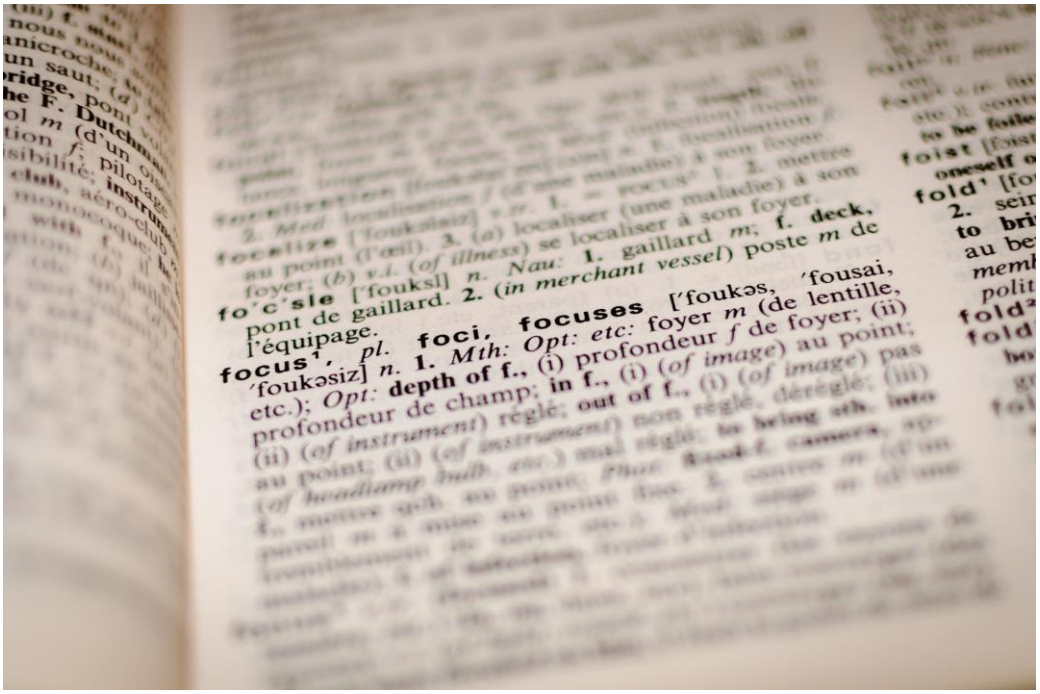
RAPPORT TP1 NLP



14/10/2017

Pierre VALENTIN

Table des matières

INTRODUCTION **2**

TACHE 1 – EXTRAIRE DES INFORMATIONS DU WORLD FACTBOOK **2**

Les Regex 2

Hymne National 2

Pourcentage d'alphabétisation 2

Exportation 3

Croissance et PIB 3

Risque Naturel 3

Nom du chef d'état 3

Adresse postale de l'ambassade des USA 4

Le résultat 4

TÂCHE 2 – DÉTECTER LA LANGUE D'UN DOCUMENT **4**

Modèle de langue 4

Lissage 5

Laplace 5

Interpolation 5

Évaluation de modèle n-gramme 5

Détection de langue 6

Résultat et Conclusion 6

CODE SOURCE **7**

Rapport TP1 NLP

PIERRE VALENTIN

INTRODUCTION

Dans ce document, je vais préciser comment j'ai extrait de l'information et réalisé de la détection de langue automatique sur des fichiers texte. Je vais ici détailler l'application des techniques vue en cours sur les expressions régulières et les modelés NGram. Le projet a été écrit en Python sans librairie de NLP. MatplotLib a était utilisé pour générer les graphiques. Code source : https://github.com/Focom/NLPWork1

TACHE 1 – EXTRAIRE DES INFORMATIONS DU WORLD FACTBOOK

Dans cette partie, nous devions extraire les informations des pays demandés depuis le world factbook. Je vais dans un premier temps détailler mes expressions régulières et enfin finir par revenir sur la précision de mon code. Dans cette partie, rendez-vous dans le répertoire part1 pour voir le code. Toutes les regex se trouvent dans info.py, sinon le reste suit la syntaxe indiquée dans le sujet.

Les Regex

Hymne National

J'ai pu identifier une ligne spécifique proche de l'information. Je parcours le fichier et essaye de faire matcher cette regex :

```
    pattern_all_file = re.compile("<span class=\"category_data\" style=\"font-
weight:normal; vertical-align:bottom;\">\"")
```

Lorsqu'on a une correspondance, il n'y a plus qu'à extraire l'information. Deux cas sont possibles, soit l'hymne est déjà en anglais dans ce cas on prend le texte entre les balises. Sinon on doit prendre le texte situé dans les parenthèses.

On essaye les deux regex et on renvoit celles qui donne un match. Ces deux lignes s'en chargent :

```
if(result_non_english):
        return result_non_english.group(4)
    if(result_english):
        return result_english.group(2)
```

Pourcentage d'alphabétisation

Ici je cherche la ligne qui contient « age 15 and over »

```
pattern_all_file = re.compile("age 15 and over")
```

Une fois cette ligne trouvée, je sais que l'information se trouve 12 ligne après. A la 12eme ligne j'extrais ce qui trouve entre les balises span.

```
pattern_good_line = re.compile("(;\">)(.*)( <\/span)")
    result = pattern_good_line.search(good_line)
    country_file.close()
    return result.group(2)
```

Exportation

Comme précédemment, je trouve un pattern unique quelque ligne avant le ligne de donnée qui m'intéresse, ensuite j'extrait la valeur. Je la transforme ensuite la valeur au bon format demandé avec cette logique :

```
    million = 1000000
    billion = 1000000000
    trillion = 1000000000000

    money_adj = pattern_money.search(clean_result.group(0)).group(3)
    if(money_adj == " million"):
        result = float_sum * million
    if(money_adj == " billion"):
        result = float_sum * billion
    if(money_adj == " trillion"):
        result = float_sum * trillion

    country_file.close()
    return "$" + str(int(result))
```

Croissance et PIB

Je cherche la ligne unique puis circule quelques lignes plus bas pour trouver la bonne ligne, j'applique ma regex puis met en forme le résultat avec le % devant.

Risque Naturel

On veut retourner oui ou non, le pays possède des risques de vent violent.

Pour cela j'identifie le paragraphe ou ce trouve les informations sur les risques naturels. Une fois trouvé je cherche si *wind, hurricans, cyclone ou typhon* se trouvent dans le paragraphe. En fonction je retourne le booléen adapté.

```
hazard_pattern1 = re.compile(".[wW]ind.")
    hazard_pattern2 = re.compile(".[Hh]urricane.")
    hazard_pattern3 = re.compile(".[Cc]yclone.")
    hazard_pattern4 = re.compile(".[Tt]yphoon.")
```

Nom du chef d'état

Comme pour les risques naturels, je crée 4 regex avec les quatre titres possibles. Si L'un deux match, je renvois le nom et prénom du chef d'état suivis de son titre.

Adresse postale de l'ambassade des USA

Premier cas simple si l'adresse contient FPO, on revoit tous ce qui se trouve avant avec cet regex :

```
    pattern_good_line = re.compile("(\>)([\w\W]*[FA]PO*) ([\w\W]*<\/span)")
```

Si cette regex ne fonctionne pas, Soit on a une BPO ou une PO, on identifie ces deux éléments et on retourne se qui se trouve après.

```
pattern_none_fbo = re.compile("(\>)(B?[P\.O]*[\w\W]*?,[\w\W]*?)[,<]([\w\W]*<\/span)?")
```

Le résultat

Lorsque je test mes expressions régulières sur le fichier csv de test, j'obtiens un taux de bonne réponse de **81%**

La précision pourrait être plus élevé si je réaliser un reg ex plus précis sur les chefs d'État de même pour gdp_per_capita, je ne mets pris que la première valeur au lieu de prendre la plus élevée. En corrigeant ces deux points, mon algorithme serait plus précis et complet.

TÂCHE 2 – DÉTECTER LA LANGUE D'UN DOCUMENT

Modèle de langue

Pour construire les modèles de langue, on parcourt les fichiers textes et on compte le nombre d'occurrence des ngram. Pour stocker ces valeurs j'utilise des dictionnaires, exemple :

Unigram - *print(unigram. proba_unigram(fileName))* :

{'A': 0.0023261824760920135, ' ': 0.1554923752907728, 'c': 0.03551305246833807, 'o': 0.06128198500904627, … }

Bigram - *print(proba_bigram(fileName))* :

{'A': {'A': 0.0, ' ': 0.2111111111111111, 'c': 0.0, 'o': 0.0, 'm': 0.0, 'p': 0.011111111111111112, 'u': 0.022222222222222223, 't': 0.05555555555555555, 'e': 0.0, 'r': 0.03333333333333333, … }, …}

Trigram – *print(proba_trigram(fileName))* :

{'co': {'A': 0, ' ': 0, 'c': 0, 'o': 0.002583979328165375, 'm': 0.6925064599483204, 'p': 0.002583979328165375, 'u': 0.028423772609819122, 't': 0, … }, …}

Ces modèles sont construits dans le cas d'un vocabulaire fermé, pour rendre ce vocabulaire ouvert, j'ajoute le caractère ¤ qui représente <unk>. La fonction helper. addUnk(n,gram,fileName) ce charge de le faire.

Une fois ajouté aux ngram précédents, on peut passer au lissage.

Lissage

Toute les fonctions se trouve dans lissage.py

Laplace

Ce lissage est très simple, on applique la formule du cours sur l'ensemble des comptes des ngram.

Ce lissage nous permet dans notre vocabulaire ouvert de donner une probabilité non nulle au ngram comportant le caractère <unk> (dans le code '¤'). Sans cela, nous n'aurions pas pu calculer la perplexité.

Interpolation

Comme pour Laplace on réalise un algorithme qui suit la formule du cours.

Cependant la gestion de <unk> n'est pas complète prenons un exemple :

Si le ngram est 'AC<UNK>'

P(AC<UNK>) = d*P(AC/<UNK>) + d *P(C/<UNK>) + d *P(<UNK>) (avec d qui représente delta)

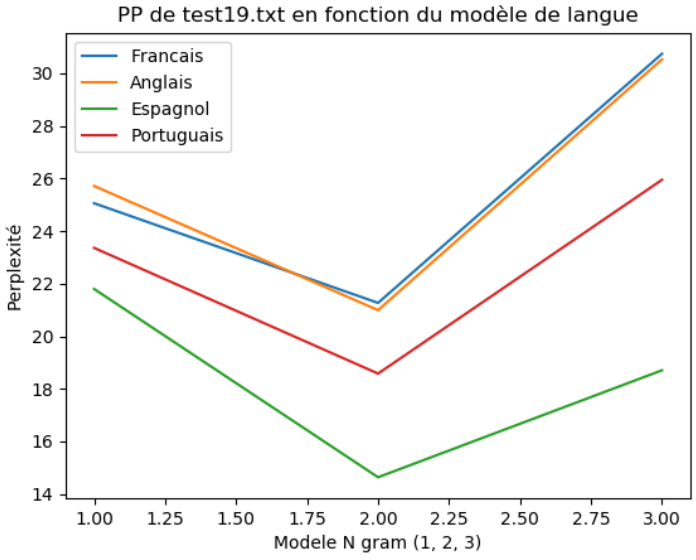
Or comme <UNK> n'est jamais apparus dans le corpus d'entrainement ça probabilité est nul est donc le lissage ne fonctionne pas. Je n'ai pas trouvé ou compris la technique pour palier à ce problème.

Évaluation de modèle n-gramme

Encore une fois, on fait appel à la formule de la perplexité vus en cour.

Si un caractère est inconnu on fait en sorte d'appeler la valeur qui correspond dans le modèle avec le caractère <unk>

Exemple de perplexité sur le fichier de test19.txt :



Détection de langue

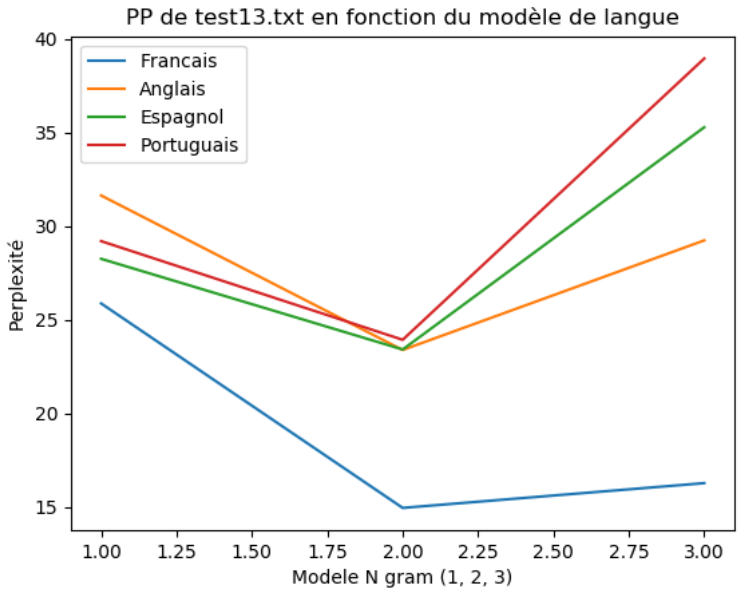
Pour détecter la langue nous créons les modèles nGram puis nous les lissons avec la méthode de Laplace.

Le lissage par interpolation n'est pas fonctionnel car il laisse les balises <unk> avec des probabilités nulles.

Une fois les modèles crée on calcule la perplexité (formule du cour) des modèles sur le même fichier de test.

Le modèle qui indique la plus faible perplexité indique la langue la plus probable.

Par exemple prenons l'évaluation sur le fichier test13.txt :



On s'aperçoit que les valeurs de perplexité pour le modèle français sont beaucoup plus basses. Le fichier est donc écrit en français.

Pour obtenir tous les résultats pour tous les fichiers exécutez cette fonction :

```
# Pour donner le résultat sur tous les fichier test dans la console
detectLang.show_all_result()
```

Résultat et Conclusion

Mes modèle Ngram prédisent bien la langue du texte, la mission est réussie.

On remarque que le modèle le plus performant dans beaucoup de cas est le bigram, le trigram n'apparait pas comme plus performant selon mes tests.

Peut-être avec un lissage plus performant que celui de Laplace, les résultats auraient été diffèrent.

Malheureusement mon lissage par interpolation n'est pas fonctionnel, je ne peux pas affirmer qu'il est plus performant.

CODE SOURCE

Aucune librairie n'a été utilisé hormis matplotlib.

Pour trouver mon code aller sur GitHub (https://github.com/Focom/NLPWork1) ou le fichier zip attaché à ce document.

La première partie se trouve dans Part1 et suis la syntaxe que vous avez demandée

La seconde partie dans Part2 tous les appels intéressant se trouve dans main.py qui est bien documenté. Libre à vous ensuite de circuler dans le code.