Le prétraitement de texte

TALN Semaine 2

Ces diapos sont basées sur celle de Dan Jurafsky

Plan pour aujourd'hui

1. **Qu'est-ce que le prétraitement ?**
2. **Caractériser les strings avec RegEx**
3. **Tokenisation**
4. **Normalisation et lemmatisation des mots**
5. ***Exercices de groupe***

Qu'est-ce que le prétraitement ?

Le prétraitement est la première étape de toutes les tâches et modèles de TALN ; il s'agit de *normaliser* les données textuelles.

Google is set to file its own suggestions for fixing the search monopoly by Dec. 20. Both sides can modify their requests before Judge Mehta is expected to hear arguments on the remedies this spring. He is expected to rule by the end of the summer.

Kent Walker, Google's president of global affairs, called the government's proposal "extreme."

"D.O.J.'s wildly overbroad proposal goes miles beyond the court's decision," he said in a blog post. "It would break a range of Google products — even beyond Search — that people love and find helpful in their everyday lives."

Qu'est-ce que le prétraitement ?

Le prétraitement est la première étape de toutes les tâches et modèles de TALN ; il s'agit de *normaliser* les données textuelles.

Google is set to file its own suggestions for fixing the search monopoly by **Dec. 20.** Both sides can modify their requests before Judge Mehta is expected to hear arguments on the remedies this spring. He is expected to rule by the end of the summer.\n

Kent Walker, Google's president of global affairs, called the government**'s** proposal **"**extreme.**"**

"**D.O.J.**'s wildly overbroad proposal goes miles beyond the court's decision," he said in a blog post. "It would break a range of Google products **—** even beyond Search **—** that people love and find helpful in their everyday lives."

Qu'en est-il de diviser le texte en mots ? Des sous-mots ?

Qu'est-ce que le prétraitement ?

Le prétraitement est la première étape de toutes les tâches et modèles deTALN; il s'agit de normaliser les données textuelles en:

- **Nettoyant le formatage**
- **Normalisant le texte**
- **Segmentant le texte en tokens, mots, phrases, ...**

Regular Expressions (RegEx)

RegEx est un langage formel pour spécifier des strings de texte

Ex. Comment pouvons-nous rechercher des mentions de ces animaux mignons dans le texte ?

- woodchuck
- woodchucks
- Woodchuck
- Woodchucks
- Groundhog
- groundhogs

Pouvons-nous rédiger une spécification formelle qui tient compte de tout cela ?

RegEx: Disjonctions (Ou)

- Les caractères ou plages entre parenthèses [ ] représentent des expressions disjonctives

Pattern	Matches
`[wW]oodchuck`	Woodchuck, woodchuck
`[1234567890]`	Tout chiffre singulier

Pattern	Matches	
`[A-Z]`	Une lettre majuscule	`Drenched Blossoms`
`[a-z]`	Une lettre minuscule	`my beans were impatient`
`[0-9]`	Un chiffre singulier	`Chapter 1: Down the Rabbit`

RegEx: Disjonctions (Ou)

- Le symbole pipe | est également pour la disjonction

Pattern	Matches
`groundhog|woodchuck`	`woodchuck`
`yours|mine`	`yours`
`a|b|c`	= [abc]
`[gG]roundhog|[Ww]oodchuck`	`Woodchuck`

RegEx: caractères génériques: . ? * +

- Les caractères génériques peuvent correspondre à n'importe quel symbole pour un nombre défini de fois.

Pattern	Matches	Examples
`beg.n`	Un symbol	`begin` `begun` `beg3n` `beg n`
`woodchucks?`	Symbol optionel	`woodchuck` `woodchucks`
`to*`	0 ou plusieurs fois un symbol	`t` `to` `too` `tooo`
`to+`	1 ou plusieurs fois un symbol	`to` `too` `tooo` `toooo`

Regular Expressions (RegEx)

RegEx est un langage formel pour spécifier des strings de texte

Ex. Comment pouvons-nous rechercher des mentions de ces animaux mignons dans le texte ?

- woodchuck
- woodchucks
- Woodchuck
- Woodchucks
- Groundhog
- groundhogs

```
[gG]roundhogs?|[Ww]oodchucks?
```

RegEx: Négation (Non)

Carat (^) comme premier caractère dans [ ] signifie la négation de la liste :

- Le carat signifie négation uniquement lorsqu'il est d'abord en [ ]
- Autres caractères spéciaux (., *, +, ?) Perdre leur signification spéciale à l'intérieur [ ]

Pattern	Matches	Examples
`[^A-Z]`	Pas une lettre majuscule letter	`Oyfn pripetchik`
`[^Ss]`	Pas ‘S’ ni ‘s’	`I have no exquisite reason”`
`[^.]`	Pas un point	`Our resident Djinn`
`[e^]`	Soit ‘e’ ou ‘^’	`Look up ^ now`

RegEx: Ancres ^ $

- ^ au début d'une regex est pour au début du string
- $ marque la fin du string

Pattern	Matches
^[A-Z]	Palo Alto
^[^A-Za-z]	1 “Hello”
\.$	The end.
.$	The end? The end!

Alias pour RegEx

- Les regex communs ont des raccourcis.

Pattern	Expansion	Matches	Examples
`\d`	`[0-9]`	Tout chiffre singulier	`Fahreneit 451`
`\D`	`[^0-9]`	Tout carac non-digit	`Blue Moon`
`\w`	`[a-ZA-Z0-9_]`	Tout alphanumeric or _	`Daiyu`
`\W`	`[^\w]`	Tout non alphanumeric or _	`Look!`
`\s`	`[ \r\t\n\f]`	Tout espace (space, tab)	`Look␣up`
`\S`	`[^\s]`	Non-espace	`Look up`

RegEx avec Python

Regex et Python utilisent tous deux la barre oblique inverse "\" pour les caractères spéciaux.

- On doit ajouter une barre oblique inverse de plus pour les regexs en Python !
    - Ex. ”\\d+" pour rechercher 1 ou plusieurs chiffres

- **OU:** utiliser la notation pour **raw string en Python** pour regex:
    - r"\d+" correspond à un ou plusieurs chiffres (au lieu de "\\d+")

Pourquoi les RegEx?

Largement utilisé à la fois dans les universitaires et dans l'industrie :

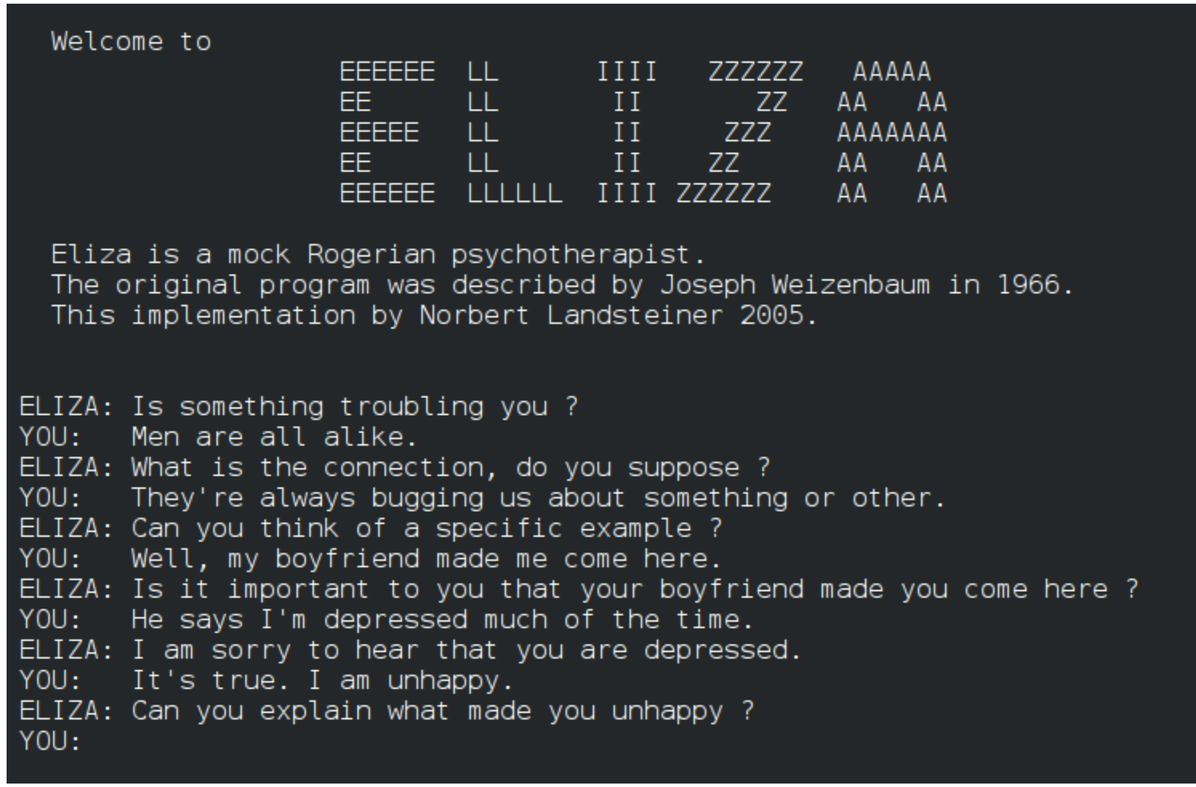
1. Une partie de la plupart des tâches de TALN, même pour les grands pipelines de LLM

2. Très utile pour l'analyse des données pour toutes données textuelles

.

Premier chatbot: ELIZA (1966)

- ELIZA était un premier système de TALN destiné à imiter un psychothérapeute.
- Ça fonctionnait via la reconnaissance de formes (**pattern matching):**
    - **Input:** “I need X.”
    - **Output:** “What would it mean if you got X?”



Premier chatbot: ELIZA (1966)

- Exemples de pattern matching de ELIZA:
    - r“.* I’m (depressed|sad)” -> “I am sorry to hear that you are X”
    - r“.* all .*” -> “In what way?”
    - r”.* always .*” -> “Can you think of a specific example?”

Tokenisation

La tokenisation est de segmenter le texte en tokens (ou en unités significatives pour le traitement).

Pourquoi c’est important ?

- **La tokenisation est la première étape de tous les modèles de langage.**

- Le langage est un système symbolique qui prend un signal continu (par exemple, la parole audio ou les gestes de la main) et le transforme en une séquence de symboles significatifs (par exemple, des morphèmes, des signes ou des mots) **-> La tokenisation décide quels sont ces unités symboliques significatives dans un système de TALN.**

Tokens vs types

- **Type**: Un élément du Vocabulaire.
- **Token**: Une instance d’un type dans le texte en cours d'exécution.

Ex. Supposons que nous tokenisons au niveau du mot :

“The NLP course students enjoyed the course very much”

Combien de tokens? **9**

Combien de types? **7**

Tokenisation simple

- Un algorithme de tokenisation de mots très simple pourrait diviser le texte sur les espaces et les signes de ponctuation.

```
str = “He is expected by the end of the summer.”

delimiter = r“[,?\.!]\s?|\s”

tokens = str.split(delimiter)
```

```
[“He”,“is”,“expected”,“by”,“the”,“end”,“of”,“the”,“summer”]
```

Problème avec cette tokenisation

On ne peux pas simplement supprimer aveuglément la ponctuation:

- m.p.h., Ph.D., AT&T, cap’n
- prix ($45.55)
- dates (01/02/06)
- URLs (http://www.hec.ca)
- hashtags (#hec)
- addresse courriel (someone@cs.colorado.edu)
- **Clitique:** a word that doesn't stand on its own, ex. "je" dans “j’ai", "le" dans “l’honneur”

Qu'en est-il des espaces ? Les expressions de plusieurs mots devraient-elles être des mots séparés?

- New York, rock ’n’ roll

Tokenisation avec NLTK

- **Alternative:** Au lieu de faire correspondre les délimiteurs entre les mots, essayons de faire correspondre des ensembles de caractères qui peuvent être des mots !

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)    # set flag to allow verbose regexps
...     ([A-Z]\.)+        # abbreviations, e.g. U.S.A.
...   | \w+(-\w+)*        # words with optional internal hyphens
...   | \$?\d+(\.\d+)?%?  # currency and percentages, e.g. $12.40, 82%
...   | \.\.\.            # ellipsis
...   | [][.,;"'?():-_`]  # these are separate tokens; includes ], [
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

Bird, Loper and Klein (2009), *Natural Language Processing with Python*. O'Reilly

Tokenisation dans d’autres langues

Mais...les langues n'utilisent pas tous des espaces ou des caractères uniques !

Alors, qu'est-ce qui détermine les limites d’un token dans d'autres langues ?

Ex. Le chinois

Les mots chinois sont composés de caractères appelés "hanzi" (ou parfois juste "zi")
Chacun représente une unité de sens appelée morphème.
Chaque mot en contient en moyenne 2,4.
Mais décider ce qui compte comme un mot est complexe.

Tokenisation dans d’autres langues

Alors, comment devrions-nous tokeniser le chinois ?

姚明进入总决赛 “Yao Ming reaches the finals”

3 words?
姚明 进入 总决赛
YaoMing reaches finals

5 words?
姚 明 进入 总 决赛
Yao Ming reaches overall finals

7 characters? (don't use words at all):
姚 明 进 入 总 决 赛
Yao Ming enter enter overall decision game

Tokenisation statistique

Pouvons-nous être plus intelligents avec la tokenisation et essayer *d'apprendre* les morphèmes pertinents d'un langage basé sur un corpus ?

Trois algorithmes communs:

- **Byte-Pair Encoding (BPE)** (Sennrich et al., 2016)
- Unigram language modeling tokenization (Kudo, 2018)
- WordPiece (Schuster and Nakajima, 2012)

Tokenisation statistique

- Il y a deux parties à tout algorithme de tokenisation statistique :

    1. **Un apprenant** qui apprend le Vocabulaire ou l'ensemble de type pour une langue basé sur un corpus d’entrainement
    2. **Un segmenteur** qui segmente un corpus de test en une séquence de tokens basé sur le vocabulaire de type appris.

Byte-Pair Encoding (BPE)

L’APPRENANT

1. Initialiser le vocabulaire comme la liste de tous les caractères individuels :
   V = [A, B, C, D,…, a, b, c, d….]
2. Répétez ensuite les étapes suivantes jusqu'à ce que la condition de terminaison soit atteinte :
   1. Choisissez les deux symboles les plus fréquemment adjacents dans le corpus d'entraînement (ex. « A », « B »)
   2. Ajoutez un nouveau symbole fusionné 'AB' au vocabulaire **V**
   3. Remplacez chaque 'A' 'B' adjacent dans le corpus par 'AB'.
3. Condition de terminaison: la taille souhaitée du vocabulaire **V** est atteinte.

Byte-Pair Encoding (BPE)

L'APPRENANT

function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens til k times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

Byte-Pair Encoding (BPE)

L’APPRENANT

Notez
Pour tenir compte du signal provenant des espaces dans de nombreuses langues, un symbole spécial de fin de mot '__' est généralement ajouté à tous les mots avant les espaces dans le corpus d'entraînement.

Ex. Tokenization_ is_ the_ first_ step_ to_ all_ NLP_ language_ models_ .

Byte-Pair Encoding (BPE)

LE SEGMENTEUR

- Maintenant que nous avons appris un vocabulaire de type, comment segmentons-nous réellement notre corpus de test en tokens?

- For-loop sur le vocabulaire, avec des tokens fusionnés dans l'ordre dans lequel nous les avons appris:

1. For chaque token fusionné dans le vocabulaire, ex, ‘`AB`’:
   1.1. Remplacez de façon greedy chaque '`A`' '`B`' adjacent par '`AB`'.
2.

Byte-Pair Encoding (BPE)

L'APPRENANT

function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens til k times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

Si votre corpus d'entrainement = votre corpus de test, cette dernière étape s'occupe de la partie segmenteur de l'algorithme.

Normalisation de mots

La tokenisation est une étape de prétraitement ; il existe d'autres formes de prétraitement que nous pourrions vouloir envisager, telles que diverses formes de normalisation des mots.

- **Conversion de majuscule/minuscule (case folding):** mettre tous les caractères soit en majuscule ou en minuscule (ex. TALN -> taln)

  Notez: la casse des lettres peut être utile pour les tâches de nature sémantique: (ex. US versus us — le pays vs le pronom)

- **Lemmatisation:** Remplacer les mots par leur mot du dictionnaire (ex. am, are, is -> be)

  Notez: Les indices syntaxiques comme le temps de verbe, les pluriels, etc. peuvent être utiles pour des tâches générative.

- **Stemming:** Supprimer les affixes des mots, en ne gardant que les “stem” (objectif similaire à la lémmatisation)

Lemmatisation

Représenter les mots par leur **lemme**, leur racine partagée = forme de mot-tête du dictionnaire :

am, are, is -> be
car, cars, car's, cars' -> car

He is reading detective stories -> He be read detective story

Lemmatisation

La lemmatisation se fait à l'aide d'un analyseur morphologique (le plus souvent des analyseurs heuristiques basés sur des règles qui dépendent du langage) qui divise les mots en morphèmes.

Rappelez-vous, la semaine dernière, nous avons parlé des morphèmes, les plus petites unités de sens dans une langue. Il y a deux types :

1. **Lexicaux** (**Stems)** : Les unités de base qui portent un sens
2. **Grammaticaux** (**Affixes)** : Les sous-mots qui doivent s'attacher à un morphème lexical et ont souvent des fonctions grammaticales.

heavenly -> heaven/-ly
cars -> car/-s
Incompetent -> in-/competent

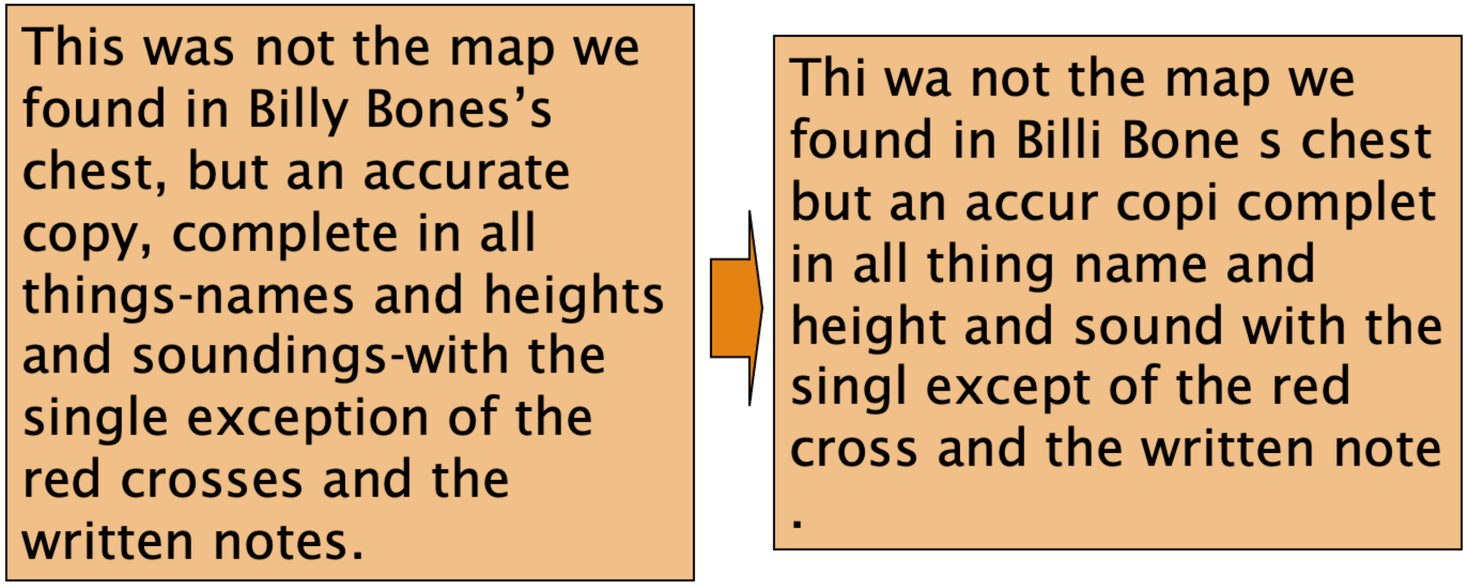
Stemming

Le stemming est une approche légèrement plus simple avec le même ethos que la lemmatisation où vous supprimez toutes les affixes, réduisant tous les mots à leurs stem, ou morphème lexical.

heavenly -> heaven
cars -> car
Incompetent -> competent ?? Incompetent

Stemming

En pratique, le stemming est souvent grossier et sélectif par rapport aux affixes sont supprimées en fonction de l'ensemble des règles heuristiques utilisées.



[pause -15 minute]

Essayons le prétraitement

Mettez vous en équipe de 2!

Ouvrez les exercices/week 2 et commencez à écrire/exécuter du code !