public malgré l'absence de correcteur grammatical dans beaucoup de langues et d'applications. Nous pouvons trouver deux correcteurs utilisant le contexte : la correction utilisée dans le moteur de recherche de Google ainsi que *Context-Sensitive Spelling Checker* d'IBM. Cette non-utilisation d'une telle correction de façon courante peut s'expliquer par le fait qu'il n'existe pas d'algorithme efficace et utilisable simplement.

Chapitre 5

Définitions générales

Dans ce chapitre, nous allons voir des outils nécessaires à la réalisation de notre correcteur en contexte.

5.1 Distance entre deux mots

Soit edist(w_i, w_j) une fonction comparant 2 chaînes de caractères et retournant un nombre positif avec comme condition

$$\text{edist}(w_i, w_j) = 0 \Leftrightarrow w_i = w_j$$

Plus edist(w_i, w_j) est grand, plus w_i est éloigné de w_j.

La distance utilisée dans Aspell [A+05] prend en compte la phonétique des mots. Elle a donc besoin d'une connaissance linguistique et dépend du langage cible. Cette distance n'est donc pas adapté à notre besoin.

Dans cette première version, nous avons choisi d'utiliser la distance minimale d'édition [WF74] qui est totalement indépendante du langage cible. Nous avons légèrement modifié cette fonction pour réduire le coût d'une inversion de lettres. Pour la calculer, il faut compter le nombre minimal d'insertion, de suppression, de substitution et d'inversion de lettres pour transformer un mot en un autre.

Exemple :

$$\text{test} \xrightarrow{s \to x} \text{te}\mathbf{x}\text{t} \xrightarrow{+e} \text{text}\mathbf{e}$$

ce qui donne une distance minimale d'édition de 2.

5.2 Similitude entre deux mots

Pour déterminer si un mot est une correction plausible d'un mot à corriger, nous utilisons une fonction sim(w_i, w_j), qui prend en argument deux mots et retourne vrai si les deux mots sont similaires et faux sinon.

Soit ϵ la chaîne vide :

$$\text{sim}(w_i, w_j) = \begin{cases} \text{vrai si } \text{edist}(w_i, w_j) \leq \dfrac{\text{edist}(w_i, \epsilon) + \text{edist}(w_j, \epsilon)}{\gamma} + c \\ \text{faux sinon} \end{cases}$$

En pratique, $\gamma = 8$ et $c = 1$. Ces valeurs ont été déterminées après plusieurs expérimentations.

5.3 Dictionnaire

Le but est de répertorier tout les mots apparaissant dans un corpus ainsi que leur fréquence.

Le corpus est parcouru mot à mot. Soit w_c le mot courant.

Soit D le dictionnaire, constitué d'un ensemble de paires $D_i = (w_i, c_i)$, les w_i étant des mots. Chaque w_i est unique. c_i est le compteur correspondant au nombre d'occurrences du mot w_i.

Le mot courant w_c est traité pour la constitution du dictionnaire :

– Si D_c existe, c_c est incrémenté.

– Sinon, $D_c = (w_c, 1)$ est ajouté à D.

Ainsi, nous possédons le nombre d'apparitions de chaque mot du corpus (voir exemple figure 5.1). Cette information permettra de calculer diverses probabilités par la suite.

w_i	de	la	et	le	il	l'	à	...
c_i	19797	14256	12419	11495	11479	10204	9066	...

FIG. 5.1 – Exemple de dictionnaire

Chapitre 6

Méthode du contexte non-ordonné

6.1 Idée intuitive de l'algorithme

Notre algorithme se décompose en deux parties distinctes : l'apprentissage qui permet d'acquérir les données nécessaires au correcteur, puis la correction en elle-même.

6.1.1 L'Apprentissage

Le but de l'apprentissage est d'apprendre les contextes de mots, le contexte d'un étant défini par les mots situés autour de ce mot. Il est alors possible de savoir quel mot a le plus de chance d'apparaître dans un certain contexte.

Notre algorithme apprend ces probabilités grâce à un texte brut : il le parcourt et mémorise les données dont il a besoin.

6.1.2 La détection et la correction

Pour corriger un mot w_c, nous observons les mots apparaissant dans son contexte. Il s'agit de détecter s'il existe un mot w_i qui apparaît souvent dans le même contexte que w_c et qui soit proche de w_c au sens de la distance minimale d'édition. Dans ce cas, il est plausible qu'une erreur ait transformé w_i en w_c.

Pour chaque mot w_k du contexte, notre algorithme note les mots w_i apparaissant dans un contexte comprenant w_k et étant proche du mot à corriger w_c. Cette méthode est plus efficace que d'effectuer la méthode intuitive qui consiste à trouver dans un premier temps tous les mots proches de w_c au sens de la distance minimale d'édition, puis de vérifier s'ils apparaissent dans un contexte approchant celui de w_c. Ainsi, nous simplifions la complexité du problème.

Ensuite, grâce aux données récoltées durant la phase d'apprentissage, nous pouvons estimer, grâce à une heuristique, si l'écriture est erronée et, si c'est le cas, ordonner les propositions de manière pertinente.

Nous présentons par la suite les deux phases distinctes de notre algorithme de manière détaillée : l'apprentissage des probabilités contextuelles puis la détection et la correction d'erreurs.

6.2 Apprentissage

L'apprentissage est réalisé sur un corpus brut. Cet algorithme est paramétré par k : le nombre de mots pris pour le contexte.

6.2.1 Dictionnaire des contextes C

Le corpus est parcouru mot à mot. Soit w_c le mot courant.

Définition

Le dictionnaire des contextes, nommé C, rassemble les cooccurrences des mots w_i et w_j, ces mots étant distants d'au plus k mots. L'ordre des mots n'est pas pris en compte. Chaque cooccurrence est assortie de sa fréquence $f_{i,j}$:

$$C = \{C_{i,j} \quad / \quad C_{i,j} = (\{w_i, w_j\}, f_{i,j})\}$$

(voir exemple figure 6.1).

w_i	lui	sorte	dans	l'	la	de	la	...
w_j	un	une	du	n'	sa	fait	maison	...
$f_{i,j}$	199	199	199	198	198	198	197	...

FIG. 6.1 – Exemple de dictionnaire des contextes avec $k = 3$

Algorithme

Le corpus est parcouru mot à mot. Lors du traitement du mot courant w_c, les $C_{c,j}$ avec $j \in [c-k, c-1]$ sont calculés. Il s'agit des k cooccurrences de w_c avec les mots apparaissant dans une fenêtre de largeur k précédant w_c (voir figure 6.2) :

- Si $C_{c,j}$ existe, $c_{c,j}$ est incrémenté.
- Sinon, $C_{c,j} = (\{w_c, w_j\}, 1)$ est ajouté à C.

Nous obtenons ainsi tous les ensembles de 2 mots situés à une distance inférieure ou égale à k, ainsi que leur fréquence.

Soit $k = 3$ et $c = 6$

On	peut	s'	assurer	un	bon	revenu.
w_1	w_2	w_3	w_4	w_5	w_6	w_7

w_3, w_4, et w_5 sont dans le contexte de w_6, le mot courant. Les cooccurrences $\{w_3, w_6\}$, $\{w_4, w_6\}$ et $\{w_5, w_6\}$ sont relevées.

FIG. 6.2 – Exemple de contexte lors de la lecture du corpus

Complexité

La complexité en espace est de $O(kn)$ avec n la taille du corpus en nombre de mots. En pratique, elle devrait être inférieure à cause de la redondance des mots et des cooccurrences.

Soit $O(f(x))$ la complexité de la recherche d'une occurrence particulière dans C, avec x la taille de l'ensemble C. Comme la taille de cet ensemble peut être majorée par kn, la complexité en temps est $O(knf(kn))$.

6.2.2 Calcul des probabilités

Nous utilisons les données acquises pendant la lecture du fichier pour calculer les probabilités des contextes notés dans C, le dictionnaire des contextes. Les fréquences c_i des mots sont obtenues *via* le dictionnaire D (voir section 5.3 page 16)

Soit B le dictionnaire des couples de mots associés à leur probabilité (voir exemple figure 6.3). Nous utilisons ici des couples (ordonnés) car la probabilité associée dépend de l'ordre des mots ($P(w_i|w_j) \neq P(w_j|w_i)$). Nous avons donc :

$$B_{i,j} = ((w_i, w_j), P(w_i|w_j))$$

w_i	une	sorte	un	lui	du	dans	...
w_j	sorte	une	lui	un	dans	du	...
$P(w_i\|w_j)$	0.6982	0.0365	0.0750	0.0267	0.0400	0.0533	...

FIG. 6.3 – Exemple de dictionnaire de couples de mots avec $k = 3$

Pour chaque $C_{i,j}$, sont définis $B_{i,j}$ et $B_{j,i}$. La probabilité est calculée ainsi :

$$P(w_i|w_j) = \frac{c_{i,j}}{c_j}$$

6.3 Algorithme de détection et de correction

6.3.1 Élaboration de la liste de propositions

Soit K_c le contexte d'un mot w_c. K_c est l'ensemble des $2k$ mots situés autour de w_c, soit les k mots situés avant et les k situés après (voir figure 6.4).

Soit $k = 3$ et $c = 6$

On	peut	s'	assurer	un	bon	revenu	en	travaillant	sérieusement.
w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}

Nous avons ici w_3, w_4, w_5, w_7, w_8 et w_9 dans le contexte du mot courant w_6, donc

$$K_6 = \{w_3, w_4, w_5, w_7, w_8, w_9\}$$

FIG. 6.4 – Exemple de contexte lors de la correction

Nous cherchons les mots déjà apparus dans le contexte des mots qui entourent le mot courant et proches du mot courant, ce qui limite la complexité de l'algorithme.

Soit w_c le mot à corriger. Pour chaque $w_j \in K_c$, nous constituons l'ensemble F_j tel que

$$\begin{aligned} F_j = \{B_{i,j} \in B| \\ \text{sim}(w_c, w_i) = \text{vrai}, \\ P(w_i|w_j) > P(w_c|w_j)\} \end{aligned}$$

(Pour la fonction similaire, voir section 5.2 page 15.)

Nous obtenons $2k$ ensembles de propositions. L'ensemble des propositions possibles, F, est la réunion des ensembles F_j (voir exemple figure 6.5).

Une heuristique calcule un score pour chaque proposition afin de les ordonner de la plus pertinente à la moins pertinente. Nous proposons ici une heuristique, mais des tests approfondis seront nécessaires pour l'affiner.

Soit G_j le sous-ensemble de F tel que

$$G_j = \{B_{i,j}, B_{i,j} \in F\}$$

On peut alors définir l'heuristique H_j suivante qui calcule un score pour chaque proposition :

$$H_j = |G_j| + \prod_{B_{i,j} \in G_j} P(w_i|w_j)$$

Cette heuristique favorise les propositions apparaissant dans plusieurs ensembles de propositions F_j, puis ensuite, les probabilités les plus fortes (voir l'exemple figure 6.5).

Soient $k = 3$ et $w_c =$ game

And

so	my	mind	game	round	to	the
0.03086	0.01207		came	0.07317	0.01620	0.01571
			same			0.00523
			gate		0.00324	0.00305
	0.00966		gave		0.00324	0.00174
0.00617			name			0.00087
			game			0.00043

business.

Les nombres sont les $P(w_i|w_j)$ calculés lors de la phase d'apprentissage. Par exemple $P(\text{round}|\text{came}) = 0.07317$

$$G_1 = \{((\text{came}, \text{so}), 0.03086), ((\text{came}, \text{my}), 0.01207), \\ ((\text{came}, \text{round}), 0.07317), ((\text{came}, \text{to}), 0.01620), \\ ((\text{came}, \text{the}), 0.01571)\}$$

Nous ne détaillons pas $G_2, \ldots, G_6$ qui sont construit de la même manière.
L'heuristique donne la meilleure note à la première proposition *came* car elle apparaît cinq fois.

Pour cet exemple, notre algorithme a utilisé *The War of the Worlds* de H. G. Wells comme corpus d'apprentissage. la phrase vient de *The Time Machine* de H. G. Wells (Chapitre 3). (Nous avons introduit une erreur en modifiant le mot original « *came* » par « *game* ».)

FIG. 6.5 – Propositions

6.3.2 Détection

Pour détecter les erreurs, nous disposons de deux heuristiques de détection $H1$ et $H2$ fondées sur l'heuristique H_j vue plus haut.

Pour $H1$, le mot w_c est considéré comme correct si la proposition possédant le score le plus fort est le mot w_c lui-même.

Pour $H2$, le mot w_c est considéré comme correct si la partie entière du plus haut score est égale à la partie entière du score du mot w_c.

Bien sûr, si le mot à corriger n'est pas dans la liste des propositions, le mot est détecté comme une erreur puisque son score est nul.

Nous obtenons ainsi un algorithme complet permettant la correction des erreurs cachées. Il est relativement proche d'autres méthodes présentes dans la littérature.

6.4 Comparaison avec l'existant

Cet algorithme est proche de la méthode *context words* de Andrew R. Golding dans [Gol95] (voir section 4.1.1 page 11).

Ces deux méthodes utilisent la même idée : utiliser les mots présents autour du mot à corriger. Elles diffèrent surtout dans l'approche du problème : Andrew R. Golding ne se préoccupe pas de la détection des erreurs puisqu'il en fait une liste *a priori* : l'ensemble des confusions. L'inconvénient est que la préparation de ces données exige un haut niveau d'expertise linguistique et qu'elle est coûteuse en temps. De plus, une telle liste ne pourra jamais être considérée comme complète.

Notre méthode porte sur la détection des erreurs. Elle est générique, ne nécessite pas d'intervention humaine et n'est pas spécialisée pour une langue particulière. Toutefois elle est limitée aux langues dont la transcription fait apparaître des mots.

La phase d'apprentissage est très semblable dans les deux approches, mais elle est limitée aux ensembles de confusions pour la méthode de Golding alors qu'elle porte sur tout le corpus dans notre approche.

La phase de correction est différente dans les deux approches. Golding calcule la probabilité de chaque mot de l'ensemble de confusion grâce au contexte puis propose le mot ayant la probabilité la plus élevée. Notre méthode utilise le contexte pour créer la liste des propositions, qui sera ensuite filtrée grâce à la fonction « sim » et aux probabilités des propositions.

Chapitre 7

Expérimentations

7.1 Méthode expérimentale

Nos expérimentations sur le correcteur en contexte portent sur le texte intégral des *Misérables* de Victor Hugo. Ce corpus est divisé en 10 parties à peu près égales (Voir tableau 7.1). Un programme insère des erreurs cachées dans le texte selon un certain pourcentage.

Partie	Nombre de mots	Parties des *Misérables*
0	55451	Tome 1 livres 1 à 3
1	59127	Tome 1 livres 4 à 8
2	53332	Tome 2 livres 1 à 4
3	52915	Tome 2 livres 5 à 8, Tome 3 livre 1
4	47882	Tome 3 livres 2 à 7
5	53405	Tome 3 livre 8, Tome 4 livres 1 et 2
6	59836	Tome 4 livres 3 à 8
7	44698	Tome 4 livres 9 à 15
8	54349	Tome 5 livres 1 à 3
9	52998	Tome 5 livres 4 à 9
Total	533993	*Les Misérables*

Tab. 7.1 – Découpage du corpus

L'apprentissage est effectué sur 9 parties choisies de manière aléatoire, et la correction est effectuée sur le jeu restant dans lequel ont été introduites des erreurs. Lors de la correction, nous récoltons des informations et calculons des statistiques pour évaluer les performances du correcteur.

7.1.1 Statistiques sur la détection

Pour évaluer la détection d'erreur, nous utilisons les taux de précision et de rappel, indicateurs classiques en classification. Ces indicateurs sont compris entre 0 et 1, 1 traduisant le résultat parfait.

Pour calculer ces indices, nous avons besoin de nommé différents ensembles. Soit D l'ensemble total de mots, D^+ les mots erronnés et D^- les mots correctes. D^+ et D^- forment une partition de D. Soit R l'ensemble des mots détectés comme erronés. Certains mots de R font partie de D^+, d'autres de D^-.

La précision fait ressortir sous forme d'un indice la proportion de mots détectés comme erronés alors que ce n'est pas le cas. Sa formule est :

$$\text{Précision}_{\text{détection}} = \frac{|D^+ \cap R|}{|R|}$$

Le rappel met en évidence la proportion d'erreurs relevées. Sa formule est :

$$\text{Rappel}_{\text{détection}} = \frac{|D^+ \cap R|}{|D^+|}$$

Ces indicateurs permettent d'évaluer de manière efficace la détection d'erreurs de notre algorithme.

7.1.2 Statistiques sur la correction

Pour évaluer la correction, nous calculons divers indicateurs.

L'indicateur proportion de correction permet de voir la proportion de mots correctement corrigés. Un mot est correctement corrigés si le mot d'origine est dans la liste des propositions. Ce calcul s'effectue uniquement sur les mots correctement détectés comme erronés. Sa formule est :

$$\text{Prop}_{\text{correction}} = \frac{\text{Nombre de mots corrigés à juste titre}}{\text{Nombre de mots détectés à juste titre comme erronés}}$$

Pour évaluer l'ordonnancement des propositions, nous calculons deux moyennes : la position moyenne de la bonne correction dans la liste des propositions et la moyenne du nombre de propositions. Ces indices nous permet d'évaluer la pertinence des informations que reçoit l'utilisateur.

7.2 Ajout des erreurs

L'ajout des erreurs cachées dans le texte est réalisé automatiquement. Il n'utilise pas de ressource externe.

Cet ajout se déroule en deux temps : la génération des erreurs possibles, puis l'ajout de ces erreurs dans le texte.

7.2.1 Génération des erreurs possibles

Pour chaque mot présent dans le corpus (*Les Misérables* dans notre cas), sont cherchés les mots qui lui sont proches et qui constituent donc les mots avec lequels peut se produire une confusion.

Soit D un dictionnaire simple (un ensemble de mots). Pour chaque mot $w_i \in D$, associer un ensemble de mots appartenant à D et proches de w_i (grâce à la fonction « sim » par exemple). Un ensemble d'erreurs possibles pour chaque mot est alors obtenu.

Certains ensembles d'erreurs possibles peuvent être vides quand il n'existe pas de mots proches du mot visé. Cependant, 84% en possède un. Nous obtenons donc une base d'erreurs possibles assez etoffée pour l'utiliser.

Ces données permettent de générer facilement des erreurs cachées.

En pratique, nous utilisons comme dictionnaire les mots apparaissant plus de dix fois dans *Les Misérables*. De cette façon, les erreurs ajoutées au corpus impliquent des mots possédant un contexte connu par notre correcteur.

7.2.2 Ajout des erreurs cachées

Cet algorithme ajoute des erreurs à un texte donné. Il prend deux paramètres en entrée : la probabilité d'ajouter une erreur et une base d'erreurs possibles précédemment déterminés.

L'ajout d'erreur est alors facile en utilisant les données récoltées. Chaque mot du corpus est affecté ou non d'une erreur selon la probabilité d'erreur désirée au moyen d'un tirage aléatoire.

Les erreurs sont insérées dans le texte grâce à un balisage XML. Chaque erreur est indiquée ainsi que le mot d'origine (la correction que nous souhaitons trouver). La sortie est donc du type :

```
J'accuse Monsieur bien qu'il
<error correction="soit">soie</error> innocent.
```

7.3 Résultats et interprétation

Un résumé des résultats est donné au tableau 7.2.

Nous constatons que la précision de la détection est très mauvaise. Elle est située entre 0,019 et 0,156. Ce résultat est dû à la surdétection de notre algorithme. Notre algorithme de correction n'est pas utilisable tant que ce problème n'est pas résolu.

Nous pouvons remarquer une différence importante entre les taux précision de la détection sur les échantillons à 1% d'erreurs (de 0,019 à 0,03) et les échantillons à 10% (entre 0,101 et 0,156). Ce phénomène peut s'expliquer simplement : si le nombre l'erreurs est plus important, le nombre de

Part.	Err.	Dét.	Précision	Rappel	Prop. Cor.	PMC	MP
0	10%	H1	0.1004138	0.9453621	0.9560931	2.71	14.02
0	10%	H2	0.1462171	0.8888417	0.9632458	2.68	14.28
0	1%	H1	0.0193273	0.9351851	0.9579207	2.71	14.11
0	1%	H2	0.0300665	0.8972602	0.9745547	2.46	14.81
5	10%	H1	0.1081926	0.9363030	0.9622054	2.57	14.49
5	10%	H2	0.1561469	0.8800168	0.9635141	2.63	14.63
5	1%	H1	0.0206164	0.9615384	0.9500000	2.29	14.16
5	1%	H2	0.0301319	0.9120603	0.9696969	2.56	14.56
6	10%	H1	0.1012948	0.9393822	0.9539662	2.52	14.10
6	10%	H2	0.1518797	0.8811320	0.9627408	2.57	14.49
6	1%	H1	0.0212247	0.9447731	0.9457202	2.58	14.38
6	1%	H2	0.0294668	0.8720682	0.9584352	2.94	16.73

Légende :
– Part. : partie des *Misérables* corrigée.
– Err. : pourcentage d'erreurs ajoutées.
– Dét. : heuristique de détection utilisée.
– Précision : précision de la détection.
– Rappel : rappel de la détection.
– Prop. Cor. : proportion de correction.
– PMC : Position moyenne du mot correct au sein des propositions.
– MP : nombre moyen d'éléments dans la liste des propositions.

Tab. 7.2 – Résultats

mots justes l'est moins, et donc la surdétection est moins importante. Nous voyons donc que la précision est fortement liée au taux d'erreurs présentes.

Par contre, nous obtenons un très bon rappel. Il est compris entre 0,87 et 0,94. Nous avons donc une part très élevée d'erreurs effectivement détectées.

Le mot correct est très souvent présent dans la liste des propositions (entre 95 et 97% des cas). La correction en elle-même est donc efficace.

La position du mot correct est satisfaisante : elle se trouve en moyenne en 2,7^{e} position donc au début de la liste des propositions, ce qui est l'objectif visé.

Par contre, le nombre de propositions est un peu élevé (une quinzaine de propositions en moyenne). Il peut facilement être réduit en enlevant les propositions les moins pertinentes, le mot correct se trouvant rarement dans les dernières positions.

Nous pouvons également remarquer les effets différents de $H1$ et $H2$. Le passage de H_1 à H_2 multiplie la précision d'environ 1,5 et diminue légèrement le rappel (au maximum de 8%). Bien que les résultats obtenus avec H_2 soient meilleurs que ceux avec H_1, ils ne sont pas pour autant satisfaisants.

Cependant, ces travaux avec deux heuristiques très proches, et qui montrent leur forte influence sur les résultats, nous laisse penser que nos travaux doivent s'orienter vers la recherche d'une heuristique permettant d'obtenir de meilleurs résultats.

Conclusion

Nous avons présenté un algorithme utilisant un contexte non ordonné pour corriger les erreurs cachées.

Les avantages de cet algorithme sont :

- la simplicité ;
- l'indépendance de toute information linguistique hormis la définition d'un mot ;
- l'utilisation d'un corpus brut pour l'apprentissage ;
- le peu de paramètres à régler (k la taille du contexte).

Les inconvénients sont :

- la taille des données générées par l'apprentissage.
- la précision : l'algorithme propose des corrections pour beaucoup de mots corrects.

Cet algorithme est destiné à être utilisé lors de la correction interactive d'un texte. Dans ce cadre, sa rapidité devrait être suffisante. Par contre, la surdétection d'erreurs constitue un réel problème.

Nous orientons donc nos recherches vers la définition d'une meilleure heuristique d'ordonnancement des propositions. Nous envisageons également de définir les contextes de manière ordonnée afin de capter des contraintes de syntaxe en plus de la sémantique.

Ces différentes modalités seront évaluées précisément sur le même corpus afin d'en analyser la pertinence.

Bibliographie

[A+05] Kevin Atkinson *et al.* : GNU Aspell. `http ://aspell.net/`, 2005.

[AGSV98] Eneko Agirre, Koldo Gojenola, Kepa Sarasola et Atro Voutilainen : Towards a single proposal in spelling correction. *In Proceedings of the 17th international conference on Computational linguistics*, pages 22–28. Association for Computational Linguistics, 1998.

[CH89] Kenneth Ward Church et Patrick Hanks : Word association norms, mutual information, and lexicography. *In Proceedings of the 27th annual meeting on Association for Computational Linguistics*, pages 76–83, Morristown, NJ, USA, 1989. Association for Computational Linguistics.

[Gol95] Andrew R. Golding : A bayesian hybrid method for context-sensitive spelling correction. *CoRR*, cmp-lg/9606001, 1995.

[GR99] Andrew R. Golding et Dan Roth : A winnow-based approach to context-sensitive spelling correction. *Mach. Learn.*, 34(1-3): 107–130, 1999.

[GS96] Andrew R. Golding et Yves Schabes : Combining trigram-based and feature-based methods for context-sensitive spelling correction. *In Proceedings of the 34th conference on Association for Computational Linguistics*, pages 71–78. Association for Computational Linguistics, 1996.

[HB05] G. Hirst et A. Budanitsky : Correcting real-word spelling errors by restoring lexical cohesion. *Natural Language Engineering*, 11:87–111, 2005.

[JM97] Michael P. Jones et James H. Martin : Contextual spelling correction using latent semantic analysis. *In Proceedings of the fifth conference on Applied natural language processing*, pages 166–173. Morgan Kaufmann Publishers Inc., 1997.

[KCG90] Mark D. Kernighan, Kenneth W. Church et William A. Gale : A spelling correction program based on a noisy chan-

nel model. *In Proceedings of the 13th International Conference on Computational Linguistics*, pages 205–210, 1990.

[Kuk92] Karen KUKICH : Technique for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439, 1992.

[MB97] Lidia MANGU et Eric BRILL : Automatic rule acquisition for spelling correction. *In ICML '97 : Proceedings of the Fourteenth International Conference on Machine Learning*, pages 187–194. Morgan Kaufmann Publishers Inc., 1997.

[Mit87] Roger MITTON : Spelling checkers,spelling correctors and the misspellings of poor spellers. *Information Processing and Management*, 23(5):495–505, 1987.

[WF74] Robert A. WAGNER et Michael J. FISCHER : The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974.

[ZFA05] Chiraz Ben Othmane ZRIBI, Fériel Ben FRAJ et Mohamed Ben AHMED : Un système multi-agent pour la détection et la correction des erreurs cachées en langue arabe. *In TALN Récital 2005*, pages 143–152. ATALA, 2005.

[ZT99] Y. ZHAO et Klaus TRUEMPER : Effective spell checking by learning user behavior. *Applied Artificial Intelligence*, 13(8):725–742, 1999.

Quatrième partie

Annexes

Annexe A

Article pour RANLP 2005

Spelling Correction in Context

Guillaume Pinot Chantal Enguehard
Laboratoire d'Informatique de Nantes Atlantique (LINA)
Université de Nantes — 2, rue de la Houssinière — BP 92208
44322 Nantes Cedex 03 — FRANCE
http://www.sciences.univ-nantes.fr/lina/
guillaume.pinot@lina.univ-nantes.fr
chantal.enguehard@univ-nantes.fr

Abstract

Spelling checkers, frequently used nowadays, do not allow to correct real-word errors. Thus, the erroneous replacement of *dessert* by *desert* is not detected. We propose in this article an algorithm based on the examination of the context of words to correct this kind of spelling errors. This algorithm uses a training on a raw corpus.

A.1 Introduction

Spell checkers distributed with text processing such as MS Word or OpenOffice are based on the use of a dictionary. The text is analyzed word by word: each word which does not appear in the dictionary is supposed to be erroneous so corrections are proposed to the user. Paradoxically, the performances of these checkers in error detection are degraded with the increase in the size of the dictionary because they are unable to detect real-word errors.

These real-word errors occur when one or more modifications of a word transform it into another word which is present in the dictionary.

example : *This chocolate cake is a famous desert.*

The omission of an *s* in *dessert* reveals the word *desert*. This error is not detected because *desert* is present in the dictionary.

This problem was tackled during the second half of the 90's, in particular by Andrew R. GOLDING in [Gol95] and [GS96]. He defines confusing sets (like {desert , dessert} for example) and then determines by examining the text which of these words is the best candidate. This method was used in other papers like [JM97] and [MB97].

First, we will explain our algorithm and then, we will compare it with the method named *context word* by Andrew R. GOLDING [Gol95].

A.2 Simultaneous Detection and Correction

Our algorithm detects and corrects the errors simultaneously.

During the examination of a word m, the algorithm compares its probability of appearing in its context with the probability that another word m' appears in the same context, m' being close to m in the sense of an arbitrary distance.

The context of a word is defined by the set of the words present in a vicinity of fixed size in number of words. Considering that it is the semantic aspect of a word which will guide the correction, we make the assumption that the order of these words is not important.

The probabilities are collected during the training part.

We now present the two distinct parts of our algorithm: the computation of the contextual probabilities and the error detection/correction process.

A.3 Training

The training is made on a raw corpus. This algorithm is parameterized by k: the number of words around a word that constitute its context.

A.3.1 Reading the Corpus

The corpus is parsed word by word. Let w_c be the current word.

Constitution of the Dictionary

The goal is to index all the words appearing in the corpus with their frequency.

Let D be the dictionary, composed of a set of pairs $D_i = (w_i, c_i)$, w_i being a word. Each w_i is unique. c_i is the number of occurrences of the word w_i.

The constitution of the dictionary is processed as follows:

- if D_c exists, c_c is incremented.
- else, $D_c = (w_c, 1)$ is added to D.

Thus, we obtain the number of appearances of each word in the corpus. This information will allow to calculate various probabilities thereafter.

Context Dictionary C

Definition

The context dictionary named C gathers the co-occurrences of the words w_i and w_j, the distance between these words being lower or equal to k words. The word order is not taken into account. Each co-occurrence is supplied with its frequency $f_{i,j}$:

$$C = \{C_{i,j} \quad / \quad C_{i,j} = (\{w_i, w_j\}, f_{i,j})\}$$

Algorithm

The corpus is parsed word by word. During the treatment of a word w_c, the $C_{c,j}$ with $j \in [c-k, c-1]$ are calculated. They are the k co-occurrences generated while combining w_c with the words appearing in a window of width k preceding w_c (see figure A.1):

- if $C_{c,j}$ exists, $c_{c,j}$ is incremented.
- else, $C_{c,j} = (\{w_c, w_j\}, 1)$ is added to C.

We thus obtain all the 2-word sets located at a distance lower or equal to k, and their frequencies.

Complexity

Complexity in space is $O(nk)$ with n the size of the corpus in number of words. In practice, it should be lower because of the redundancy of words and co-occurrences.

Let $O(f(x))$ be the complexity of the search, with x the size of the database in which the search is processed. As the size of this base can be raised by nk, complexity in time is $O(nkf(n))$.

A.3.2 Calculating the Probabilities

We use the data gathered during the parsing of the corpus to calculate the probabilities of the contexts kept in C.

Let B be the dictionary of the pairs of words associated with their probability. We thus have:

$$B_{i,j} = ((w_i, w_j), P(w_i|w_j))$$

$B_{i,j}$ and $B_{j,i}$ are defined for each $C_{i,j}$. The probability is calculated as follows:

$$P(w_i|w_j) = \frac{c_{i,j}}{c_j}$$

A.4 Detection and Correction

A.4.1 Similarity Between Two Words

Let $\text{edist}(w_i, w_j)$ be a function comparing two strings and returning a positive number with the condition

$$\text{edist}(w_i, w_j) = 0 \Leftrightarrow w_i = w_j$$

The largest $\text{edist}(w_i, w_j)$ is, the most distant w_i is from w_j.

The Aspell [A+05] distance function takes in account the phonetic of words so it needs a linguistic knowledge and depends on the target language. In this first version, we choose to use the minimal edit distance [WF74] which is totally independent of the target language. However, we slightly modified this function to reduce the cost of the inversion of letters.

To determine if a word is a plausible correction of the word to be corrected, we use a $\text{sim}(w_i, w_j)$ function, which takes in arguments 2 words and returns true if the 2 words are similar and false if not.

Let ϵ be the empty string, we can define sim as in figure A.2.

In practice, we will take $\gamma = 8$ and $c = \text{edist}(\texttt{"a"}, \epsilon)$. These values have been determined after several experimentations.

A.4.2 Detection and Correction Algorithm

Let K_c be the context of a word w_c. K_c is the set of the $2k$ words located around w_c, that is to say the k words located before w_c and the k words located after w_c (see figure A.3).

Let w_c be the word to correct. For each $w_j \in K_c$, we constitute the set F_j such that

$$\begin{aligned} F_j = \{B_{i,j} \in B| \\ &\text{sim}(w_c, w_i) = \text{true}, \\ &P(w_i|w_j) > P(w_c|w_j)\} \end{aligned}$$

Let $k = 3$ and $c = 6$

We	can	have	a	lot	of	money.
w_1	w_2	w_3	w_4	w_5	w_6	w_7

w_3, w_4 and w_5 are in the context of w_6, that implies the sets $\{w_3, w_6\}$, $\{w_4, w_6\}$ and $\{w_5, w_6\}$.

Figure A.1: Example of a Context During a Corpus Parsing

$$\text{sim}(w_i, w_j) = \begin{cases} \text{true if } \text{edist}(w_i, w_j) \leq \frac{\text{edist}(w_i,\epsilon)+\text{edist}(w_j,\epsilon)}{k} + c \\ \text{false else} \end{cases}$$

Figure A.2: the $\text{sim}(w_i, w_j)$ definition

Let $k = 3$ and $c = 6$

We	can	see	that	this	is	a	very	good	dog.
w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}

We here have w_3, w_4, w_5, w_7, w_8 and w_9 in the context of w_6, that gives

$$K_6 = \{w_3, w_4, w_5, w_7, w_8, w_9\}$$

Figure A.3: Example of a Context During the Correction

Let $k = 3$ and $w_c =$ game

And	so	my	mind	game	round	to	the	business.
	0.0308	0.0120		came	0.0731	0.0162	0.0157	
				same			0.0052	
				gate		0.0032	0.0030	
		0.0096		gave		0.0032	0.0017	
	0.0061			name			0.0008	
				game			0.0004	

The numbers are $P(w_i|w_j)$. For example $P(\text{round}|\text{came}) = 0.07317$

$$G_1 = \{((\text{came}, \text{so}), 0.03086), ((\text{came}, \text{my}), 0.01207), ((\text{came}, \text{round}), 0.07317), ((\text{came}, \text{to}), 0.01620), ((\text{came}, \text{the}), 0.01571)\}$$

We do not detail $G_2, \ldots, G_6$ which are built in the same way. The heuristic gives the highest score to the first proposition *came* because it appears 5 times.

For this example, we trained our algorithm on the novel *The War of the Worlds* by H. G. Wells. The sentence came from *The Time Machine* by H. G. Wells (Chapter 3) with the error added.

Figure A.4: Propositions

We thus obtain $2k$ sets of propositions. The set of the possible propositions, F, is the union of the sets F_j (see example figure A.4).

We now need a heuristic to give a score to each proposition in order to have them in a pertinent order. We here propose a first heuristic, but we are also elaborating tests to refine it.

Let G_j be the subset of F such that

$$G_j = \{B_{i,j}, B_{i,j} \in F\}$$

One can then define the following H_j heuristic:

$$H_j = |G_j| + \prod_{B_{i,j} \in G_j} P(w_i|w_j)$$

This heuristic favors the propositions appearing in several sets of proposals F_j, then the strongest probabilities (see example in figure A.4).

A.5 Comparison with Other Works

This algorithm is close to the *Context Words Method* by Andrew R. Golding [Gol95].

A.5.1 The Context Words Method

In [Gol95] like in other articles based on it ([JM97], [GS96] and [MB97]), confusion sets are used to correct real-word errors. A confusion set is a set of words which can be confused among each other, because of their close spellings ($\{\text{dessert}, \text{desert}\}$) or because they are often confused ($\{\text{between}, \text{among}\}$).

During the training, a set of $(w_c, w_i, P(w_c|w_i))$ is created (with w_c a word belonging to at least one confusion set, w_i any word). To correct a word, the probabilities of all the words of the corresponding confusion sets are computed, the highest probability being proposed in correction.

A.5.2 Comparison of the Two Methods

These two methods are based on the same idea: using the words present around the word to correct. They differ especially in the way of establishing sets and also in the nature of them:

- Golding supposes that its confusion sets are preestablished.

  We automatically determine words which can be confused during the correction process. This selection is based on the corpus itself and on our similarity function.

$$\text{Precision on detection} = \frac{\text{Number of words rightly detected as being erroneous}}{\text{Number of words detected as being erroneous}}$$

$$\text{Recall on detection} = \frac{\text{Number of words rightly detected as being erroneous}}{\text{Number of words rightly detected}}$$

$$\text{Precision on correction} = \frac{\text{Number of correctly corrected words}}{\text{Number of words rightly detected as being erroneous}}$$

Figure A.5: Formulas of the precision and the recall

- The Golding's confusion sets are disjoint (their intersections are empty).

  This is not the case in our method: for each word w_i, we determine automatically a list of words that are similar to w_i and which occur in the same context of the words cooccurring with w_i. The list established for w_i and the list established for w_j ($w_i \neq w_j$) can encounter some words in commom.

A.6 Experimentations

A.6.1 Corpus

We would have liked to experiment our method on the same corpus as Golding in order to compare fruitfully our results with those he has obtained. Unfortunately, the Brown corpus used by Golding is not free, so we could not perform our algorithm on it.

So, our experiments on this spelling checker use the novel *les Misérables* by Victor HUGO. This corpus is divided into two parts: the training part (480588 words) and the part to be corrected (53405 words) in which errors have been added.

A.6.2 Experimental Method

We performe the correction and then generate the precision and the recall on the detection of error as well as the precision on correction (see figure A.5 for the formulas).

A.6.3 Adding Errors

Real-word errors are previously added automatically without using any external resource.

This introduction is done in two steps: first the generation of the possible errors, then the introduction of these errors in the text.

Generation of the Possible Errors

Let D be a simple dictionary (a set of word). For each word $w_i \in D$, we associate a set of words included in D and close to w_i (in the sense of our similarity function). This method generates a base which can be used to generate real-word errors.

In practice, we use as dictionary the words appearing more than ten times in *les Misérables* to select errors using words whose context is known by our corrector. Errors on low frequency words (like apax) could not be detected in such experimentations because their context is completely unknown.

Adding Real-word Errors

Two parameters control the introduction of real-word errors: the density of inserted errors and the previously determined possible errors.

Errors are located using a XML tag which keeps the original word (this is the correction we wish to find).

Example: we introduce the word “game” instead of “came” in the text “And so my mind came round to the business.”:

```
And so my mind
<error correction="came">game</error>
round to the business.
```

Each word of the corpus is affected or not by an error according to the probability of error fixed by the wished density.

A.6.4 Results

A summary of the results is given in the table A.1.

Density	Precision	Recall	P. on correction
10%	0.1081926	0.9363030	0.9622054
1%	0.0206164	0.9615384	0.9500000

Table A.1: Results

We note that the precision on detection is very bad. This overdetection of our algorithm is problematic.

On the other hand, we obtain a very good recall on detection and the precision of the correction is more than 95%. The correction in itself is thus very efficient.

A.7 Conclusion

We have presented here an algorithm that uses non-ordered contexts to detect and correct real-word errors.

The advantages of this algorithm are:

- simplicity;
- independence from any linguistic information;
- use of a raw corpus for the training;
- few parameters have to be regulated.

The disadvantages are:

- the significant size of the data generated by the training.
- the low precision on detection: the algorithm proposes corrections for a lot of correct words.

Our algorithm is intended to be used during the interactive correction of a text so its speed should be sufficient. On the other hand, the overdetection of errors constitutes a real problem.

We thus direct our research towards the definition of better heuristics of scheduling of the propositions. The size of the training corpus may also influence the quality of the results, its influence should be observed. We also plan to define ordered contexts to use the syntax in addition to semantics.

These various methods will be precisely evaluated on the same corpus to analyze their relevance.