Normalisation

Thach Jean-Pierre Wong Leo
Département d'informatique et de recherche opérationnelle
Université de Montréal

1. Introduction

En utilisant diverses règles d'expressions régulières que nous avons conçues, nous avons nettoyé et normalisé le corpus avec l'aide de l'éditeur de flux *sed*. Ensuite, nous avons comparé nos résultats de l'extrait de 1000 lignes du corpus avec la normalisation effectuée sur le modèle pré-entrainé *Monoise*.

2. Gestion et normalisation des mots

2.1 Temps d'exécution

Pour la normalisation confectionnée à la main sur le corpus, le temps d'exécution des 35 commandes *sed* énoncées dans l'annexe est de 8 secondes.

2.2 Observations du corpus

Tableau 1 - Comparaison initiale et finale du vocabulaire et du nombre de mots sur le corpus avec une normalisation confectionnée à la main

Taille du vocabulaire initiale (*train_posts.csv*)	2101405	Réduction de 26.15%
Taille du vocabulaire finale	1551844	
Nombre de mots initiale (*train_posts.csv*)	95711929	Augmentation de 1.5682%
Nombre de mots finale	97236828	

On constate qu'une grande partie des données qui nécessitent d'être normalisées contiennent une variété d'expressions qui représentent des sentiments ou émotions. En effet, il existe différentes manières d'exprimer le rire, la tristesse, la joie, etc. Conséquemment, nous avons normalisé ces expressions en des classes pour chaque sentiment ou émotion.

Pour la classe RIRE, nous traitons les différentes variantes que nous avons répertorié, par exemple, *haha, hoho, hihi, hehe, lol, lmao*, etc. Celles-ci sont traités en vérifiant les caractères précédentes et suivantes de ces correspondances. Nous obtenons 140862 correspondances combinées pour cette classe pour le corpus.

Pour les caractères spéciaux telles que les points, les tirets, les virgules, les tildes, les points d'exclamations et des flèches, nous avons trouvé plus de 1559682 correspondances combinées.

De plus, le jeu de données possède une grande variété de séquences de lettres consécutives pour des mots. Toutefois, certaines séquences appartiennent à des mots. Pour traiter ce cas, nous avons uniquement pris en compte les correspondances qui possèdent plus de 2 lettres consécutives pour ensuite les réduire à 2. Le but de réduire à 2 lettres consécutives pour les 114880 correspondances est de tenter de normaliser avec prudence le plus possible toutes les différentes variantes puisqu'en réduisant à 1 engendrera une perte d'information, e.g. le mot *good ou goooooddd* devient *god*. Dans notre cas, le mot *goooooddd* se change à *goodd*. Par conséquent, nous conservons le mot *good* dans le vocabulaire.

Par ailleurs, nous traitons aussi les hyperliens, le temps, les émoticônes, les courriels, le replacement de *0* par la lettre *o*, etc.

2.3 Observations sur l'extrait de 1000 lignes du corpus

Tableau 2 - Comparaison initiale et finale du vocabulaire et du nombre de mots sur l'extrait de 1000 lignes du corpus avec une normalisation confectionnée à la main

Taille du vocabulaire initiale (*sent.1000*)	25357	Réduction de 7.2484%
Taille du vocabulaire finale (*normalise.out*)	23519	
Nombre de mots initiale (*sent.1000*)	189294	Augmentation de 2.277%
Nombre de mots finale (*normalise.out*)	193705	

Pour l'extrait du corpus constitué de 1000 lignes, on observe que 23 des commandes *sed* utilisées, affectent moins de 10 correspondances et que la majorité des correspondances représentent la normalisation des ponctuations. Ainsi, pour 65% des commandes *sed* confectionnées, celles-ci s'avèrent inefficace pour cet extrait.

3. Modèle pré-entrainé *Monoise*

On a testé le modèle pré-entrainé *Monoise*, un modèle de normalisation lexicale, que nous avons interrogé à partir du *Shell*, ce modèle génère des candidats de mots avec le correcteur orthographique Aspell et avec un modèle de *word embedding* entrainé sur des données de Twitter. Puis, une forêt d'arbre décisionnel est employée pour le classement de ces candidats.

Pour la forêt d'arbre décisionnel du modèle fourni se nommant LexNorm2015, nos observations se rapportent sur l'extrait de 1000 lignes du corpus qui a été distribué.

3.1 Temps d'exécution

Le temps d'exécution est de 1 heure pour l'extrait de 1000 lignes du corpus.

3.2 Observations

Tableau 3 - Comparaison initiale et finale du vocabulaire et du nombre de mots sur l'extrait de 1000 lignes du corpus avec l'utilisation de *Monoise*

Taille du vocabulaire initiale (*sent.1000*)	25357	Réduction de 20.18%
Taille du vocabulaire finale	20240	
Nombre de mots initiale (*sent.1000*)	189294	Augmentation de 0.00565%
Nombre de mots finale	190369	

3.3 Comparaison avec normalise.out

Pour les commandes qui affectent plus de 30 correspondances pour la sortie du fichier *normalise.out*, on constate que les ponctuations, les variantes d'expressions de rire et également les duplications de séquences de lettres consécutives, retournent un nombre non négligeable de correspondances. On a respectivement 1469 correspondances qui n'ont pas été gérées pour la classe DOTS, 120 pour la classe STRONG!, 238 pour la classe DASHES, 180 pour la classe RIRE combinée et 153 pour les duplications de séquences de lettres consécutives.

Finalement, on voit une réduction du vocabulaire de 7.2484% avec notre normalisation à 20.18% avec *Monoise*. Ce qui explique cette amélioration vient du fait que *Monoise* emploie Aspell pour détecter et traiter les fautes d'orthographes des mots et donc, arrive à réduire considérablement la taille du vocabulaire.

4. Discussions

On constate que les cas particuliers énumérés précédemment telles que les ponctuations ne sont pas bien gérés pour cet extrait de corpus. Conséquemment, une combinaison de ce modèle pré-entrainé pour traiter les mots possédant des erreurs d'orthographes en conjonction d'expressions régulières pour les cas particuliers réduira possiblement le vocabulaire davantage. En supposant que l'on dispose de ressources suffisantes pour effectuer la normalisation avec le modèle pré-entrainé *Monoise* pour tout le corpus d'entraînement et le corpus de validation, on pourrait possiblement améliorer la précision d'un classifieur de texte.

Références

MoNoise: Modeling Noise Using a Modular Normalization System. (van der Goot & van Noord, 2017)
https://bitbucket.org/robvanderg/monoise/src/master/

Annexes

Tableau 4 - commandes de normalisation

Voici une liste de commandes *sed* que nous avons utilisées pour normaliser les mots du corpus:

Ordre des commandes *sed*	Commande	Classe	Commande pour compter le nombre de correspondances affectées	Nombre de correspondances affectées (train_posts.csv)	Nombre de correspondances affectées (sent.1000)
1	sed 's/\.\.[\.]*/ DOTS /g' train_posts.csv > res.csv	DOTS	grep '\.\.[\.]*' train_posts.csv -o | wc -l	1290380	2795
2	sed -i -E 's/!!([!1]|one)*/ STRONG! /g' res.csv	STRONG!	grep -E '!!([!1]|one)*' res.csv -o | wc -l	147776	185
3	sed -i -E 's/\-\-[\-]*[>]+/ RIGHT_ARROW /g' res.csv	RIGHT_ARROW	grep -E '\-\-[\-]*[>]+' res.csv -o | wc -l	2258	1
4	sed -i -E 's/\-\-[\-]*/ DASHES /g' res.csv	DASHES	grep -E '\-\-[\-]*' res.csv -o | wc -l	83524	254
5	sed -i -E 's/\,\,[\,]*/ COMMAS /g' res.csv	COMMAS	grep -E '\,\,[\,]*' res.csv -o | wc -l	1188	1
6	sed -i -E 's/\~[\~]*/ TILDES /g' res.csv	TILDES	grep -E '\~[\~]*' res.csv -o | wc -l	34556	53
7	sed -i -E 's/\b([a-zA-Z0-9_\.\+\-]+@[a-zA-Z0-9\-]+\.[a-zA-Z0-9\-\.]+)\b/ COURRIEL /g' res.csv	COURRIEL	grep -E '\b([a-zA-Z0-9_\.\+\-]+@[a-zA-Z0-9\-]+\.[a-zA-Z0-9\-\.]+)\b' res.csv -o | wc -l	4288	4

8	sed -i -E 's/_urllink_/urllink/g' res.csv	urllink	grep -E '_urllink_' res.csv -o | wc -l	13650	26
9	sed -i -E 's/(http[s]?:\/\/)?www.[^]*\.[a-z]+[^,"]*/ urllink /g' res.csv	urllink	grep -E '(http[s]?:\/\/)?www.[^]*\.[a-z]+[^,"]*' res.csv -o | wc -l	5456	4
10	sed -i -E 's/\b([gbmwuha]*hah[hakz]*)\b/ RIRE /g' res.csv	RIRE	grep -E '\b([gbmwuha]*hah[hakz]*)\b' res.csv -o | wc -l	59119	92
11	sed -i -E 's/mweheh[eh]*/ RIRE /g' res.csv	RIRE	grep -E 'mweheh[eh]*' res.csv -o | wc -l	7	0
12	sed -i -E 's/\btee\ ?(heh|hee)[he]*\b/ RIRE /g' res.csv	RIRE	grep -E '\btee\ ?(heh|hee)[he]*\b' res.csv -o | wc -l	705	2
13	sed -i -E 's/\b[he]*(heh|hee) [he]*\b/ RIRE /g' res.csv	RIRE	grep -E '\b[he]*(heh|hee)[he]* \b' res.csv -o | wc -l	31654	71
14	sed -i -E 's/\b[tahi]*hihi[hi]*\b/ RIRE /g' res.csv	RIRE	grep -E '\b[tahi]*hihi[hi]*\b' res.csv -o | wc -l	405	1
15	sed -i -E 's/\b[ho]*hoho[ho]* \b/ RIRE /g' res.csv	RIRE	grep -E '\b[ho]*hoho[ho]*\b' res.csv -o | wc -l	183	0
16	sed -i -E 's/\b[hu]*huhu[hu]* \b/ RIRE /g' res.csv	RIRE	grep -E '\b[hu]*huhu[hu]*\b' res.csv -o | wc -l	104	0
17	sed -i -E 's/\b[lo]*lol[loxz]*\b/ RIRE /g' res.csv	RIRE	grep -E '\b[lo]*lol[loxz]*\b' res.csv -o | wc -l	46548	54
18	sed -i -E 's/\b(rofl|roflmao|roflmao|rotf(lmao|lmfao|l)?\b)/ RIRE /g' res.csv	RIRE	grep -E '\b(rofl|roflmao|roflmao|rotf(lmao|lmfao|l)?\b)' res.csv -o | wc -l	414	1

19	sed -i -E 's/\b(lmao|lmfao)\b / RIRE /g' res.csv	RIRE	grep -E '\b(lmao|lmfao)\b' res.csv -o | wc -l	1674	5
20	sed -i -E 's/\b[ja]*jaja[ja]* \b/ RIRE /g' res.csv	RIRE	grep -E '\b[ja]*jaja[ja]*\b' res.csv -o | wc -l	49	0
21	sed -i -E 's/[^a-z\"](=p|:p|:-p)/ STUCK_OUT_TONGUE /g' res.csv	STUCK_OUT_TONGUE	grep -E '[^a-z\"](=p|:p|:-p)' res.csv -o | wc -l	10764	4
22	sed -i -E 's/xox[xo0]*/ HUGS_KISSES /g' res.csv	HUGS_KISSES	grep 'xox[xo0]*' res.csv -o | wc -l	983	0
23	sed -i -E 's/\(?[0-9]?:?[0-9] {2}:[0-9]{2}:?[0-9]{2}?(am|pm| p.?m.?| a.?m.?)?\b/ TIME /g' res.csv	TIME	grep -E "\(?[0-9]?:?[0-9]{2}:[0-9]{2}:?[0-9]{2}?(am|pm| p.?m.?| a.?m.?)?\b" res.csv -o | wc -l	31249	35
24	sed -i -E 's/>=o/ ANGRY_FACE /g' res.csv	ANGRY_FACE	grep -E ">=o" res.csv -o | wc -l	11	0
25	sed -i -E 's/(:d[d]*|:>|:o\)|:->|:-d)/ SMILE_FACE /g' res.csv	SMILE_FACE	grep -E "(:d[d]*|:>|:o\)|:->|:-d)" res.csv -o | wc -l	8582	18
26	sed -i -E 's/:x/ KISS_FACE /g' res.csv	KISS_FACE	grep ":x" res.csv -o | wc -l	681	0
27	sed -i -E 's/(>:o|:o|:-o|:-0)[^a-z\"]/ SURPRISE_FACE /g' res.csv	SURPRISE_FACE	grep -E "(>:o|:o|:-o|:-0)[^a-z\"]" res.csv -o | wc -l	794	0
28	sed -i -E 's/(:*-\(|:*\()[^a-z\"]/ SAD_FACE /g' res.csv	SAD_FACE	grep -E "(:*-\(|:*\()[^a-z\"]" res.csv -o | wc -l	51	0
29	sed -i -E 's/(\^\.\^|\^_\^|\^o\^)/ HAPPY_FACE /g' res.csv	HAPPY_FACE	grep -E '(\^\.\^|\^_\^|\^o\^)' res.csv -o | wc -l	2765	8

30	sed -i -E 's/(\@_\@)/ PANIC_FACE /g' res.csv	PANIC_ FACE	grep -E '(\@_\@)' res.csv -o | wc -l	158	14
31	sed -i -E 's/ (:\/|:s|=\/) / HESITANT_FACE /g' res.csv	HESITA NT_FAC E	grep -E ' (:\/|:s|=\/) ' res.csv -o | wc -l	1539	0
32	sed -i -E 's/\bgrr[r]*\b/ ANG RY_SENTIMENT /g' res.csv	ANGRY_ SENTIM ENT	grep -E '\bgrr[r]*\b' res.csv -o | wc -l	4230	9
33	sed -i -E 's/([a-z])0([a-z])/\1o\2/g' res.csv		grep -E '[a-z]+0[a-z]+' res.csv -o | wc -l	5844	3
34	sed -i -E 's/([a-z])\1\1+/\1\1/g' res.csv		grep -E '([a-z])\1\1+' res.csv -o | wc -l	114880	180
35	sed -i -E 's/\b[wo]*woo[|-]?[hwo]*\b/ CELEBRATION_SENTIM ENT /g' res.csv	CELEBR ATION_ SENTIM ENT	grep -E '\b[wo]*woo[|-]?[hwo]*\b' res.csv -o | wc -l	8054	6