TP 1 - TRAITEMENT DE CORPUS ET LOI DE ZIPF

LANGAGE NATUREL

Objectif

L'objectif de ce TP est de se familiariser avec le traitement de corpus. Nous travaillerons sur le corpus des TED talks. Il s'agit d'un texte suffisamment grand, homogène et libre de droits, téléchargeable gratuitement en plusieurs langues.Tous les exercices demandent l'utilisation du terminal de ligne de commande Linux. Il est conseillé de noter toutes les commandes utilisées dans un fichier texte ou sur papier pour pouvoir répéter des étapes facilement en cas d'erreurs détectées tardivement. N'hésitez pas à consulter la page `man` des commandes ou chercher sur internet.

1. Prétraitements

1.1. **Rappel des commandes Unix.** Rappelons-nous de quelques commandes Unix utiles pour le traitement des fichiers textuels. Nous allons utiliser le fichier `ted.txt` contenu dans `cours-corpus.zip`.

(1) Naviguez jusqu'au dossier `cours-corpus` et affichez le fichier `ted.txt` avec `less` ou `cat`. Comptez ne nombre de lignes, mots et caractères avec `wc`. Affichez les 10 premières et dernières lignes avec `head` et `tail`.

(2) Est-ce que le mot "oiseau" apparaît dans le corpus ? Pour le savoir, on utilise la commande `grep -E '␣oiseau␣' ted.txt`. [1] Combien de lignes contiennent ce mot ?

(3) Utilisez `grep` avec une *expression régulière* pour chercher tous les mots qui finissent par le suffixe *-ment*. Combien de lignes sont retournées ? Utilisez l'option `-o` de `grep` combinée avec `sort` et `uniq` pour découvrir combien de mots **différents** correspondent à cette recherche.

(4) Utilisez la commande `sed` pour remplacer tous les espaces par des tirets dans le corpus. Cette commande prend la forme `sed 's/REGEXP/REMPLACEMENT/g'`, où `REGEXP` est une expression régulière et `REMPLACEMENT` est une chaîne de caractères qui remplacera les suites qui correspondent à `REGEXP` dans le fichier. Par exemple, `sed 's/a/e/g'` remplace toutes les lettres 'a' par des 'e'.

1.2. **Segmentation, tokenisation et casse.** Pour les outils de TAL, il est utile d'avoir composé d'une suite de phrases séparées par retour chariot (`\n`). Les phrases sont des suites de mots séparés par des espaces. Il faut aussi homogénéiser la casse au début des phrases pour éviter que des mots soient considérés différents selon leur position dans la phrase.

(1) Avec `sed`, insérez des retours à la ligne après chaque point final ".". Combien de phrases ? Améliorez cette segmentation afin de prendre en compte d'autres séparateurs de phrase sans sur-segmenter, par exemple, les URLs et les abbréviations suivies d'un point ?

(2) Utilisez maintenant le segmenteur de phrases fourni à l'aide de la commande ci-dessous. Combien de phrases ? Comparez avec la sortie de la question précédente. Quelles sont les différences ?

```
bin/split-sentences.perl -l fr < ted.txt | sed -E '/<P>/d' > ted-sent.txt
```

(3) Les signes de ponctuation simples sont attachés aux mots précédents. Utilisez `sed -r 's/([^␣]),␣/\1␣,␣/g'` pour séparer les virgules. Combien de mots dans le corpus avant et après la modification ? Séparez les points, parenthèses, etc. des mots adjacents.

(4) Utilisez maintenant le tokeniseur fourni à l'aide de la commande ci-dessous. Combien de mots ? Comparez avec la sortie de la question précédente. Quelles sont les différences ?

```
bin/tokenizer.perl -l fr < ted-sent.txt > ted-sent-tok.txt
```

(5) Changez la casse à l'aide de `tr` ou `sed 's/./\L&/g'`, de façon à ce que toutes les lettres deviennent minuscules. Vérifiez le résultat.

(6) Utilisez maintenant le logiciel d'homogénéisation de casse "intelligente" à l'aide de la commande suivante, et vérifiez le résultat.

```
bin/lowercase.py -a complex -v --from PlainCorpus ted-sent-tok.txt > ted-sent-tok-low.txt
```

1. Espace représenté par le caractère ␣ pour la lisibilité.

(7) Les phrases trop courtes sont peu utiles et peuvent être supprimées. Les phrases trop longues sont difficiles à analyser, il vaut mieux les supprimer aussi. Utilisez `awk` pour afficher uniquement les phrases ($0) ayant $NF >= 5$ et $NF <= 50$ tokens. Modifiez les seuils, vérifiez combien de phrases sont filtrées et enregistrez le résultat sous le nom `ted-sent-tok-low-flength.txt`.

2. Statistique textuelle

2.1. **Compter des mots.** Les mots dans un texte sont distribués selon une loi de Zipf. C'est-à-dire, très peu de mots apparaissent un grand nombre de fois, et beaucoup de mots apparaissent seulement 1 ou 2 fois.

(1) Générez une liste appelée `ted-words.txt` de tokens (occurrences de mots) présents dans le corpus (`ted-sent-tok-low.txt`), un par ligne, à l'aide de la commande `sed` qui remplace les espaces par des retours à la ligne.

(2) Générez une liste `ted-counts.txt` contenant les types (mots uniques) avec leur nombre d'occurrences, à l'aide des commandes `sort` et `uniq -c`. Quel est le mot le plus fréquent du corpus et combien de fois apparaît-il ?

(3) À l'aide de la commande `wc`, comptez le nombre de types et de tokens dans le corpus. De combien est le rapport type/token ? Comment interpréter cette mesure ?

(4) Combien de mots apparaissent une seule fois dans le corpus (*hapax legomena*) ? Quelle est la proportion de *hapax* dans le vocabulaire ?

(5) Normalisez les comptes de mots dans l'intervalle [0..1]. Pour cela, vous pouvez utiliser `awk`, `perl`, `python` ou un autre langage de script de votre choix.

(6) Avec `gnuplot`, dessinez le graphique des comptes normalisés triés (*rankplot*). Que peut-on voir ?

```
echo "plot \"ted-counts-norm.txt\" with linespoints" | gnuplot -p
```

(7) Répétez le dessin avec une échelle logarithmique sur les axes x et y, en rajoutant les commandes `set logscale x; set logscale y` avant la commande `plot`. Que peut-on voir ?

3. Encodages de caractères

Les différents encodages de caractères représentent une source potentielle d'erreurs en traitement de corpus, surtout en français, à cause des diacritiques (accents). Il est important d'utiliser le même encodage pour tous les traitements.

(1) Vérifiez quel est l'encodage de caractères du fichier `ted-latin1.txt` à l'aide de la commande `file`. Essayez d'afficher le fichier avec `less` ou `cat`.

(2) Utilisez la commande `iconv` pour convertir le fichier de l'encodage `latin1` (ISO-8859-1) vers l'encodage Unicode `UTF-8`. Vérifiez le résultat avec `less`.

(3) Essayer de convertir le fichier original de `latin1` vers `ascii`. Que fait l'option `-c` de `iconv` ?

(4) Ouvrez les deux fichiers, en utf-8 et latin1, avec un éditeur binaire (`hexedit`, `xxd`, etc). Sur les 16 premiers octets du fichier, où sont les différences entre les encodages ? Combien d'octets sont nécessaires pour les lettre accentuées et non accentuées ?

Note : Dans la suite des TPs et pour vos projets, nous conseillons d'utiliser systématiquement l'encodage UTF-8. Vous pourrez toujours vérifier l'encodage de vos fichiers avec `file` et les convertir avec `iconv`.