Modèle de langue basé sur l'analyse distributionnelle

William LÉCHELLE

24 août 2012

1 Introduction

J'ai réalisé ce ("second") stage de M1 au RALI [1] (Recherche appliquée en linguistique informatique) — laboratoire de l'Université de Montréal — sous la direction de Philippe Langlais. Le laboratoire suit 3 axes de recherche, qui correspondent à peu près aux spécialités des 3 professeurs qui le constituent : la recherche d'informations, le résumé automatique, et la traduction automatique. Mon stage ne s'inscrit pas vraiment dans ces thématiques, mais plutôt dans le cadre général du traitement automatique des langues : des modèles de langues sont embarqués dans toutes les applications qui ont à évaluer la qualité d'un texte, qu'il soit reconnu ou produit automatiquement.

1.1 Modèles de langue

En traitement automatique des langues, on a fréquemment besoin de savoir évaluer la correction d'une phrase donnée. Un modèle de langue répond à cette question en évaluant une **probabilité** pour une phrase donnée (implicitement : sa fréquence parmi des tirages aléatoires de phrases correctes dans la langue en question). Pour une phrase de m mots,

$$p(w_1 \dots w_m) = \prod_{i=1}^{m} p(w_i | w_1 \dots w_{i-1}).$$

1. http ://rali.iro.umontreal.ca/rali/

On peut faire l'hypothèse que la plausibilité du i^e mot ne dépend que des quelques k qui le précèdent, et la formule devient

$$p(w_1 \dots w_m) = \prod_{i=1}^{m} p(w_i | w_{i-k} \dots w_{i-1}).$$

Un **modèle de langue n-gramme** est ce qui permet d'évaluer, d'après un **historique** (ou contexte) de $n-1$ mots $w_1 \dots w_{n-1}$, la probabilité $p(w_n | w_1 \dots w_{n-1})$ du prochain mot.

Pour obtenir un modèle de langue, on commence par compter les occurences de chaque séquence de n mots dans un grand corpus (dit d'entrainement), on **lisse** ces statistiques (pour éviter de rejeter totalement ($p = 0$) des exemples qu'on a eu le malheur de ne pas rencontrer dans le corpus d'entrainement)(et là se trouve toute la difficulté de l'exercice), et enfin, on a la simple formule de l'estimation par **maximum de vraisemblance**

$$(w_n | w_1 \dots w_{n-1}) = \frac{|w_1 \dots w_n|}{|w_1 \dots w_{n-1}|}.$$

Les modèles (intéressants) les plus simples utilisent un seul mot d'historique ($n = 2$), et c'est le cas qui va nous intéresser ici. Les performances augmentent (de plus en plus lentement) jusqu'environ \$n=5\$[4], mais les coûts de calcul (en espace pour stocker le modèle, ou en temps s'il est davantage compressé[3]) eux augmentent énormément.

1.2 Lissage

De nombreuses méthodes de lissage existent, utilisant additions aux et normalisations des comptes existants :

- Good-Turing : un n-gramme vu r fois est prétendu avoir été vu $(r+1)\frac{n_{r+1}}{n_r}$ fois ;
- Jelinek-Mercer ;
- Knesser-Ney...

Une technique intéressante consiste à utiliser les comptes d'historiques un peu plus courts si l'historique complet n'a jamais été vu, en oubliant les derniers mots. Par exemple, si on veut estimer la probabilité p("mort"|"le petit chat est"), mais que « le petit chat est mort » n'a jamais été vu à l'entrainement, on peut se rabattre sur « chat est mort », puis

sur « est mort », dont les statistiques sont probablement mieux estimées (mais moins précises par rapport au problème). Cela permet de former des modèles dits *back-off*, qui pondèrent (savamment) l'avis de modèles d'ordre inférieur. Les méthodes de Jelinek-Mercer et de Knesser-Ney, par exemple, visent à déterminer ces coefficients de manière optimale.

Mon travail a consisté en le développement d'une autre méthode pour pallier aux défauts des comptes obtenus à l'entrainement (qui concernent principalement les mots les moins fréquents), qui utilise la proximité sémantique pour déduire des estimations plus fiables des comptes de mots rares que celles observées à l'entrainement, à partir de comptes de mots voisins, mais plus courants (donc dont l'observation est déjà fiable).

2 État de l'art

En 1993, Dagan et al.[1] introduisirent une nouvelle technique de "lissage", ou plutôt, une autre méthode pour obtenir la fréquence attendue d'une paire de mots lorsque ceux-ci n'ont pas été (ou suffisament) vus lors de l'entrainement du modèle. Cela correspond au cas de modèles bigrammes, où il s'agit de prédire le mot suivant l'unique mot d'historique connu (à chaque étape de l'évaluation de la probabilité de la phrase).

2.1 L'idée...

Cette approche utilise le fait que des mots similaires arrivent dans les mêmes contextes : sans (bonnes) connaissances sur la probabilité d'un mot dans un contexte, on peut utiliser les probabilités d'occurence des mots qui lui sont similaires. Il faut bien sûr tenir compte des différences de fréquences d'apparition des mots, donc multiplier par le rapport des deux fréquences. Par exemple, si *chien* a la probabilité p d'arriver dans le contexte h, *canidé* aura la probabilité $p * \frac{|\text{canidé}|}{|\text{chien}|}$ d'arriver après le même contexte.

2.2 ... et le calcul

Plus précisément, des bigrammes similaires ont des valeurs d'**information mutuelle** similaires. L'information mutuelle est définie par :

$$I(w_1, w_2) = \log_2 \frac{p(w_1, w_2)}{p(w_1)p(w_2)} = \log_2 \frac{p(w_1|w_2)}{p(w_1)},$$

où $p(x)$ désigne la probabilité de l'occurence du mot x. Cette mesure quantifie l'association des deux mots : plus elle est élevée, et plus les mots sont fréquemment rencontrés ensemble plutôt que séparément (les mesures négatives sont ramenées à zéro et ignorées).

L'information mutuelle d'un bigramme, par cette méthode, est alors la moyenne de celles (non-nulles) des bigrammes qui lui sont le plus proches. La **fréquence estimée** de cette paire de mots est alors

$$|w_1, w_2|_{est} = \frac{1}{N}|w_1||w_2| * 2^{I(w_1,w_2)}$$

(où N est la taille du corpus). Cette valeur est utilisée plutôt que la valeur vue à l'entrainement (typiquement, un 0 non significatif) pour calculer la probabilité du bigramme (en normalisant par rapport à la somme des fréquences estimées).

Reste à définir la relation de similarité dont il est question.

2.3 Quelle similarité ?

On peut considérer que deux paires de mots sont similaires si chacun des mots qui les composent sont respectivement similaires, mais pour simplifier, et surtout réduire le bruit dans les données, on ne considère que des paires où un seul mot est remplacé par un mot proche, et où l'autre reste identique. Au niveau des mots, ce qui est nécéssaire, c'est que **deux mots soient semblables s'ils apparaissent dans les mêmes contextes**.

N'ayant pas utilisé la métrique de l'article, je ne la mentionne que brièvement :

– 2 mots w_1 et w_2 sont similaires à un troisième w si

$$\text{sim}(w_1, w_2, w) = \frac{\min(I(w, w_1), I(w, w_2))}{\max(I(w, w_1), I(w, w_2))}$$

est élevée (en réalité, c'est une similarité "à gauche", mais je le néglige dans cette explication) ;
– On pondère ces valeurs pour l'ensemble du lexique : l'information mutuelle maximale utilisée dans le calcul est un bon poids, car cela accentue l'importance des mots effectivement proches de w_1 et w_2.
– On a alors

$$\begin{aligned} \text{sim}(w_1, w_2) = & \frac{\sum_w \text{sim}(w_1, w_2, w) W_{(w_1, w_2, w)}}{\sum_w W_{(w_1, w_2, w)}} \\ = & \frac{\sum_w \min(I(w, w_1), I(w, w_2))}{\sum_w \max(I(w, w_1), I(w, w_2))}, \end{aligned}$$

pour w parcourant le vocabulaire connu du modèle.
– Pour éviter une complexité en $O(n^3)$, on définit un ensemble de "voisins forts" pour chaque mot (donc l'information mutuelle avec ce mot est et significative et élevée), et on ne calcule la similarité qu'entre deux mots qui partagent assez de voisins forts.

3 Outils utilisés

3.1 Implémentation de modèles de langue : SRILM

SRILM [2] est un toolkit pour construire et utiliser des modèles de langues statistiques, développé depuis 1995 par le *SRI International's Speech Technology and Research (STAR) Laboratory*. Cette bibliothèque écrite en C++ implémente plusieurs méthodes de lissage courantes, et me permet a la fois d'obtenir des comptes de bigrammes sur mon corpus, et de comparer mon modèle a une référence bien établie. Par contre, je n'ai pas réussi à en obtenir des étapes intermédiaires de calcul lors de l'utilisation de modèles de langues, ce qui m'aurait simplifié une bonne partie de mon travail. Des interfaces aux modèles de langues existent, mais j'ai développé mon code en python, alors que ces interfaces sont écrites principalement en C ou en Java, donc il aurait été trop long de les utiliser en devant les réinterfacer.

2. http ://www.speech.sri.com/projects/srilm/

3.2 Thésaurus : FreDist

FreDist[2] est un thésaurus pour le français, développé [3] en python et en 2011 par l'équipe ALPAGE de L'INRIA. Il dresse, pour 4 types de mots (noms, verbes, adjectifs, et adverbes), la liste des 100 mots les plus "proches" de chaque mot qu'il connait, obtenue automatiquement sur la base d'un grand corpus (450 millions de mots, partagés entre Wikipédia et la presse française). La relation de similarité qu'ils extraient étant celle de l'analyse distributionnelle (quels mots apparaissent dans quels contextes), elle est tout à fait compatible avec l'information mutuelle. J'ai donc pu employer les premiers mots de chaque entrée du thésaurus comme "mots les plus similaires" pour appliquer la méthode ci-dessus sans devoir mesurer moi-même leur similarité, ni chercher les voisins forts. Par contre, j'ai du (car appliquer un étiquetteur syntaxique est assez long computationellement) réunifier les 4 types de mots qu'ils séparaient, écrasant au passage les entrées des mots appartenant à plusieurs classes de mots en même temps. Je pense que ça n'impacte que négligeablement les performances.

J'aurais souhaité — mais je n'ai pas eu le temps de le faire lors du stage — mesurer la qualité de leurs suggestions, par exemple en appliquant la métrique de l'article à un petit échantillon, et en cherchant les voisins forts partagés ; ou bien en redéveloppant (automatiquement) un nouveau thésaurus avec les outils qu'ils mettent à disposition [4] (en changeant la mesure de similarité, ou bien en utilisant un autre corpus pour l'entrainement du thésaurus, j'aurais peut-être pu obtenir de meilleurs résultats).

3.3 Tokenizeur : NLTK

NLTK [5] (Natural Language ToolKit) est une bibliothèque Python open-source, dont le projet est mené par Steven Bird, Edward Loper et Ewan Klein. Comme son nom l'indique, NLTK contient un ensemble d'outils qui permettent de travailler sur du texte, de larges corpus, et embarque également certains algorithmes d'apprentissage machine des plus courants. Je m'en suis servi pour tokenizer les données extraites de wikipédia. Malheureusement, il semblerait que la qualité du résultat laisse à désirer, notamment au niveau des caractères accentués, souvent mis de côté comme des mots à part.

3. http ://fredist.gforge.inria.fr/
4. https ://gforge.inria.fr/frs/ ?group$_{\text{id}}$=2570
5. http ://nltk.org/

4 Implémentation

4.1 Corpus employé

J'ai utilisé des articles de Wikipédia (disponibles dans les ressources du laboratoire, au format XML), auxquels j'ai enlevé la plupart de la structure propre à l'encyclopédie (les titres de sections les plus fréquents, par exemple), et tous les éléments spéciaux (images, hyperliens, certaines listes...). Cette source a l'avantage de pouvoir fournir de grandes quantités de texte de bonne qualité, mais l'inconvénient qu'il faille ensuite filtrer le XML pour obtenir le simple contenu textuel des articles. Après un long filtrage (à l'aide d'outils unix en ligne de commande, `grep`, `sed`, `tr`, `uniq` , etc.), restent 126 millions de mots.

La plus grande partie de ce corpus sert à l'entrainement du modèle, et une petite partie (1%, pour 1,3 millions de mots) sert à l'évaluation du modèle (comme il est d'usage en apprentissage machine : entrainer le modèle sur le texte sur lequel il sera évalué biaise les résultats).

Pour traiter de telles quantités de texte, j'ai dû utiliser le cluster du RALI.

4.2 Vocabulaire

Pour s'assurer que les mots inconnus soient les mêmes pour tous les estimateurs, j'ai utilisé SRILM pour remplacer tous les mots (tokens) du corpus de test non présents dans le corpus d'entrainement par le même token OOV (Out Of Vocabulary) dans les fichiers de comptes, token qui est alors traité normalement (et dont les statistiques sont donc la somme de celles de tous les mots inconnus). C'est une technique courante (pour ne pas faire cas des mots sur lesquels on n'a de toutes façons pas d'information), et le modèle de référence bénéficie de la même simplification.

Il aurait été intéressant de pouvoir limiter le vocabulaire à celui pour lequel le thésaurus connaît les mots similaires, mais celui-ci ne représentant que 23% du total, cela ne paraît pas envisageable. Heureusement, la plupart des mots inconnus du thésaurus semblent être des déterminants et de la ponctuation, et quelques erreurs de tokenisation. La plupart de ces tokens sont souvent vus à l'entrainement, et leur fréquence observée est correcte, donc cela ne pose pas de problème. (La notion de "mot similaire" s'applique mal à ces mots-là, qui plus est.)

4.3 Fréquences et normalisation

J'ai utilisé les scripts de SRILM pour établir des fichiers de comptes de bigrammes à partir de mes corpus de test et d'entrainement. Pour tous les bigrammes de test, on fait appel au thésaurus pour savoir à quels bigrammes connus se référer pour faire la moyenne de leurs informations mutuelles, et obtenir une fréquence estimée pour le bigramme de test. Les informations mutuelles, et fréquences des mots, sont obtenus à partir du corpus d'entrainement. Comme détaillé dans la suite, il est nécéssaire de normaliser les fréquences estimées pour pouvoir en tirer des probabilités exploitables.

Longtemps, j'ai cru que je pourrai n'appliquer cette méthode qu'aux seuls bigrammes pour lesquels les informations obtenues à l'entrainement étaient insatisfaisantes (vus trop peu de fois, typiquement, jamais, par simple (et inévitable) incomplétude du corpus) ; et pour les autres, utiliser leur fréquence telle qu'observée à l'entrainement (environ comme le modèle de référence, qui lisse ces statistiques). Malheureusement, pour normaliser les fréquences estimées des bigrammes non-vus à l'entrainement, il faut non seulement retirer une masse de probabilité à ceux vus à l'entrainement (et "combien ?" est une première question), mais surtout, il faut que **pour chaque mot d'historique**, la somme des fréquences observées et des fréquences estimées somme à 1. Autrement dit, il aurait fallu faire un calcul de normalisation par mot du lexique, ce qui serait immensément trop long à réaliser.

J'ai donc finalement décidé d'appliquer cette méthode à tous les bigrammes de test, en incluant le fait qu'un mot est similaire à lui-même, et donc que si un bigramme est correctement représenté dans les données d'entrainement, l'observation de sa propre information mutuelle comptera fort dans la moyenne pour déterminer une estimation d'icelle (l'estimation étant meilleure que l'observation, dans l'esprit de la méthode).

5 Évaluation

5.1 Perplexité

La mesure la plus couramment utilisée pour évaluer les modèles de langues est leur **perplexité** (sur un corpus de test). Issue de la théorie de l'information, cette mesure quantifie le "nombre de choix" qu'a le modèle à chaque fois qu'on l'interroge, en moyenne. Plus ce nombre est bas,

plus le texte est prévisible pour le modèle de langue (car plus la probabilité qu'il a attribué à chacun des mots était élevée). Mathématiquement, avec le corpus de test de n phrases $\mathscr{S} = \{s_1, \ldots, s_n\}$ (N mots au total), où $s_i = \{w_1^i, \ldots, w_{n_{s_i}}^i\}$ est une phrase de n_{s_i} mots, l'entropie (par mot) de test est :

$$H = -\frac{1}{\sum_{i=1}^{n} n_{s_i}} \log(\prod_{i=1}^{n} p(s_i)) \quad = -\frac{1}{N} \sum_{i=1}^{n} \sum_{j=1}^{n_{s_i}} \log p(w_j^i | w_{j-1}^i).$$

La perplexité du modèle est alors :

$$PP = 2^H = p(s_1, \ldots, s_n)^{-\frac{1}{N}}.$$

Ce nombre est typiquement compris entre 70 et 400.

La grande difficulté que j'ai rencontré à utiliser cette mesure est la nécéssaire normalisation des "scores" (en l'occurence, les fréquences estimées), à laquelle il est nécéssaire de procéder pour pouvoir les interpréter en temps que probabilités (qui somment à 1). Lorsque les probabilités des bigrammes sont simplement calculées par maximum de vraisemblance (rappel : $p(w_j|w_{j-1}) = |w_{j-1}w_j|/|w_{j-1}|$), la somme des comptes nous assure que ces fréquences sont bien (interprétables comme) des probabilités, mais dès lors que j'**estime** les fréquences des bigrammes qui n'ont pas été vus à l'entrainement, il n'y a plus aucun repère pour s'assurer que ces fréquences somment bien à 1.

Pour normaliser ces fréquences estimées (une fois établi que les fréquences de tous les bigrammes de test seraient estimées), j'ai pris un échantillon aléatoire de bigrammes dans le vocabulaire du modèle, calculé leurs fréquences estimées, et extrapolé à l'ensemble du lexique. Cela peut malheureusement introduire beaucoup de variance dans les résultats, mais il n'est pas envisageable de calculer la fréquence estimée du vocabulaire au carré.

5.2 Ranking

Une autre possibilité pour comparer les performances des modèles de langue est de leur soumettre une tâche de *ranking* : on demande au modèle, pour chaque contexte donné, les mots qu'il estime être les plus probables (d'après des scores, mais sans probabilités), et on observe les rangs qu'il attribue aux véritables mots suivants (dans un corpus de test). Cela évite d'avoir à normaliser, et permet plus facilement d'observer sur quels mots, ou quels contextes, le modèle fait les erreurs les plus importantes.

Malheureusement, je n'ai pas eu le temps d'implémenter cette approche.

5.3 Modèle de référence

Comme référence, j'ai utilisé le modèle bigramme standard de SRILM, qui utilise la méthode de lissage de Knesser-Ney. C'est un modèle *back-off* qui pondère systématiquement les avis de tous les modèles d'ordre inférieur, avec des coefficients qui dépendent fortement du contexte, et qui sont appris par un algorithme d'apprentissage, en fonction des fréquences des historiques auxquels ils correspondent (bien sûr, pour un modèle bigramme, tout cela est très limité, mais les scores sont quand même lissés).

J'obtiens une perplexité de 234,24.

5.4 Résultats

Malheureusement, les résultats ne sont pas au rendez-vous, et j'obtiens une perplexité moyenne (qui varie avec l'échantillon choisi pour normaliser les fréquences estimées) un peu inférieure à 10^5. Il y a sûrement encore une énorme marge de progrès à réaliser en améliorant les conditions de l'expérience, par exemple en mesurant l'importance de la partie du vocabulaire que le thésaurus ne connait pas, ou en s'assurant que la tokenization fait très peu d'erreurs, mais je n'ai pas eu l'occasion d'aller très loin dans cette direction.

6 Perspectives et conclusion

J'ai implémenté la méthode de Dagan et al. [1] pour estimer l'information mutuelle et la fréquence attendue de bigrammes non-vus lors de l'entrainement d'un modèle de langue, sur un grand corpus tiré de Wikipédia, traité avec les outils de la ligne de commande unix, et le tokeniseur de NLTK. J'ai également utilisé les scripts de la bibliothèque SRILM de modèles de langues pour construire mon modèle, et le thésaurus FreDist [2] comme mesure de similarité sémantique. Les résultats ne sont pas encore au point, probablement plus à cause des conditions de l'expérience que de la qualité de la méthode.

Une fois au point, une piste intéressante serait de combiner ce nouveau modèle avec un autre plus standard, et de n'utiliser la moyenne de l'information mutuelle des paires semblables que pour les mots rares dont les mots proches ont des statistiques plus fiables (plutôt que pour tous les mots, comme c'est le cas actuellement). En accordant une masse de probabilité à

chaque modèle, suivant les bigrammes qu'il sait estimer avec le plus de précision, on peut souvent obtenir de meilleurs résultats. Enrichir le thésaurus serait une autre piste pour améliorer les performances de ce modèle.

Enfin, une des principales limites du modèle est l'utilisation des seuls bigrammes. Arriver à généraliser cette méthode, ne serait-ce qu'à des trigrammes, serait très prometteur, bien que cela nécéssite probablement des quantités de calcul beaucoup plus importantes, à la fois pour étendre la notion d'information mutuelle, et pour chercher les voisins sémantiques compatibles entre eux.

Références

[1] Ido Dagan, Shaul Marcus, and Shaul Markovitch. Contextual word similarity and estimation from sparse data. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, ACL '93, pages 164–171, Stroudsburg, PA, USA, 1993. Association for Computational Linguistics.

[2] Enrique Henestroza Anguiano and Pascal Denis. FreDist : Automatic construction of distributional thesauri for French. In *Actes de la 18ème conférence sur le traitement automatique des langues naturelles*, pages 119–124, Montpellier, France, France, 2011.

[3] Adam Pauls and Dan Klein. Faster and smaller n-gram language models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics : Human Language Technologies - Volume 1*, HLT '11, pages 258–267, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[4] R. Rosenfeld. Two decades of statistical language modeling : where do we go from here ? *Proceedings of the IEEE*, 88(8) :1270 –1278, aug. 2000.