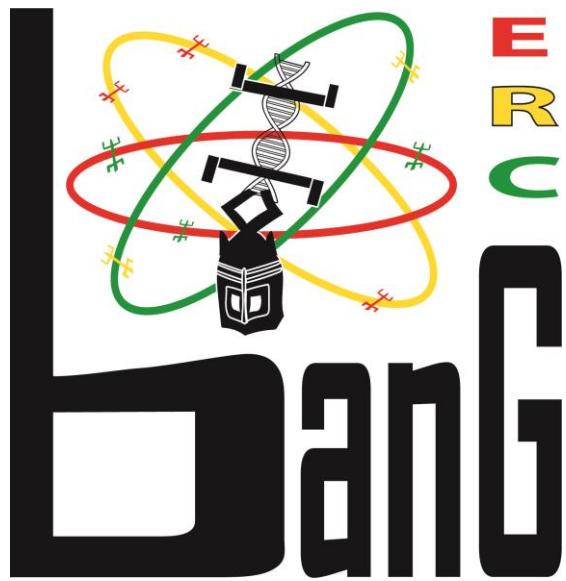




LINGUISTIQUE COMPUTATIONNELLE

COURS 1



Promise Dodzi Kpoglu
promisedodzi@gmail.com / promise-dodzi.kpoglu@cnrs.fr

UCAD, Dakar.
Novembre, 2024.



Prélude

Données brutes:

1	DOCULECT	GLOSS	IPA
2	BankanTey	domestic animal	dʒáw+dì+m
3	BankanTey	domestic animal	dá.ábà+m
4	BenTey	domestic animal	àrsè.é+m
5	Bunoge	domestic animal	kó+m.bò
6	DogulDomBendi	domestic animal	bélè+g
7	DogulDomBendi	domestic animal	bélè+gù
8	DonnoSo	domestic animal	bélú
9	DonnoSo	domestic animal	bélú
10	JamsayGourou	domestic animal	gàr+sè+gé
11	JamsayDouentz	domestic animal	àsè+gé
12	JamsayDouentz	domestic animal	gàsè+gé
13	JamsayMondoro	domestic animal	àsè+ge
14	Mombo	domestic animal	dèbù+béli
15	Mombo	domestic animal	béli
16	BonduSoNajamt	domestic animal	dúmēt+ŋ.gó
17	BonduSoKindige	domestic animal	dùmō.ó+n
18	Nanga	domestic animal	gàsè+gé
19	Penange	domestic animal	dá.ábá
20	PergeTegu	domestic animal	àsè+gé

Prélude

code

```
def lexstatExperiment(input_file, coverage_num, lexstat_output):  
  
    wl = Wordlist(input_file)  
    retain = []  
    for language, coverage in wl.coverage().items():  
        if coverage > coverage_num:  
            retain.append(language)  
  
    new_wl = {0: [c for c in wl.columns]}  
    for idx, language in wl.iter_rows("doculect"):  
        if language in retain:  
            new_wl[idx] = wl[idx]  
    new_wl = Wordlist(new_wl)  
  
    lex = LexStat(new_wl)  
    lex.get_scorer(runs=10000)  
    lex.cluster(method='lexstat', threshold=0.55, ref='cogid')  
  
    return lex.output('tsv', filename=lexstat_output)  
  
  
def alignmentExperiment(input_name, output_name, output_type="html"):  
    lex=LexStat(input_name)  
    alm = Alignments(lex, ref='cogid')  
    alm.align()  
    return alm.output(output_type, filename=output_name)
```

Prélude

ID	DOCULECT	CONCEPT	IPA	TOKENS	COGID
8954	BankanTey	(man's) sister	pòjâ+m	p ò j â + m	282 ²
8955	BenTey	(man's) sister	pòjâ.á	p ò j â á	282 ²
8956	DogulDomBendiely	(man's) sister	sáà	s á à	284 ⁶
8957	DogulDomKundialang	(man's) sister	sà.áŋ	s à á ŋ	284 ⁶
8958	DonnoSo	(man's) sister	sà.à	s àà	284 ⁶
8959	DonnoSo	(man's) sister	jà.àsà.à	j àà s àà	287 ³
8960	JamsayDouentza	(man's) sister	jèsà.á	j è s à á	287 ³
8962	TebulUre	(man's) sister	sá	s á	284 ⁶
8963	Tiranige	(man's) sister	bà.à	b àà	289
8964	TommoSoTongoTongo	(man's) sister	sáá	s áá	284 ⁶

Qui suis-je

- Etat civil:
 - Nom et prénoms: Promise Dodzi Kpoglu
 - Nationalité: ghanéenne
 - Profession : chercheur post-doctorat au LLACAN – CNRS (projet BANG)
- Formation:
 - Licence: langue et culture française, KNUST, Kumasi, Ghana
 - Master: Sciences de langage, spécialité Linguistique, Université de Lille, France
 - Doctorat: Sciences de langage, spécialité Linguistique, Université de Lille, France/ Leiden University, Pays-Bas
- Mes recherches:
 - Morphosyntaxe
 - phonétique/phonologie
 - langues Kwa
 - **TAL assisté par ordinateur**

Plan de cours

- Introduction ---- cours 1
 - Le Traitement Automatique des Langues (TAL) assisté par ordinateur
 - Le mot
 - Le nettoyage
- Normalisation ---- cours 2
 - La tokenisation
 - La lemmatisation
 - La racinisation (stemming)
- Métriques de distance ---- cours 3
 - Jaccard
 - Hamming
 - Levenshtein
- Travaux pratiques: identification des cognats ---- cours 4
 - Codage d'un algorithme simple de détection des cognats

Introduction

- Une même notion exprimée différemment selon le domaine d'expertise:
 - Industries de la langue --- l'industrie
 - Linguistique computationnelle (LC) ---- la linguistique
 - Traitements Automatiques des Langues (TAL) ---- l'informatique/la linguistique/la traductologie/les humanités numériques etc.
- L'utilisation de la dernière terminologie:
 - Technologies de langues --- terminologie qui n'est pas strictement académique
 - Linguistique computationnelle --- notion assez restreinte à la linguistique
 - Souvent, chevauchement entre LC et TAL (en termes de méthodologies, terminologies etc.)
- Traitements Automatique des Langues:
 - 3 termes lexicaux nécessitant une explication afin de comprendre les démarches impliquées.

Introduction

Langues

- Langues:
 - Conception saussurienne: Système de signes, règles et conventions partagé par une communauté
- Les langues constituent les supports des données linguistiques manipulées en TAL, et qui peuvent prendre plusieurs formes:
 - Textes écrits (long ou court)
 - Dialogues (écrits, oraux, signées)
 - Unités syntaxiques
 - Unités lexicales/morphologiques
 - Unités phonétiques
- Pour ce cours, nous allons nous concentrer sur les unités lexicales

Introduction

Automatique

- Automatisme:
 - Opération par des moyens mécaniques, dans ce cas l'ordinateur
 - Opération automatique ici évoque une suite d'actions effectuées par l'ordinateur dans un ordre chronologique i.e. un programme
- Automatisation de l'analyse des langues:
 - Automatisation complète: tâche effectuée en totalité par l'ordinateur sans l'intervention de l'humain
 - Automatisation partielle (assisté par ordinateur): tâche partagée, une partie effectuée par l'ordinateur et une autre partie effectuée par l'humain
- Pour ce cours, nous allons nous situer dans un cadre d'automatisation partielle:
 - Vous aurez à travailler en binôme avec l'ordinateur

Introduction

Traitement

- Traitement:
 - Action sur l'objet, langue, en lui faisant subir un travail de manipulation, transformation, voire création (via des opérations automatiques)
- En TAL (assisté par ordinateur), ce traitement peut nécessiter:
 - des connaissances linguistiques ---- prérequis pour l'humain
 - des formalismes, qui permettent à l'humaine de communiquer avec l'ordinateur -----
ex: des langages de programmation
 - La disponibilité d'un moyen technique pour l'opération automatique i.e. un ordinateur
- Dans ce cours, nous allons nous appuyer sur nos connaissances:
 - Des langues (vos données)
 - Manipulation d'un ordinateur
 - Du langage de programmation python (n'ayez pas peur!)

L'unité lexicale et TAL

Du corpus au mot

- Toute activité en TAL commence avec un corpus
 - Qu'est-ce qu'un corpus?
 - Composé des textes et des métadonnées
 - textes d'un linguiste de retour du terrain
 - des textes traduits ou non-traduits
 - des textes de différents écrivains
 - Des textes historiques, légaux etc.
 - Texte, souvent composé d'une séquence de graphèmes.
 - Pour l'instant, nous allons nous concerter que du texte, et aborder le cas des métadonnées à un autre moment
 - Les tâches à exécuter demande souvent que le corpus soit réduit à des formes atomiques
 - Certaines étapes s'avèrent pertinentes dans un processus de ‘pré-traitement’.
 - Le nettoyage des données
 - La tokenisation
 - La lemmatisation
 - La racinisation

L'unité lexicale et TAL

Du corpus au mot: le nettoyage

- Prenons ce texte par exemple:

⌚ Trop cool le show d'hier avec @AmadeusOfficiel et @officielwallyseck 😊 ! J'ai trop hâte de te voir
100 à ziguinchor!!! #Jëli #Mbëggeeldafaneex # <https://t.co/abc123efg>

- Nettoyage à faire dans ce texte (en utilisant des expressions régulières):

- Retirer ou convertir les émojis
- Retirer ou anonymiser les mentions de noms afin de respecter la confidentialité i.e. éthique scientifique
- Convertir les hashtags en mot-simples i.e. retirer les #
- Supprimer le lien URL qui est non-pertinent pour une analyse
- Eliminer les ponctuations pour retourner un texte composer que de mots.
- Mettre tout les lettres en minuscule.

- Le texte nettoyé:

trop cool le show dhier avec amadeusofficiel et officielwallyseck jai trop hâte de te voir à ziguinchor jëli mbëggeeldafaneex

L'unité lexicale et TAL

Du corpus au mot: le nettoyage

- Exercice:
 - Connectez-vous au lien <https://regex101.com/>
 - Qui peut nous décrire la page?
 - Familiarisez-vous avec certaines des expressions les plus communes dans la rubrique *QUICK REFERENCE*.
 - Saisissez un texte dans l'espace *TEST STRING*.
 - Sélectionnez ce que vous voulez faire dans la rubrique *FONCTION*.
 - Mettez l'expression rationnelle souhaitée dans l'espace *REGULAR EXPRESSION*.
 - Les formes sélectionnées sont en surbrillance.
 - Consultez Google pour des expressions rationnelles plus sophistiquées pour des tâches sophistiques.
 - Sélectionnez le langage de programmation souhaité dans la rubrique *FLAVOR*.
 - Copiez le code en cliquant sur *Code Generator* sous *TOOLS*.

L'unité lexicale et TAL

Du corpus au mot: la tokenisation

- La tokenisation fait référence au fait de réduire les textes aux unités atomiques

La tokénisation fait référence au fait de réduire les textes aux unités atomiques

[*La, tokenisation, fait, référence, au, fait, de, réduire, les, textes, aux, unités, atomiques*]

- A première vue, l'on pourra penser qu'il suffit de séparer les mots ayant un espace/une ponctuation entre eux pour les tokéniser.
 - Ceci présuppose qu'à un mot = espace/ponctuation devant; espace/ponctuation derrière
- Tokénisez le texte suivant, qui provient de la langue turc:

korkusuzlaştırbileceklerimiz

‘ceux que nous pouvons rendre courageux’

- Sachant bien que:

Korku = la peur

Korkusuz – sans peur (la suffixation en "-suz" induit le sens de “sans peur”)

Korkusuzlaştır – rendre ‘sans peur’ (la suffixation en "-laştı" forme un verbe qui indique le fait de “rendre quelqu'un sans peur”)

Korkusuzlaşabilecek – capacité à rendre ‘sans peur’ (la suffixation en "-ebilecek" ajoute un sens de possibilité “peut rendre ‘sans peur’”)

L'unité lexicale et TAL

Du corpus au mot: la tokenisation

- Des langues agglutinantes telles que le turc pose un défi à la tokénisation ‘banale’.
- La tradition orthographique d'une langue peut aussi poser un défi à la ‘tokénisation banale’

chinois: 请问您可以告诉我们怎么回家吗?

français: ‘Est-ce que vous pouvez nous indiquer comment faire pour rentrer à la maison?’

- Il n'y a pas ni d'espace, ni de ponctuation entre les graphèmes.
- Même le registre peut poser un challenge à la tokenisation banale

⌚ Trop cool le show d'hier avec @AmadeusOfficiel et @officielwallyseck 😊 ! J'ai trop hâte de te voir 💯 à ziguinchor!!! #Jëli #Mbëggéeldafaneex # <https://t.co/abc123efg>

- Doit on séparer *Mbëggéeldafaneex* en des mots distincts où le garder tel afin de préserver le style?

L'unité lexicale et TAL

Du corpus au mot: la tokenisation

- Pour la question de style; il suffit de faire un choix.
 - Quels choix feriez-vous si vous avez une telle donnée?
- Pour les défis posés à la tokénisation par les propriétés morphologiques et orthographiques des langues, la solution consiste souvent à segmenter morphologiquement.
- Segmentation morphologique: réduction en des formes morphémiques des textes:
le-s enfant-s jou-ent
Korku-suz-laştır-abilecek-lerimiz

L'unité lexicale et TAL

Du corpus au mot: la lemmatisation/racinisation

- Pour les langues flexionnelles, considérez- vous le formes *mangeons* et *mange* dans cette phrase, comme étant différentes?

nous mangeons ce qu'il mange

- La racinisation consiste à réduire une forme à sa racine – enlever tout indice de la morphosyntaxe:

il faut que nous allions → il fau que nous all

nous mangeons ce qu'il mange → nous mang ce qu'il mang

- La lemmatisation consiste à réduire une forme à son lemma – lexème de base:

il faut que nous allions → Il falloir que nous aller

nous mangeons ce qu'il mange → nous manger ce que il manger

L'unité lexicale et TAL

Conclusion

Exercice:

Les enfants jouent dans le parc!!.

Ils aiment, si tu veux savoir, 23 courir et s'amuser ☺.

Chaque jour, ils apprennent de nouvelles choses.

Ils sont heureux.

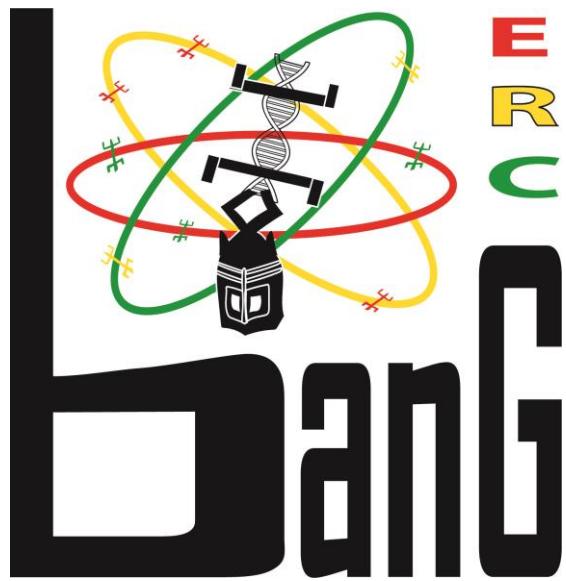
- Faire un pré-traitement sur le texte ci-dessus en utilisant les procédés suivants:
 - Le nettoyage (en vous servant d'une expression rationnelle, si possible)
 - La tokénisation
 - La racinisation
 - La lemmatisation
- Pouvez-vous appliquer ces procédés sur un texte dans votre langue maternelle?

Bibliographie

- Bellman, R., & Kalaba, R. (1957). Dynamic programming and statistical communication theory. *Proceedings of the National Academy of Sciences*, 43(8), 749-751.
- Dickinson, M., Glass, L., Brew, C., & Meurers, D. (2024). *Language and computers*. BoD–Books on Demand.
- Haspelmath, M. (2014). On system pressure competing with economic motivation. *Competing motivations in grammar and usage*, 197-208.
- Jurafsky, D., & Martin, J. H. (2008). Speech and Language Processing: An introduction to speech recognition, computational linguistics and natural language processing. *Upper Saddle River, NJ: Prentice Hall*.
- Wagner, R. A., & Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1), 168-173.



LINGUISTIQUE COMPUTATIONNELLE COURS 2: LA NORMALISATION



Promise Dodzi Kpoglu
promisedodzi@gmail.com / promise-dodzi.kpoglu@cnrs.fr

UCAD, Dakar.
Novembre, 2024.



Plan du cours

Normalisation ---- cours 2

- La distinction token/type
- Le N-grams
- La loi de puissance de Zipf
- La loi de brièveté de Zipf
- La loi de Heaps

L'unité lexicale et TAL

Le mot: token/type

- La tokénisation produit des *tokens*
- Une suite consécutive de *tokens* d'un certain nombre (N) dans un texte est appelé un *N-gram*
 - Suite de deux, trois etc. *tokens* === bigram, trigram, tetragram, pentagram etc.
====2-gram, 3-gram, 4-gram, 5-gram etc.

Je mange

Je mange toujours

Je mange toujours bien

Je mange toujours bien mieux

- On peut générer pour un texte donné, un certain n-grams

Je suis là

[(je, suis), (suis, là)]

- Exercice:
 - Quel genre d'analyse peut-on faire avec l'ensemble de n-grams dans un texte?
 - On peut aussi générer un n-gram de graphèmes/sons, permettant une analyse phonétique/phonotactique, voire morphologique.

L'unité lexicale et TAL

Le mot: token/type

- Identification des mots:

Nous mangeons ce que nous avons décidé de manger

Je suis à Dakar et il est à Accra

- Circonscrire la notion de mot (distinction *token/type*):

- Mot-forme (*token* en TAL)

Type en TAL

- Morphème: indépendant/lié vs. grammatical/lexical
 - Formes indépendantes lexicales: dérivées vs. non-dérivées

Ex: *une rose est une rose est une rose = 3 types (une, rose, est), 8 tokens*

- On peut décider de garder les formes fléchies des *types*, ou leurs formes lemmatisées, ou racinisées selon les objectifs.

L'unité lexicale et TAL

Le mot: token/type

- Or, les données linguistiques recueillies sur le terrain (le corpus) contiennent des différentes propriétés morphologiques.
- Activité:
 - Pouvez-vous nous spécifier quelques propriétés morphologiques des données à votre disposition?
 - Si vous voulez classifier des textes selon des thèmes en vous appuyant sur les mots-clés dans chaque texte, quels choix de *types* feriez-vous?
 - Qu'est-ce qui motive votre choix?
 - Et si vous voulez faire un algorithme pour gloser automatiquement votre texte à partir d'un glossage précédent que vous avez effectué?
- Question banale mais pertinente:
 - Comment les formes morphologiques sont-elles distribuées dans le corpus?

L'unité lexicale et TAL

Le mot dans le corpus: les principes de distribution

- La distribution des mots dans un corpus suit deux principes fondamentaux:
 - La loi de puissance de zipf
 - La loi de brièveté de zipf
- La loi de puissance de zipf:
 - La fréquence d'un mot est inversement proportionnelle à son rang de fréquence.
 - Ceci veut dire que dans un corpus, le *type*(A) le plus fréquent d'un corpus a à peu près deux fois la fréquence du deuxième *type* (B) le plus fréquent
 - Si les fréquences sont tracées sur une illustration en deux dimension (x et y), on notera que la fréquence des mots chute brusquement : de quelques mots très fréquents (mots grammaticaux) jusqu'à une longue traîne de nombreux mots peu fréquents (mots lexicaux).
 - Ceci représente une distribution de la loi de puissance, générée par une fonction mathématique où y est une fonction de x élevée à une certaine puissance α (généralement, une valeur négative).

L'unité lexicale et TAL

Le mot dans le corpus: la loi de puissance de zipf

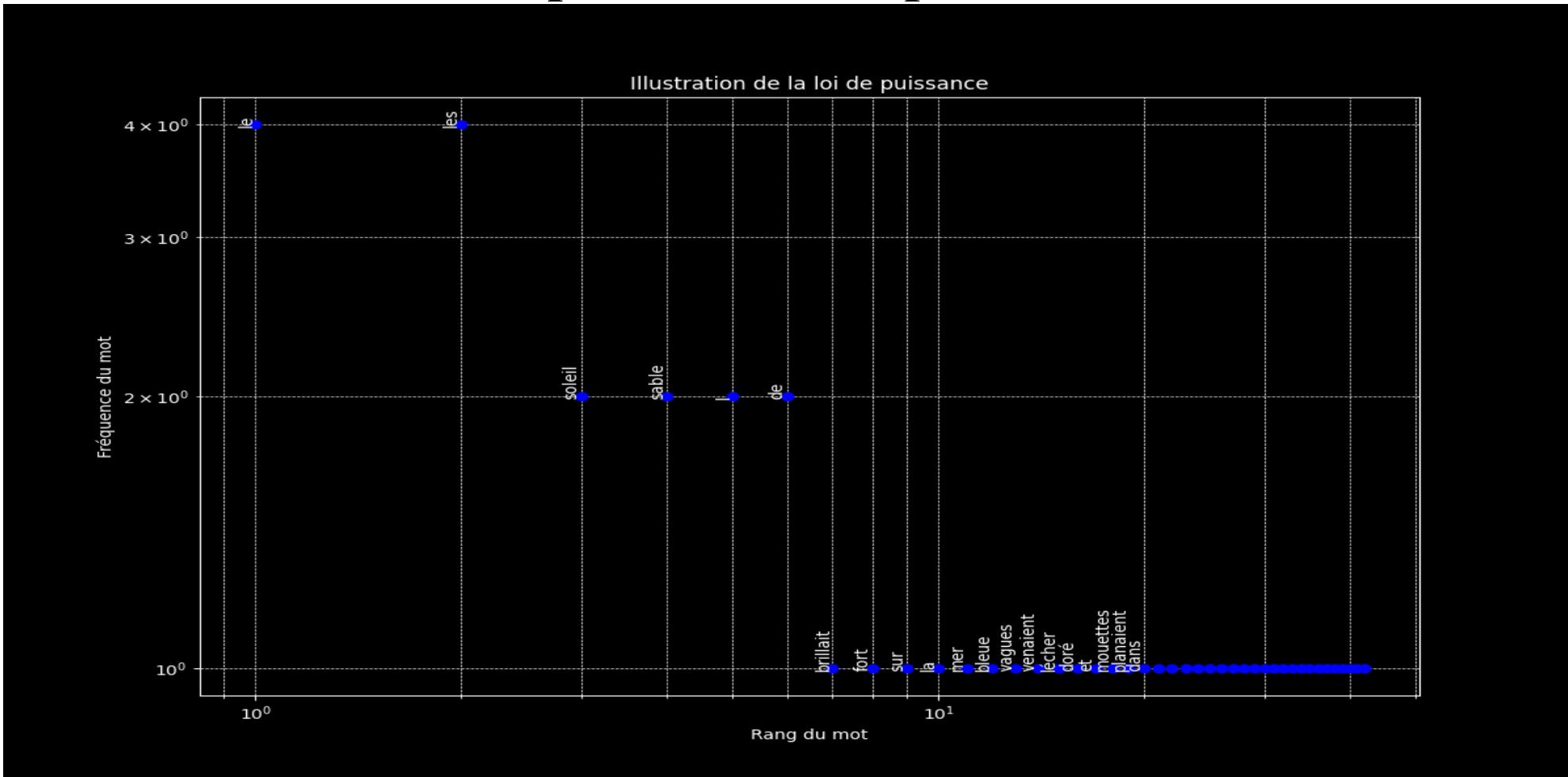
- Illustration de la loi de puissance de zipf:
 - Essayez de compter le nombre des *tokens* dans cette phrase (l'=le/la, du=de+le):

Le soleil brillait fort sur la mer bleue. Les vagues venaient lécher le sable doré, et les mouettes planaient dans le ciel. Le vent soufflait doucement, apportant avec lui la odeur salée de le océan. Les enfants jouaient en riant, construisant des châteaux de sable tandis que les adultes profitaient de le soleil.
 - Essayez de classer les *types* des différents *tokens* que vous observez dans les données
 - Classez les *types* selon le nombre d'occurrence de ses *tokens* afin d'obtenir les types ayant les 10 premiers rangs.
 - Est-ce que vous obtenez des résultats comme le suivant?

L'unité lexicale et TAL

Le mot dans le corpus: la loi de puissance de zipf

- Illustration de la loi de puissance de zipf:



L'unité lexicale et TAL

Le mot dans le corpus: la loi de puissance de Zipf

- Renseignement de la loi de puissance de zipf:
 - Dans un corpus de données, quelques mots ont des fréquences hautes alors que la majorité à une fréquence base.
 - Le mots les plus fréquents sont souvent des mots grammaticaux.
 - Dans plusieurs tâches automatiques (pas la totalité), nous avons besoin des mots lexicaux.
 - Dans un corpus issue d'une documentation linguistique, nous sommes souvent intéressés par les *types* de rangs intermédiaires (pas trop haut, pas trop bas).
 - Mais dans un corpus donné, la plupart de *types* auraient une seule manifestation i.e. *hapax legomenon* (pl. *hapax legomena*)
 - On peut avoir aussi quelques *dis legomena* (deux occurrences).

L'unité lexicale et TAL

Le mot dans le corpus: la loi de Heaps

- Renseignement de la loi de puissance de zipf:
 - Selon la *loi de Heaps*, plus un corpus est grand, plus nous avons la possibilité de rencontrer un maximum de *types*.
 - Autrement dit:
$$|V| = k * N^\beta$$

$|V|$ étant le nombre de types
 N étant le nombre de tokens
 k et β étant des constants qui dépend du corpus et de la langue
mais $0 < \beta < 1$
- Ainsi, nous avons intérêt à avoir un corpus assez conséquent (ceci est relatif)
- Astuce:
 - le taux de découvert d'un nouveau type ralenti avec l'élargissement du corpus.
 - Au moment où le taux de découvert est trop lent, on commence à penser que le corpus devrait satisfaire aux exigences de base

L'unité lexicale et TAL

Le mot dans le corpus: la loi de Heaps

- Exercice:

- Reprenez le texte de l'exercice précédent.
- Multipliez chaque *token* par dix (k dans la formule).
- Comptez le nombre de *types* (y) et le nombre de *tokens* (x)
- Après combien de *tokens* rencontrez vous un nouveau *type*?
- Reprenez la formule Heaps:

$$|V| = k * N^\beta$$

- Prenez les valeurs logarithmique de chaque coté:

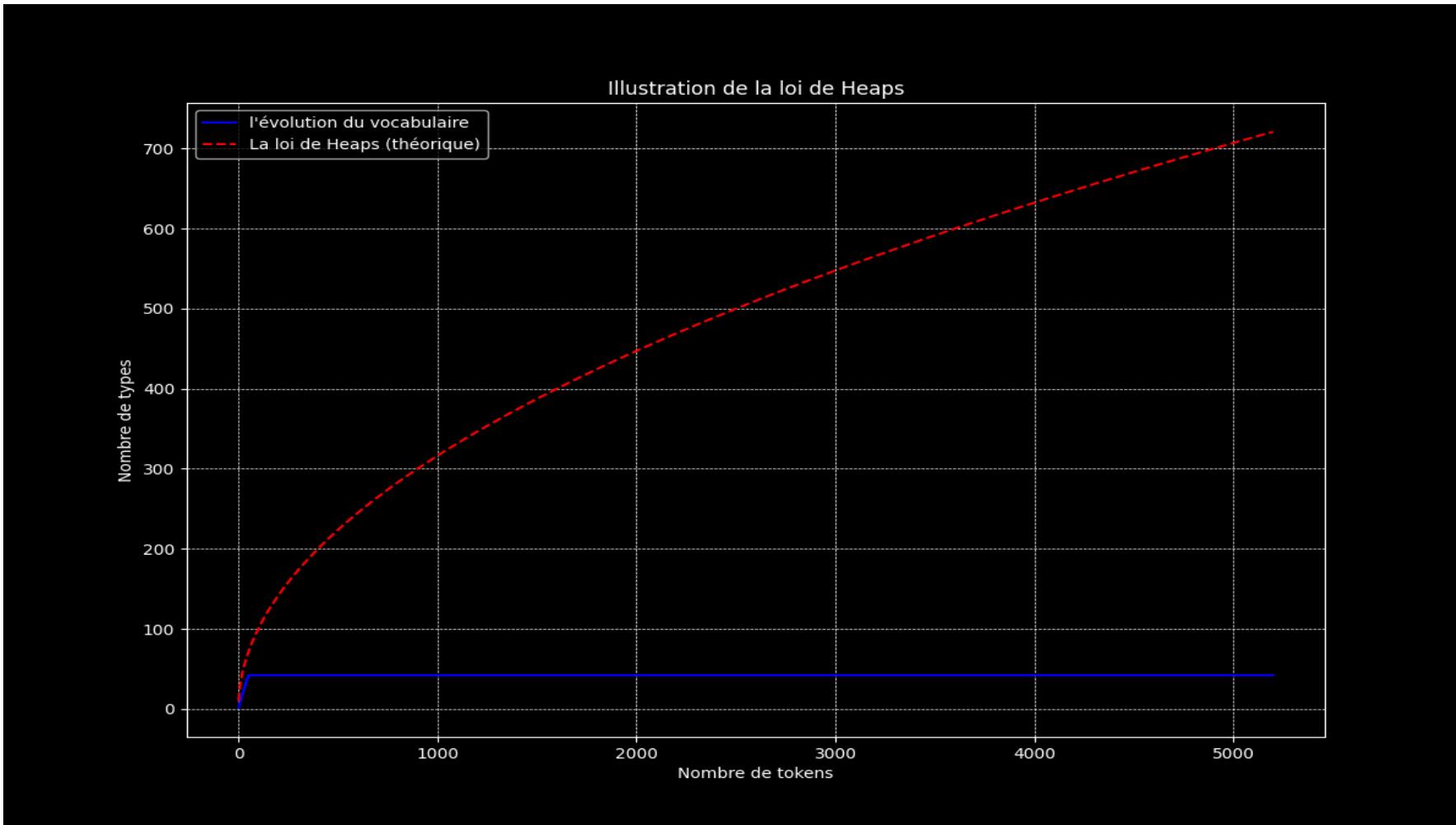
$$\log(|V|) = \log(k) + \beta * \log(N)$$

- Ceci ressemble enfin à la formule de la régression linéaire i.e. $y = a + b * x$; avec β qui indique la pente de la ligne de regression.
- Si vous avez un logiciel statistique, rentrez le y et le x pour avoir le paramètre β qui vous permet d'obtenir une courbe qui ressemble au suivant:

L'unité lexicale et TAL

Le mot dans le corpus: la loi de Heaps

- Exercice:



L'unité lexicale et TAL

Le mot dans le corpus: la loi de brièveté

- La loi de brièveté de zipf:
 - Plus un mot est fréquent, plus il est court (sons/syllabes)
 - Ceci est en adéquation avec le constat en linguistique fondamentale souvent appelée la ‘motivation économique’
 - Toutefois, en linguistique fondamentale, il y a parfois des débats entre la ‘motivation économique’ et la ‘motivation systémique’ i.e. pression du système linguistique (Haspelmath, 2024):

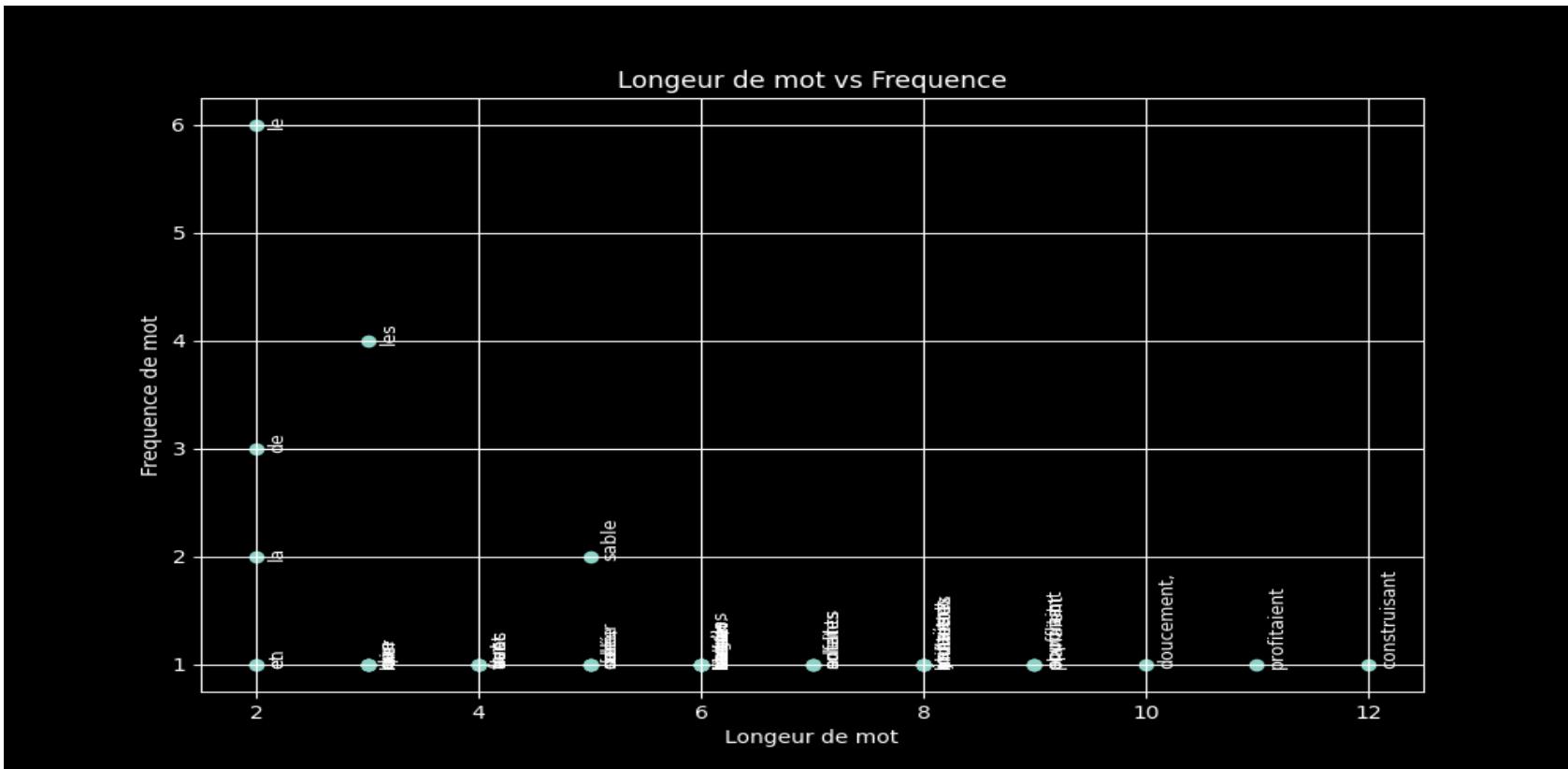
"Eyes" en anglais est environ trois fois plus fréquent que "eye", bien qu'il s'agisse d'une forme plus longue avec un suffixe supplémentaire "-s". Le cas de *eye/eyes* est donc inattendu d'un point de vue économique. De toute évidence, ce qui se passe ici, c'est que *eye/eye-s* se conforme au modèle majoritaire *book/books*, c'est-à-dire que la pression du système l'emporte sur l'économie (pp. 197)
 - Ainsi, la loi de brièveté ne doit pas être prise dans des termes absolus, mais plutôt en des termes tendanciels.

L'unité lexicale et TAL

Le mot dans le corpus: la loi de brièveté

- Exercice:

- Prenez le texte de l'exercice précédent.
- Comptez le nombre de *types* et *tokens*.
- Qu'est-ce que vous observez?
- comment expliquez-vous les mots qui ne suivent pas la tendance?



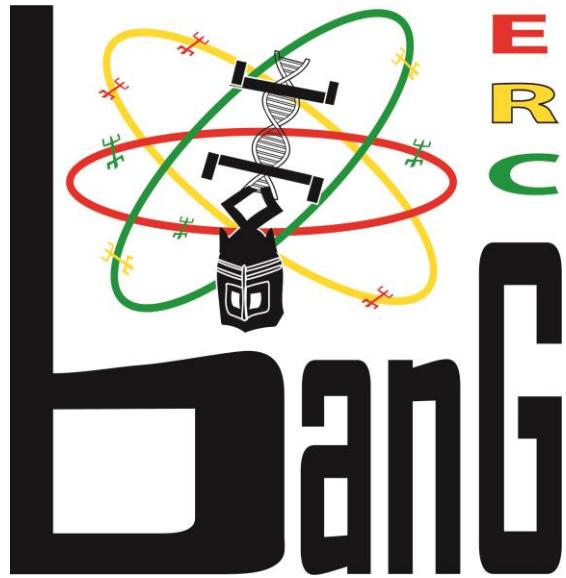
Bibliographie

- Bellman, R., & Kalaba, R. (1957). Dynamic programming and statistical communication theory. *Proceedings of the National Academy of Sciences*, 43(8), 749-751.
- Dickinson, M., Glass, L., Brew, C., & Meurers, D. (2024). *Language and computers*. BoD–Books on Demand.
- Haspelmath, M. (2014). On system pressure competing with economic motivation. *Competing motivations in grammar and usage*, 197-208.
- Jurafsky, D., & Martin, J. H. (2008). Speech and Language Processing: An introduction to speech recognition, computational linguistics and natural language processing. *Upper Saddle River, NJ: Prentice Hall*.
- Wagner, R. A., & Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1), 168-173.



LINGUISTIQUE COMPUTATIONNELLE

COURS 3



Promise Dodzi Kpoglu
promisedodzi@gmail.com / promise-dodzi.kpoglu@cnrs.fr

UCAD, Paris.
Décembre, 2024.



Plan du cours

- Métriques de similarité (lexicales)---- cours 3
 - Jaccard
 - Sorensen-Dice
 - Hamming
 - Levenshtein

Métriques de similarité

Introduction

- Admettons que vous avez une liste de mots de votre corpus.
- Vous voulez savoir certaines choses:
 - A quel point un mot de votre corpus ressemble à d'autres mots du corpus?
 - A quel point les mots d'un texte du corpus ressemblent aux mots d'un autre texte du corpus i.e. à quel point texte A et texte B se ressemblent
- Effectivement, on commence par mesurer la similarité entre les mots.
- Par quels procédés pensez-vous que nous pouvons mesurer la similarité entre les mots suivants?

manger vs *danser*

danser vs *danse*

nation vs *internationalité*

partir vs *gouverner*

- Plusieurs métriques permettent d'obtenir des résultats assez intéressants
 - Lexicales/sémantiques/graphiques etc.
- Nous allons nous limiter à 1 type de métrique dans ce cours:
 - Des métriques lexicales

Métriques de similarité

les métriques lexicales

- Les métriques lexicales s'appuient sur les graphèmes, d'une manière ou d'une autre, dans deux mots, pour déterminer leur similarité.
v-e-n-i-r vs *s-o-r-t-i-r*
- On peut distinguer trois types de métriques lexicales:
 - Les métriques basées sur des séquences (Sequence-based)
 - **Les métriques basées sur des *tokens* (token-based)**
 - **Les métriques basées sur édition (Edit-based)**
- Chacune de ces métriques est calculée en se basant sur des algorithmes spécifiques
 - Nous allons regarder seuls les deux derniers types de métriques lexicales

Métriques de similarité

les métriques basées sur des tokens

- Ces métriques sont obtenues en faisant des calculs sur les graphèmes des *tokens*.

v-e-n-i-r

- Nous allons étudier deux de ces algorithmes:
 - La similarité Jaccard
 - La similarité Sorensen-Dice (utilisée pour amplifier les similitudes)
- Pour la similarité Jaccard, il s'agit de diviser l'intersection des n-grams dans les deux mots, par l'union de ceux-ci.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

- Pour la similarité Sorensen-Dice (souvent appelé le coefficient Dice), l'intersection des n-grams dans les deux mots est multipliée par 2.

$$D(A,B) = \frac{2 * |A \cap B|}{|A| + |B|}$$

Métriques de similarité

les métriques basées sur des tokens

- Illustration avec deux mots *maisons* et *maisonnettes*

- Similarité Jaccard:

set A = {m,a,i,s,o,n}
set B = {m,a,i,s,o,n,e,t}

$$A \cap B = \{m, a, i, s, o, n\} \Rightarrow 6 \text{ graphèmes}$$
$$A \cup B = \{m, a, i, s, o, n, e, t\} \Rightarrow 8 \text{ graphèmes}$$

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{6}{8} = 0,75$$
$$= 0,75 * 100 = 75\% \text{ de similarité}$$

- Similarité Sorensen-Dice:

set A = {m,a,i,s,o,n} \Rightarrow 6 graphèmes
set B = {m,a,i,s,o,n,e,t} \Rightarrow 8 graphèmes

$$A \cap B = \{m, a, i, s, o, n\} \Rightarrow 6 \text{ graphèmes}$$

$$D(A, B) = \frac{2 * |A \cap B|}{|A| + |B|} = \frac{2 * 6}{6 + 8} = 0,86$$
$$= 0,86 * 100 = 86\% \text{ de similarité}$$

- Exercice: pouvez-vous calculer ces deux métriques en utilisant des 2-grams? i.e. {'ma', 'am', etc.}.

Métriques de similarité

les métriques basées sur édition

- NB: Nous allons nous servir de l'une de ces métriques dans la partie pratique.
- Les algorithmes qui aident à calculer ces métriques se basent sur les transformations permettant d'obtenir un mot B à partir d'un mot A
bon vs. *son*
- Si j'ai le mot *bon*, comment pourrais-je faire pour obtenir le mot *son*?
- Nous allons étudier deux algorithmes qui scorent une telle opération:
 - La distance Hamming
 - La distance d'Edition
- La distance Hamming: les différences dans deux mots d'une même taille.

$$d = \sum_{i=1}^n \delta(A_i, B_i)$$

A_i et B_i sont des graphèmes dans les positions i dans les deux mots
 $\delta(A_i, B_i)$ est une fonction qui est 1 si $A_i \neq B_i$ et 0 si $A_i = B_i$.

Métriques de similarité

les métriques basées sur édition

- Distance Hamming:

bonheur vs. honneur

b	o	n	j	o	u	r
h	o	n	n	e	u	r

Longeur de A = $\text{len}([b, o, n, j, o, u, r]) = 7$

Longuer de B = $\text{len}([h, o, n, n, e, u, r]) = 7$

Indexes de différences = $\text{len}([0, 3]) = 2$

Distance Hamming = $\frac{2}{7} = 0,29$

univers vs. université

u	n	i	v	e	r	s			
u	n	i	v	e	r	s	i	t	é

Longeur de A = $\text{len}([u, n, i, v, e, r, s]) = 7$

Longuer de B = $\text{len}([u, n, i, v, e, r, s, i, t, é]) = 10$

Distance Hamming =!!!!!! (longeurs différentes)

Métriques de similarité

les métriques basées sur édition

- La distance d'Edition: il s'agit de l'ensemble des opérations permettant d'obtenir un mot B à partir d'un mot A.
- Les opérations le plus souvent comptabilisées sont:
 - L'insertion des graphèmes
 - La suppression des graphèmes
 - La substitution des graphèmes
 - Dans d'autres versions, on considère aussi la transposition des graphèmes – nous n'allons pas le considérer dans la version que nous allons étudier.
- La distance est obtenue par une division du nombre d'opérations par la longueur du mot le plus long.

Métriques de similarité

les métriques basées sur édition

I	N	T	E	*	N	T	I	O	N	
*	E	X	E	C	U	T	I	O	N	
Sup	Sub	Sub		Ins	Sub					

= distance d'édition = 5 / 9

- Nous pouvons donner du poids à chaque opération selon les données:
 - Si nous ne voulons pas accepter des substitutions
 - Un poids de 2 pour une opération de substitution donne un score de 8 pour les deux mots *intention* et *exécution*.
- Dans ce cours, nous n'utiliserons pas de version pondérée.

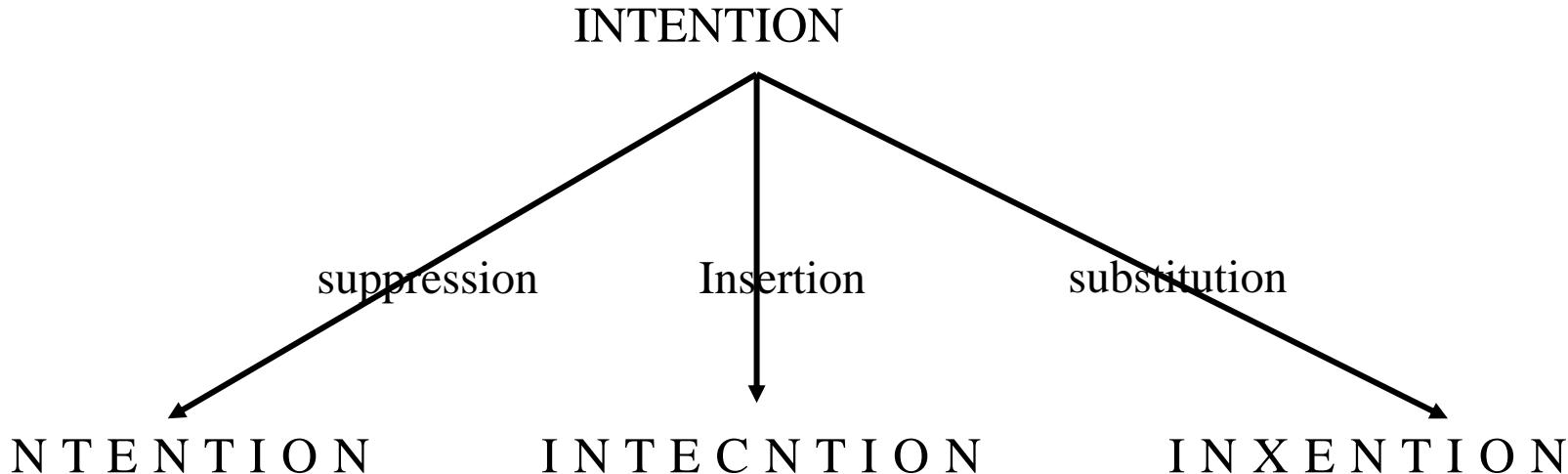
Un algorithme

l'algorithme de distance d'édition (distance Edit)

- Nous savons comment fonctionne la distance Edit
- Comment faire en sorte qu'un ordinateur puisse aussi calculer cette distance pour deux mots?
- On peut se servir d'un algorithme!
- Un **algorithme** est un ensemble d'instructions et d'opérations permettant de résoudre de problèmes.
- Autrement dit, nous devons donner des instructions accomplissant les opérations d'édition (algorithme de calcul de distance d>Edit) à l'ordinateur afin qu'il puisse calculer la distance Edit automatiquement.

Un algorithme

l'algorithme de distance d'édition (distance Edit)



- Une manière de concevoir l'algorithme c'est de parcourir les résultats de toutes les opérations afin de chercher un chemin qui permet d'obtenir EXECUTION.
- Ensuite, on calcule le nombre d'opérations comptabilisées, pour obtenir le score.
- Problème: une telle recherche + calcul sera trop onéreux car trop de candidats à parcourir

Un algorithme

l'algorithme de distance d'édition (distance Edit)

- Plutôt que d'utiliser l'approche brute et difficile à implémenter, nous pouvons utiliser une approche diviser-pour-régner: la recherche de la voie la plus rapide est divisée en des sous-tâches.
- Ce genre d'approche est appelée en informatique la programmation dynamique (Bellman 1957)
- ...et cette approche a été adoptée par Wagner and Fischer (1974) pour coder l'algorithme de détection de la distance Edit (minimale) entre deux mots.
- Toute l'idée repose sur la construction d'une matrice, pour ensuite insérer dans chaque cellule de la matrice un graphème des deux mots.
- En parcourant la matrice pour calculer la distance entre chaque sous-forme, on obtient la distance Edit minimale dans la cellule qui représente la dernière colonne et la dernière ligne

Un algorithme

l'algorithme de distance d'édition (distance Edit)

		I	N	T	E	N	T	I	O	N
	0	1	2	3	4	5	6	7	8	9
E	1									
X	2									
E	3									
C	4									
U	5									
T	6									
I	7									
O	8									
N	9									

- Première étape: numérotter chaque graphème selon son indice
- Le 0 représente le cas où il n'y a pas de graphème
 - Si il y a un mot 0 et le mot EXÉCUTION, il faudra 9 opérations (insertions) pour changer 0 en EXECUTION.

Un algorithme

l'algorithme de distance d'édition (distance Edit)

		E	X	E	C	U	T	I	O	N
	0	1	2	3	4	5	6	7	8	9
I	1	1								
N	2	2								
T	3	3								
E	4	3								
N	5	4								
T	6	5								
I	7	6								
O	8	7								
N	9	8								

- Deuxième étape: calculer la distance entre sous-ensembles
- Distance (I, E)=1; Distance (IN, E)=2; Distance (INT, E)=3; Distance (INTE, E); Distance (INTEN, E)=4; Distance (INTENT, E); Distance (INTENTI, E)=6
- Distance (I, EX)=2, Distance (I, EXE)=3, Distance (i, exec)=4, Distance (i, execu)=5, Distance (I, EXECUT)=6, Distance (I, EXECUTI)=6, Distance (I, EXECUTIO)=7, Distance (I, EXECUTION)=9

Un algorithme

l'algorithme de distance d'édition (distance Edit)

- Exercice:
 - Pouvez-vous remplir la matrice avec le reste des distances calculées?

Un algorithme

l'algorithme de distance d'édition (distance Edit)

		E	X	E	C	U	T	I	O	N
	0	1	2	3	4	5	6	7	8	9
I	1	1	2	3	4	5	6	6	7	8
N	2	2	2	3	4	5	6	7	7	7
T	3	3	3	3	4	5	5	6	7	8
E	4	3	4	3	4	5	6	6	7	8
N	5	4	4	4	4	5	6	7	7	7
T	6	5	5	5	5	5	5	6	7	8
I	7	6	6	6	6	6	6	5	6	7
O	8	7	7	7	7	7	7	6	5	6
N	9	8	8	8	8	8	8	7	6	5

- Distance Edit = 5

Un algorithme

l'algorithme de distance d'édition (distance Edit)

- Si on observe le tableau rempli plus attentivement on verra un schéma:
 - En supposant que nous accordons un coût de 1 pour chaque divergence et 0 pour chaque accord
 - On constatera que chaque score est le résultat du minimum ([ligne $i - 1$, colonne j]+1, [ligne i , colonne $j - 1$]+1, [ligne $i - 1$, colonne $j - 1$]+coût)

	$j=0$	$b_{j=1}$	$a_{j=2}$	$i_{j=3}$	$l_{j=4}$
$i=0$	0	1	2	3	4
$m_{i=1}$	1	1	2		
$a_{i=2}$	2	1			
$i_{i=3}$	3				
$s_{i=4}$	4				

$$\text{edit}(m, b) = \min (1+1, 1+1, \underline{0+ \text{coût}=1}) = 1$$

$$\text{edit } (m, ba) = \min(2+1, \underline{1+1}, \underline{1+\text{coût}=1}) = 2$$

$$\text{edit } (ma, ba) = \min(1+1, 2+1, \underline{1+\text{coût}=0}) = 1$$

- Pouvez-vous remplir le reste des cellules?

Un algorithme

l'algorithme de distance d'édition (distance Edit)

- Pour résumer les étapes du calcul de la distance Edit:
 - Construire une matrice
 - La cellule une et deux de la première colonne doivent être vides
 - Arranger les graphèmes du mot A un par un dans le reste des cellules de la première colonne
 - Remplir les cellules juste en dessous avec les indices de chaque graphème
 - Arranger les graphèmes du mot B un par un dans le reste des cellules de la première ligne
 - Remplir les cellules juste à coté avec les indices de chaque graphème
 - Assigner le coût de dissimilarité (1 pour dissimilarité, 0 pour similarité)
 - Remplir chaque cellule en utilisant la formule:
minimum ([ligne i -1,colonne j]+1, [ligne i , colonne j-1]+1, [ligne i-1, colonne j-1]+coût)
 - La distance Edit (minimum) c'est le score dans la dernière cellule en bas à droit.

Un algorithme

l'algorithme de distance d'édition (distance Edit)

FONCTION edit_distance(str1, str2):

 m ← LONGEUR(str1)

 n ← LONGEUR(str2)

 CREER dp[0...m][0...n] # Step 1: Initialiser la matrice dp

 POUR i DE 0 A m:

 dp[i][0] ← i

 # Initialiser la première colonne

 POUR j DE 0 A n:

 dp[0][j] ← j

 # Initialiser la première ligne

 POUR i DE 1 A m: # Step 2: remplir la matrice dp

 POUR j DE 1 A n:

 SI str1[i-1] == str2[j-1]:

 coût ← 0

 SINON:

 coût ← 1

 dp[i][j] ← MIN(dp[i-1][j] + 1, dp[i][j-1] + 1, dp[i-1][j-1] + cost)

RETOURNER dp[m][n] # Step 3: Retourner la distance Edit minimum

Un algorithme

l'algorithme de distance d'édition (distance Edit)

- Prochain cours:
 - Nous allons utiliser cette algorithme pour construire un autre algorithme pour faire une analyse linguistique

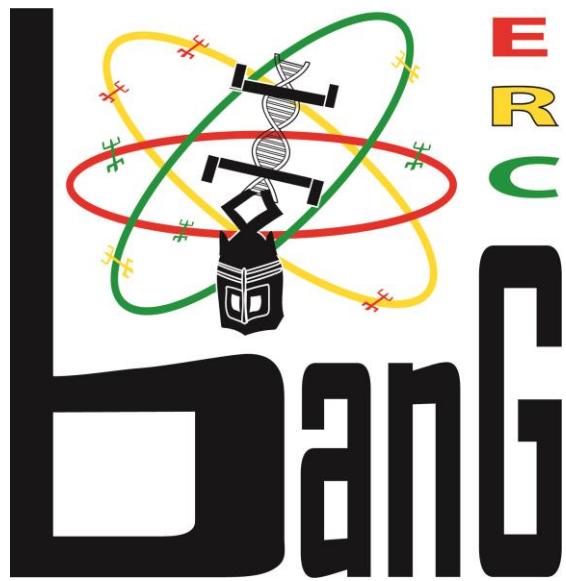
Bibliographie

- Bellman, R., & Kalaba, R. (1957). Dynamic programming and statistical communication theory. *Proceedings of the National Academy of Sciences*, 43(8), 749-751.
- Dickinson, M., Glass, L., Brew, C., & Meurers, D. (2024). *Language and computers*. BoD–Books on Demand.
- Haspelmath, M. (2014). On system pressure competing with economic motivation. *Competing motivations in grammar and usage*, 197-208.
- Jurafsky, D., & Martin, J. H. (2008). Speech and Language Processing: An introduction to speech recognition, computational linguistics and natural language processing. *Upper Saddle River, NJ: Prentice Hall*.
- Wagner, R. A., & Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1), 168-173.



LINGUISTIQUE COMPUTATIONNELLE

COURS 4: DETECTION DE COGNATS - APPROCHE ASSISTÉE PAR ORDINATEUR



Promise Dodzi Kpoglu
promisedodzi@gmail.com / promise-dodzi.kpoglu@cnrs.fr

UCAD, DAKAR.
Décembre, 2024.



Plan du cours

- Les cognats
- Les algorithmes de détection de cognats
- Codage et analyse

Cognats

Introduction

- Les cognats sont des mots ayant un ancêtre commun
esp. *familia* vs: fr. *famille* origine commune = lat. 'familia'
- Les cognats peuvent être:
 - Des cognats entiers = phonétiquement fort, sémantiquement similaire ex. Ger. *herz* vs. Eng. *heart* ‘cœur’
 - Des cognats partiels = moins de la moitié des graphèmes phonétiques participent dans la relation de cognacité; similarité sémantique peut être aussi partielle.
ex. Ger. *walfisch* vs. Eng. *whale* ‘requin’
 - Des cognats opaques (dialexèmes): dissimilarité phonétique et sémantique mais ancêtre commun. Ex: Albanien. *gardh* ‘cour’ vs Romani. *kher* “famille” == Proto Indo-Européen **g^herd^h-* ‘enfermer’

Algorithmes de detection automatique de cognats

- Types d'algorithmes qui permettent de détecter des cognats:
 - des algorithmes à base de règles
 - des algorithmes à base des distances atomiques
 - des algorithmes sémantiques ----- plongements lexicaux (vecteurs similaires=probabilité de cognacité)
 - des algorithmes d'apprentissage automatique (supervisé ou non-supervisé; approches bayésiennes, d'alignement etc.)
 - des algorithmes d'apprentissage profonde (supervisé ou semi-supervisé; réseaux neuronal siamois, Transformers etc.)
- Dans cet atelier, nous allons essayer de coder un modèle à base des distances atomiques

Algorithmes de detection automatique de cognats

les algorithms à base de règles

- Les algorithmes à base de règles:
 - Propice pour des linguistes ayant une maîtrise des propriétés linguistiques de la langue
 - On code les règles dans un langage de programmation
 - On applique ces règles à un ensemble de données
- Inconvenance:
 - le champ d'application d'un tel algorithme est restreint
 - s'applique difficilement à des données différentes de celles sur lesquelles on a travaillées

Algorithmes de detection automatique de cognats

les algorithmes à base des distances atomiques

- Algorithmes à base des distances atomiques
- Des calculs de distance sont à la base de la détection des cognats
- Métriques de distance:
 - Levenshtein (Edit),
 - Hamming (nombre de différences),
 - Jaccard ($A \text{ intersection } B / A \cup B$),
 - Longest Common Substring(taille de sous-segment partagé) etc.*
- A la suite de ceci, les mots ayant une forte similarité sont considérés comme étant des cognats

Algorithmes de detection automatique de cognats

les algorithmes à base des distances atomiques

- Souvent on restreint le calcul aux mots ayant le même sens
- Ainsi, les données requises pour ces algorithmes sont à préparer selon des critères bien spécifiques.
- Modèles
 - Le calcul de distance peut être fait sur la forme orthographique ou phonétique des mots
 - Ceci veut dire que ces algorithmes peuvent être soit d'un ordre orthographique soit phonétique
 - Le calcul peut aussi se faire sur des mots convertis en "classes", donnant ainsi un troisième type de modèle
 - Enfin, on peut avoir un modèle qui est une combinaison de l'un des modèles cités plus haut
 - Pourquoi la sophistication est-elle souhaitée: en réduisant les mots à leur forme phonétique/léxématique par exemple, on s'assure que la phonétique/morphologie n'empêche pas de détecter des cognats, ou ne permet pas de détecter des faux cognats.

Algorithmes de detection automatique de cognats

les algorithms à base des distances atomiques

- Dans cet atelier:
 - Langage de programmation: Python
 - un model très basique et simple: un modèle orthographique
 - Distance à intégrer: Levenshtein
 - Pas d'intégration morphologique, pas de pondération avec d'autres modèles
 - Mais, si nous avons le temps, nous allons essayer d'y intégrer des probabilités à priori (connaissances de la linguistique typologique)
 - L'idée c'est de montrer comment on peut intégrer les connaissances linguistiques dans un modèle aussi simple, afin d'avoir des résultats plus probants

Les codes

- Dans Google, saisissez ‘Google colab’
- Cliquez sur le premier lien
- En bas, à gauche de la sous-fenêtre, cliquez sur ‘new notebook’
- En haut à gauche, cliquez sur ‘file’ ou ‘fichier’ en français
- Défilez en bas et cliquez sur ‘upload notebook’ ou ‘télécharger notebook’
- Localisez le fichier ‘cognats_orthographic.ipynb’ sur votre ordinateur pour l’ajouter
- Ensuite, cliquez sur l’icone des dossiers à gauche de l’écran
- Cliquez sur l’icone de téléchargement
- Localisez le fichier ‘cognate_data_1.csv’ pour le charger dans votre environnement de travail