Université de Montréal

IFT6285 (TALN) — Devoir3
Correction de mots

Contact :
Philippe Langlais
RALI/DIRO
Université de Montréal
+1 514 343 61 11 ext: 47494
felipe@iro.umontreal.ca
http://www.iro.umontreal.ca/~felipe/

dernière compilation : 24 septembre 2020 (16:28)

Contexte

La correction orthographique est un domaine de recherche qui a été plutôt développé dans un context industriel. Dans ce devoir nous allons aborder la correction de mots comme un problème d'identification des voisins d'un mot erroné, le candidat à la correction étant sélectionné dans ce voisinage. Une excellente source d'inspiration est le blog de Peter Norvig sur le sujet. Vous pouvez si vous le souhaitez reprendre tout ou partie de son code, ou l'implémentation pyspellchecker.

Données

Vous avez accès à une liste de 3935 paires de mots (mot erroné / correction, séparés par une tabulation) avec laquelle vous pouvez développer votre devoir : devoir3-train.txt. Cette liste provient du corpus toefl-spell (Flor et al. 2019).

Vous disposez également du lexique voc-1bwc.txt des 201 315 mots rencontrés au moins 16 fois dans la partie d'entraînement du corpus 1Bwc (Chelba et al. 2013), ayant une longueur comprise entre 3 et 29 caractères, ne contenant aucun chiffre, contenant au moins une lettre et ne terminant pas par un point. [1] Chaque mot est accompagné de sa fréquence dans 1Bwc (un espace sépare la fréquence du mot).

À faire

1. Vous devez écrire un programme `corrige` [2] qui lit sur l'entrée standard (`stdin`) une liste de mots à corriger (un par ligne) et qui affiche pour chacun d'eux sur la sortie standard (`stdout`) les corrections identifiées en ordre décroissant de vraisemblance. Votre programme prendra comme seul argument obligatoire un lexique de mots parmi

1. Plus précisément, la commande suivante a été lancée : cat $in/news* | tr ' ' '\n' | tr '[:upper :]' '[:lower :]' | sort | uniq -c | sort -k1,1n | awk '$1 >15 && length($2) > 2 && length($2) < 30 && $2 ! /[[:digit :]]/ && $2 ~ /[a-z]/ {print $0}' | grep -v '\-\-' | grep -v '\.$' où $in vaut le répertoire de 1Bwc installé au DIRO (voir devoir 1).
2. Votre programme peut s'appeler `corrige.py` ou `corrige.cpp`, etc.

lesquels rechercher la correction ; chaque mot est accompagné de sa fréquence (calculée dans un grand corpus). Vous pouvez doter votre programme d'options permettant de contrôler le comportement de votre correcteur. Le format de sortie de votre programme est prescrit par l'exemple qui suit où pour chaque mot à corriger, votre programme doit afficher une ligne qui contient dans cet ordre et séparés par une tabulation le mot à corriger, la correction la plus vraisemblable et d'éventuelles corrections supplémentaires que vous classerez en ordre décroissant de vraisemblance. Si aucune correction n'est identifiée, mettez le mot à corriger en guise de correction.

head -n 100 devoir3-train.txt | awk '{print $1}' | \
corrige voc-1bwc.txt > devoir3-sortie [3]

L'approche qui sera développée dans votre programme consiste à chercher pour un mot à corriger un ensemble de voisins au sens d'une distance particulière (vous en considérerez au moins trois) et à classer ces corrections soit directement à l'aide de la distance, soit à l'aide du modèle unigramme que constitue le lexique passé en argument au programme, soit avec les deux. Votre but est d'étudier l'impact des distances utilisées pour calculer les voisins et de décider d'une meilleure approche au problème de correction. Votre meilleure approche peut combiner plusieurs distances. Votre programme devra par défaut employer votre meilleure approche : L'invocation de votre programme sans option doit donc correspondre à ce que vous pensez être la meilleure façon de corriger la liste de mots passée en entrée pour un lexique donné en argument.

Note : vous ne pouvez pas faire d'hypothèse particulière sur la nature du lexique fourni au programme `corrige` autre que son format (lignes de fréquence et mot séparés par un espace).

2. Produire un rapport (format pdf, en français ou en anglais) d'**au plus deux pages** qui compare/combine au moins 3 distances vues en cours pour résoudre la recherche de voisins d'un mot à corriger. Votre rapport doit identifier :
   — les codes/libraries que vous ré-utilisez le cas échéant,

3. La sortie est produite par un programme qui utilise le soundex de manière naïve.

— votre mesure de la qualité de votre correcteur, le découpage eventuel du corpus distribué en train/dev et test,
— l'impact des distances étudiées sur les temps de correction, sur les performances,
— une analyse de votre meilleure approche et de ses limites.

Remise

La remise est à faire sur Studium sous le libellé `devoir3`. Vous devez remettre votre code et votre rapport (format pdf, texte en anglais ou en français) dans une archive (gzip, tar, tar.gz) dont le nom est préfixé de `devoir3-name1` ou `devoir3-name1-name2` selon que vous remettez seul ou a deux, où `name1` et `name2` sont à remplacer par l'identité des personnes faisant la remise (`prénom_nom`). Donc si j'avais à remettre seul mon solutionnaire au devoir3, je le ferais sous le nom `devoir3-philippe_langlais.tar.gz`. Assurez vous que le nom des personnes impliquées dans le devoir soit indiqué sur tous les documents remis (code et rapport). Le devoir est à remettre en groupe d'au plus deux personnes au plus tard vendredi 2 octobre à 23h59.
Note : Aucune donnée ni modèle n'est demandé : juste le rapport et le code.

Chelba, Ciprian et al. (2013). "One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling". In : *CoRR* abs/1312.3005. URL : http://arxiv.org/abs/1312.3005.

Flor, Michael et al. (2019). "A Benchmark Corpus of English Misspellings and a Minimally-supervised Model for Spelling Correction". In : *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*. Florence, Italy : Association for Computational Linguistics, pages 76–86. DOI : 10.18653/v1/W19-4407. URL : https://www.aclweb.org/anthology/W19-4407.