

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Instituto de Informática
Departamento de Informática Aplicada

Aula 3: Manipulação e Comparação de Textos:

Expressões Regulares, Pré-processamento de Texto, Distância Mínima de Edição

Prof. Dennis Giovani Balreira



INF01221 - Tópicos Especiais em Computação XXXVI:
Processamento de Linguagem Natural



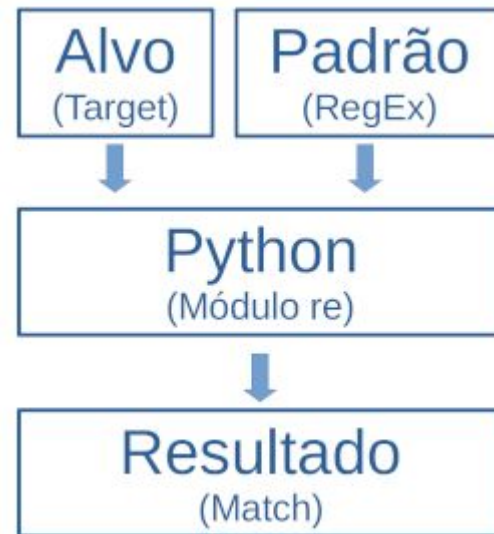
Conteúdo

- Expressões regulares
- Pré-processamento de texto:
 - Segmentação de sentenças
 - Normalização:
 - Correção ortográfica
 - Case folding (conversão para minúsculas)
 - Remoção de acentos, caracteres especiais e pontuação
 - Expansão de contrações
 - Remoção de stop words
 - Lematização
 - Stemming
 - Tokenização
- Distância mínima de edição

Expressões regulares

Expressões regulares (regex)

- Padrões utilizados para encontrar e manipular texto de maneira eficiente
- Permitem especificar buscas complexas em cadeias de caracteres de uma forma compacta e flexível
- Aplicações:
 - Validação de formulários
 - Processamento de arquivos de logs
 - Extração de dados
- Sua definição possui variações
 - Na disciplina serão estudadas as expressões regulares estendidas (básicas e avançadas) [Jurafski,2024]



Expressões regulares: Operadores básicos

- 1. Todas expressões regulares possuem formato:
 /**<str>**/ , onde **<str>** é uma string a ser buscada
 - Ex: /abc/ (procura por todos os abc em um texto)
- 2. Expressões regulares são case sensitive
 - Ex: /Ana/ e /ana/ (textos distintos)
- 3. Concatenação: AB
 /**<c1><c2>...<cN>**/ , onde **<cK>** é um caractere
 - Ex: /arroz/ é a junção dos caracteres 'a', 'r', 'r', 'o', 'z'

Expressões regulares: Operadores básicos

- 4. Disjunção: $[AB]$

$/[\langle c1 \rangle \langle c2 \rangle \dots \langle cN \rangle]/$, onde $\langle cK \rangle$ é um caractere

- Ex: $/[0123456789]/$ (um único dígito de qualquer tipo)
 $/[aA]rroz/$ (string “Arroz” ou “arroz”)

- 5. Intervalo: $A-B$

$/[\langle c1 \rangle - \langle c2 \rangle]/$, onde $\langle cK \rangle$ é um caractere

- Ex: $/[0-9]/$ (único dígito de qualquer tipo)
 $/[a-z]/$ (única letra minúscula)

- 6. Negação: A

$/[^{\langle c1 \rangle} \dots \langle cN \rangle]/$, onde $\langle cK \rangle$ é um caractere

- Ex: $/[^A-Z]/$ (não é uma letra maiúscula)
 $/[^Aa]/$ (não é ‘A’ nem ‘a’)

Expressões regulares: Operadores básicos

- 7. Opcionalidade: $A?$

$/\langle c1 \rangle?/$, onde $\langle cK \rangle$ é um caractere

- Ex: $/alunos?/$ (aluno ou alunos)
 $/bot?a/$ (boa ou bota)

- 8. Kleene *: A^* zero ou mais ocorrências

$/\langle c1 \rangle^*/$, onde $\langle cK \rangle$ é um caractere

- Ex: $/ba^*/$ (“b!”, “ba!”, “baa!”, ...)
 $/[0-9][0-9]^*/$ (números inteiros)

- 9. Kleene +: A^+ uma ou mais ocorrências

$/\langle c1 \rangle^+/$, onde $\langle cK \rangle$ é um caractere

- Ex: $/ba^+!/$ (“ba!”, “baa!”, ...)
 $/[0-9]^+!/$ (números inteiros)

Expressões regulares: Operadores básicos

- 10. Curinga (“Wildcard”): `.` assume qualquer caractere
 - Ex: `/fa.a/` (faca, fada ou fala)
`/não.*não/` (detecta dois “não” seguidos)
- 11. Caracteres especiais: `\<ck>`
- 12. Âncoras: `^`, `$`, `\b`, `\B` delimitam locais na string
 - `^`: início da string
 - Ex: `/^O/` (frases que começam com ‘O’)
 - `$`: fim da string
 - Ex: `/\./` (frases que terminam com ‘.’)
 - `\b`: fronteira de palavra
 - Ex: `/\bgato\b/` (palavra “gato”)
 - `\B`: não-fronteira de palavra
 - Ex: `/\Brato\B/` (subpalavra “rato”)

Expressões regulares: Operadores avançados

- 1. Disjunção: |
 - Ex: /arroz|feijão/ (string “arroz” ou “feijão”)
- 2. Precedência: ()
 - Ex: /pré\-(pago|definido)/
- 3. Ordem de precedência:

Parenthesis	()
Counters	* + ? {}
Sequences and anchors	the ^my end\$
Disjunction	

Expressões regulares: Atalhos

Pattern	Expansion	Matches	Examples
<code>\d</code>	<code>[0-9]</code>	Any digit	Fahre ^{ne} it <u>4</u> 51
<code>\D</code>	<code>[^0-9]</code>	Any non-digit	<u>B</u> lue Moon
<code>\w</code>	<code>[a-zA-Z0-9_]</code>	Any alphanumeric or _	<u>D</u> aiyu
<code>\W</code>	<code>[^\w]</code>	Not alphanumeric or _	Look <u>!</u>
<code>\s</code>	<code>[\r\t\n\f]</code>	Whitespace (space, tab)	Look <u> </u> up
<code>\S</code>	<code>[^\s]</code>	Not whitespace	<u>L</u> ook up

Expressões regulares: Processo de criação

- Ao codificar **expressões regulares**, é importante focar em **dois tipos de erros**:
 - **1. Não encontrar strings que deveriam ter sido encontradas**
Falsos Negativos: deu “negativo” (não encontrado) e está “falso” (errado)
Ex: Na procura por ocorrências de “gato” não encontrar “Gato”
Solução: aumentar o “recall”*
 - **2. Encontrar strings que não deveriam ter sido encontradas**
Falsos Positivos: deu “positivo” (encontrado) e está “falso” (errado)
Ex: Na procura por ocorrências de “gato” encontrar “gatorade”
Solução: aumentar a “precision”*

*Estes conceitos são estudados em Aprendizado de Máquina e serão trabalhados futuramente na disciplina

Expressões regulares: Importância

- As expressões regulares desempenham um papel surpreendentemente grande
 - Amplamente usadas tanto na academia quanto na indústria
- Motivos principais:
 1. Fazem parte da maioria das tarefas de **processamento de texto**, até mesmo em pipelines de modelos de linguagem neural
 - Incluem formatação de texto e pré-processamento
 2. São muito úteis para **análise de dados textuais**

Expressões regulares: Aplicação ELIZA

- Sistema inicial de PNL que imitava um psicoterapeuta rogeriano
 - Joseph Weizenbaum, 1966.
- Usa correspondência de padrões para combinar termos, por exemplo:
 - Ex: "Eu preciso de X" traduz para
"O que significaria para você se você conseguisse X?"

- Exemplo:

s/. * I'M (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/

s/. * I AM (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/

s/. * all .*/IN WHAT WAY?/

s/. * always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE?/



Expressões regulares: Exemplos

- **Exercício 1:** Crie uma expressão regular para encontrar todos os nomes próprios em um texto. Considere que todos os nomes próprios começam com letra maiúscula e são seguidos por letras minúsculas.
Texto de exemplo: "Ana e João foram ao parque com Carlos."
- **Exercício 2:** Escreva uma expressão regular para identificar números de telefone brasileiros no formato (XX) XXXXX-XXXX.
Texto de exemplo: "Você pode ligar para (21) 98765-4321."

Expressões regulares: Exemplos

- **Exercício 1:** Crie uma expressão regular para encontrar todos os nomes próprios em um texto. Considere que todos os nomes próprios começam com letra maiúscula e são seguidos por letras minúsculas.

Texto de exemplo: "Ana e João foram ao parque com Carlos."

Resposta: `\b[A-Z][a-z]*\b/`

- **Exercício 2:** Escreva uma expressão regular para identificar números de telefone brasileiros no formato (XX) XXXXX-XXXX.

Texto de exemplo: "Você pode ligar para (21) 98765-4321."

Resposta: `^([0-9][0-9]) [0-9][0-9][0-9][0-9][0-9]\-[0-9][0-9][0-9][0-9]/`

(Alternativa): `^(\d{2}) \d{5}-\d{4}/`

Material complementar

- Bibliotecas em python sobre expressões regulares:
 - <https://docs.python.org/3/library/re.html>
 - <https://pypi.org/project/regex/>
- Ferramentas online para testar regex:
 - <https://regex101.com/>
 - <https://regexr.com/>
- Pattern Matching with Regular Expressions in Python (automatetheboringstuff):
 - <https://automatetheboringstuff.com/2e/chapter7/>
- Tutoriais regex:
 - <https://realpython.com/regex-python/>
 - https://www.w3schools.com/python/python_regex.asp
 - <https://www.programiz.com/python-programming/regex>

Pré-processamento de texto

Pré-processamento de texto

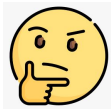
- Técnicas aplicadas ao texto bruto para transformá-lo em sequências bem definidas de componentes linguísticos, proporcionando uma estrutura e notação padrão
 - Ajudam a limpar, normalizar e estruturar o texto
 - Com exceção da Tokenização, as demais etapas são “opcionais”!
 - Dependem da tarefa e do tipo de modelo usado
- Etapas:
 - (1) Segmentação em sentenças
 - (2) Normalização:
 - Correção ortográfica
 - Case folding (conversão para minúsculas)
 - Remoção de acentos, caracteres especiais e pontuação
 - Expansão de contrações
 - Remoção de stop words
 - Lematização
 - Stemming
 - (3) Tokenização

Pré-processamento de texto: (1) Segmentação em sentenças

- Processo de **dividir um texto em sentenças** (unidades básicas de significado)
- Passo importante, pois permite análise mais granular de um texto em unidades menores e mais gerenciáveis
- Envolve a identificação de delimitadores de sentença:
 - Ponto final
 - Interrogação
 - Exclamação

...

Problemas?

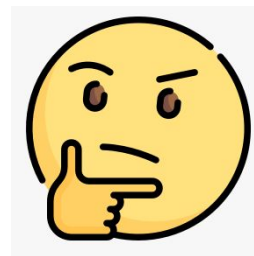


Pré-processamento de texto: (1) Segmentação em sentenças

- Processo de **dividir um texto em sentenças** (unidades básicas de significado)
- Passo importante, pois permite análise mais granular de um texto em unidades menores e mais gerenciáveis
- Envolve a identificação de delimitadores de sentença:
 - Ponto final
 - Interrogação
 - **Abreviações e siglas**: “Dr.”, “Sr.”, “U.S.A”
 - Exclamação
 - **Pontuações múltiplas**: “Ela disse: ‘O quê? Não acredito!’”
 - **Idiomas diferentes**:
 - ... Em Tailandês:
 - Problemas? Texto original: "วันนี้ฉันไปตลาดฉันซื้อมะม่วงมาเยอะมากแม่บอกให้ทำขนมไทย"
(Wan nī chān pī tlād chān sūx māmwng mā yexa māk mǎe bōk hī thǎ khnm thīy)
 - Tradução: "Hoje eu fui ao mercado. Comprei muitas mangas. Minha mãe disse para fazer uma sobremesa tailandesa."

Pré-processamento de texto: (2) Normalização

- Conjunto de técnicas usadas para **padronizar o texto**, eliminando variações e inconsistências que possam afetar a análise posterior em tarefas de PLN
- As etapas são **opcionais!**
 - Seu uso depende da tarefa
 - Nem todas tarefas são/precisam ser aplicadas (lemma vs. stem)
- Estudaremos as seguintes etapas:
 - 1. Correção ortográfica
 - 2. Case folding (conversão para minúsculas)
 - 3. Remoção de acentos, caracteres especiais e pontuação
 - 4. Expansão de contrações
 - 5. Remoção de stop words
 - 6. Lematização
 - 7. Stemming



Pré-processamento de texto: (2) Normalização

- 1. Correção ortográfica
 - Identifica e corrige erros de digitação ou ortografia no texto
 - Exemplo: "tecnlogia" → "tecnologia"
 - Ferramentas: Dicionários ortográficos e corretores automáticos, como o PySpellChecker em Python
- 2. *Case folding* (conversão para minúsculas)
 - Converte todas as letras do texto para minúsculas
 - Exemplo: "Casa" → "casa"
 - Ferramentas: Funções nativas de manipulação de strings em Python, como `.lower()`

Pré-processamento de texto: (2) Normalização

- 3. Remoção de acentos, caracteres especiais e pontuação
 - Remove ou substitui caracteres especiais ('&', '@'), acentos ('á', 'é') e pontuações do texto (.,;))
 - Exemplo: "São Paulo!" → "Sao Paulo"
 - Ferramentas: Bibliotecas como unidecode em Python para acentos, e regex para eliminar caracteres especiais
- 4. Expansão de contrações
 - Transformação de formas contraídas para suas formas completas
 - Exemplo: "can't" → "cannot"
 - Ferramentas: Dicionários específicos para expansão de contrações e substituições de texto

Pré-processamento de texto: (2) Normalização

- 5. Remoção de stop words
 - Palavras muito comuns em um idioma que geralmente não carregam muito significado semântico relevante
 - São normalmente consideradas stopwords:
 - Artigos, preposições e conjunções (não possuem semântica)
 - Alguns verbos de ligação, advérbios e adjetivos
 - Estima-se que 40% do texto sejam stopwords
 - Problemas: “Ser ou não ser, eis a questão”: Removendo-se as stopwords sobra apenas “questão”
 - Ferramentas: Bibliotecas como NLTK e SpaCy em Python oferecem listas de stop words padrão que podem ser personalizadas

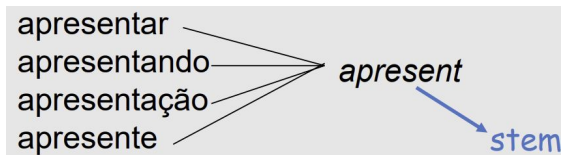
Pré-processamento de texto: (2) Normalização

- 6. Lematização

- Processo de converter palavras para sua forma base ou lema, levando em consideração o contexto gramatical
- Analisa a morfologia das palavras e transforma em sua forma canônica, como o infinitivo de verbos ou o singular de substantivos
- Serve para unificar as formas variantes de uma palavra
- Exemplo: "caminhando", "caminhei" → "caminhar"
- Ferramentas: NLTK e SpaCy em Python oferecem lematizadores

- 7. Stemming (pronúncia /'stemɪŋ/)

- Redução de palavras ao seu radical ou raiz, removendo sufixos e prefixos
- Serve para unificar as formas variantes de uma palavra
- O stem (ou radical) é a parte que sobra após a remoção do afixo
- Não precisa ser uma palavra válida
 - Mas precisa captar o significado dela
- Ferramentas: Algoritmos como o Porter Stemmer disponíveis no NLTK



Pré-processamento de texto: (2) Normalização

- 7. Stemming (pronúncia /'stɛmɪŋ/)
 - Tipos de stemmers:
 - Baseado em regras
 - Estatísticos

Pré-processamento de texto: (2) Normalização

- 7. Stemming (pronúncia /'stɛmɪŋ/)

- Tipos de stemmers:

- Baseado em regras: utilizam um conjunto de regras pré-definidas para cortar sufixos e prefixos das palavras
 - A maioria dos stemmers em uso são desta categoria
 - Exemplos (língua inglesa):
 - Lovins [1]
 - **Porter [2]**
 - Paice [3]
 - Simples e rápidos

– sses → ss	stresses → stress
– ies → i	ponies → poni
– s → ∅	cats → cat

- Estatísticos

[1] Lovins, J. B. "Development of Stemming Algorithms." *Mechanical Translation and Computational Linguistics*, 11(1-2), 22-31. 1968.

[2] Porter, M. F. "An Algorithm for Suffix Stripping." *Program*, 14(3), 130-137. 1980.

[3] Paice, C. D. "Another Stemmer." *ACM SIGIR Forum*, 24(3), 56-61. 1990

Pré-processamento de texto: (2) Normalização

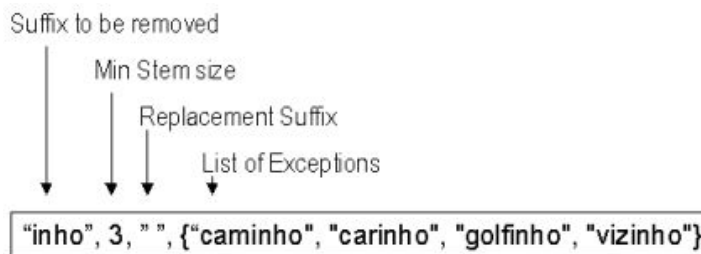
- 7. Stemming (pronúncia /'stɛmɪŋ/)
 - Tipos de stemmers:
 - Baseado em regras
 - Estatísticos: utilizam métodos baseados em dados para determinar o radical de uma palavra
 - O stemmer aprende a partir das estatísticas de uma coleção teste (precisa de um grande corpora)
 - Exemplos:
 - Contagem de frequência [1]
 - Baseado em clustering [2]
 - Flexíveis e adaptáveis

[1] Oard, D., Gonzalo, J., Sanderson, M., & Zajic, D. Statistical Stemming and Backoff Translation Strategies. CLEF. (2000)

[2] Majumder, P. et al. YASS: Yet another suffix stripper. ACM Trans. Inf. Syst., 25(4), 18. (2007)

Pré-processamento de texto: (2) Normalização

- 7. Stemming (pronúncia /'stɛmɪŋ/)
 - Tipos de stemmers:
 - Baseado em regras
 - Estatísticos
 - Estudos concluíram que **métodos estatísticos** produzem **resultados equivalentes ao Porter** [1]
 - RSLP[2]: stemmer para o Português
 - Baseado em conjunto de regras
 - Está implementado no NLTK
(https://www.nltk.org/_modules/nltk/stem/rslp.html)



[1] Bacchin et al.: Experiments to evaluate a statistical stemming algorithm. University of Padua at CLEF. (2002)

[2] V.M.Orengo & C. Huyck "A Stemming Algorithm for the Portuguese Language" SPIRE. (2001)

Pré-processamento de texto: (2) Normalização

- 7. Stemming (pronúncia /'stɛmɪŋ/)
 - Erros associados:
 - *Understemming*: deixar de remover sufixos
Ex: "bares" → "bare"
"adequado" → "adeq"
"adequação" → "adequaç"
 - *Overstemming*: remover caracteres que fazem parte do stem
Ex: "avião" → "avi"
"bebê" → "beb"
"bebendo" → "beb"

Pré-processamento de texto: (2) Normalização

- 7. Stemming (pronúncia /'stɛmɪŋ/)
 - Dificuldades:
 - Muitas exceções!
 - Homógrafos (mesma grafia, significados diferentes)
Ex: casais (“dois casais” ou “vós casais”)
 - Verbos:
 - Redução para o infinitivo, verbos irregulares
 - Mudanças no radical
Ex: emitir - emissão
 - Nomes próprios

Pré-processamento de texto: (3) Tokenização

- É o processo de dividir um texto em unidades menores chamadas tokens
- Dada uma sequência de caracteres e uma unidade de indexação, a tokenização separa esta sequência em tokens (termos, palavras etc.)
- Solução simples para separar palavras: espaços e pontuação
 - São “separadores”, logo podem ser descartados
- Ex: *“Dê sempre preferência às lâmpadas LED, pois elas consomem muito menos que as lâmpadas convencionais”*

Quantos tokens (palavras) há nessa sentença?

Pré-processamento de texto: (3) Tokenização

- É o processo de dividir um texto em unidades menores chamadas tokens
- Dada uma sequência de caracteres e uma unidade de indexação, a tokenização separa esta sequência em tokens (termos, palavras etc.)
- Solução simples para separar palavras: espaços e pontuação
 - São “separadores”, logo podem ser descartados
- Ex: *“Dê sempre preferência às lâmpadas LED, pois elas consomem muito menos que as lâmpadas convencionais”*

Quantos tokens (palavras) há nessa sentença?

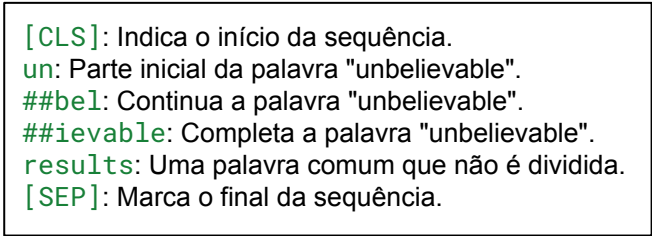
- 15 tokens
- 14 tokens distintos (lâmpadas aparecem 2x)

Pré-processamento de texto: (3) Tokenização

- Não é trivial decidir o que manter/descartar (Ex: números)
- Depende de cada linguagem
 - **Alemão:** substantivos compostos sem espaços
Ex: Kindergarten (jardim de infância)
Lebensversicherungsgesellschaftsangestellter
(funcionário da companhia de seguros de vida)
 - **Chinês, Japonês, Tailandês:** não utilizam espaço entre palavras
Ex: 私は寿司が大好きです (Watashi wa sushi ga suki desu.)
("Eu gosto de sushi.")
- Tokens “não usuais” são comuns
Ex: C++, T.V., abc@gmail.com

Pré-processamento de texto: (3) Tokenização

- Na prática precisa ser **eficiente** (rápida)
 - É um dos primeiros e mais fundamentais passos de PLN
 - É aplicada a grandes volumes de texto
 - Deve permitir processamento e análise em tempo real
- **Não necessariamente separa textos em “palavras”!**
 - O modelo BERT, por exemplo, tokeniza "unbelievable results" em:
`['[CLS]', 'un', '##bel', '##ievable', 'results', '[SEP]']`
- Bibliotecas conhecidas de Python para tokenização:
 - NLTK (<https://www.nltk.org/>)
 - spaCy (<https://spacy.io/api/tokenizer>)

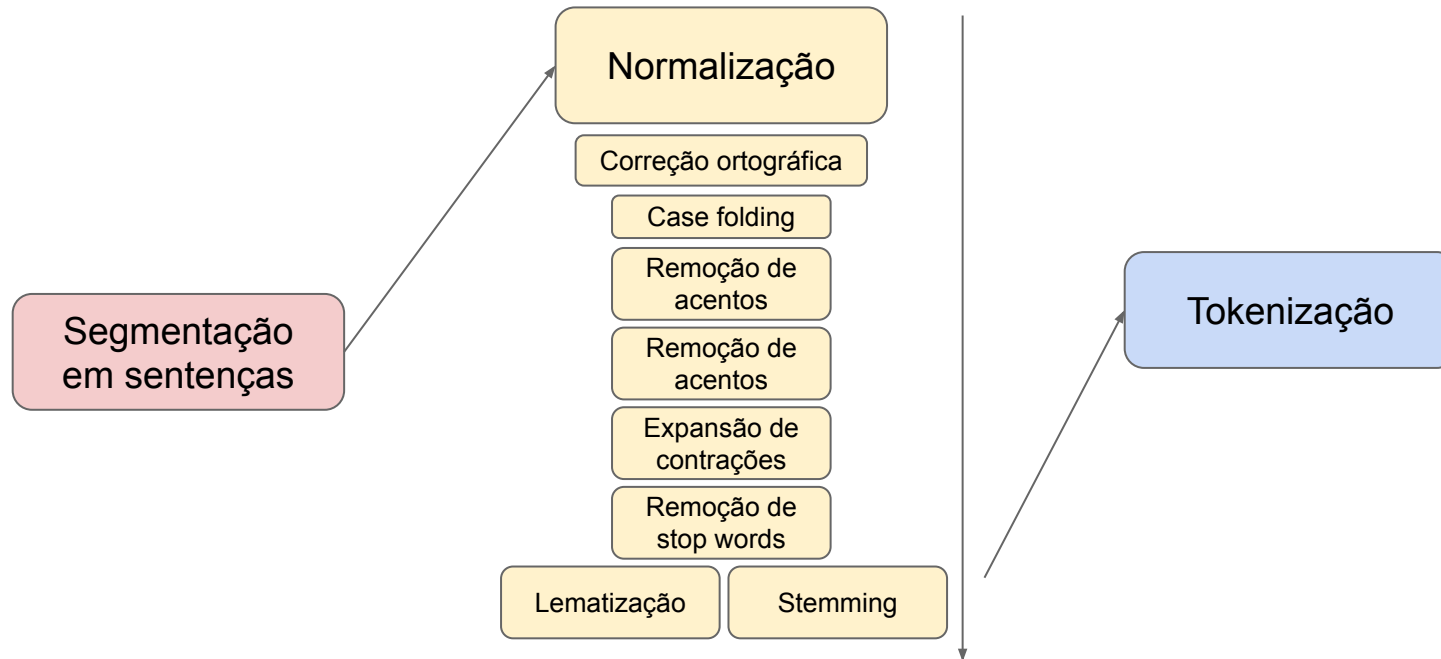


[CLS]: Indica o início da sequência.
un: Parte inicial da palavra "unbelievable".
##bel: Continua a palavra "unbelievable".
##ievable: Completa a palavra "unbelievable".
results: Uma palavra comum que não é dividida.
[SEP]: Marca o final da sequência.

Pré-processamento de texto

- Como resolver estes desafios de pré-processamento?
 - **Modelos de aprendizado profundo:** utilizar **redes neurais treinadas em grandes corpora** para aprender padrões contextuais que indicam o fim de uma sentença
 - **Modelos baseados em regras gramaticais:** desenvolver **regras linguísticas específicas** para identificar possíveis finais de sentença com base na estrutura gramatical e nas partículas da linguagem
 - **Análise de frequência e probabilidade:** utilizar **métodos estatísticos** para prever onde as sentenças terminam, com base em padrões comuns de uso da linguagem

Pré-processamento de texto



Distância mínima de edição

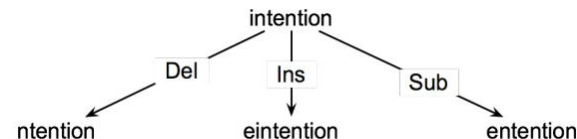
Distância mínima de edição

- Métrica utilizada para quantificar a diferença entre duas sequências de caracteres (geralmente palavras ou frases)
- Representa o número mínimo de edições de um caractere para transformar uma sequência na outra
- Aplicações:
 - Correção ortográfica: encontrar palavras próximas digitadas
 - Tradução automática: comparar tradução gerada pela máquina com a sua tradução “gabarito” (*ground truth*)
 - Análise de sequências de DNA: sequências são comparadas para encontrar mutações

AGGCTATCACCTGACCTCCAGGCCGATGCCC
TAGCTATCACGACCGCGGTCGATTGCCCCGAC

-AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
TAG-CTATCAC--GACCGC--GGTCGATTGCCCCGAC

Distância mínima de edição



- A **distância de Levenshtein** considera três operações:

- **Inserção:** custo 1
- **Deleção:** custo 1
- **Substituição:** custo 2 (considera “INS + DEL”)

- O algoritmo utiliza **programação dinâmica** para preencher uma matriz com os custos acumulados das edições necessárias

- Exemplos:

- $\text{dist}(\text{casa}, \text{caso}): 1$
 - $\text{sub}(a \rightarrow o)$ [caso]

- $\text{dist}(\text{patos}, \text{porta}): 4$
 - $\text{sub}(a \rightarrow o)$ [potos], $\text{ins}(r)$ [portos], $\text{sub}(o \rightarrow a)$ [portas], $\text{del}(s)$ [porta]

I N T E * N T I O N
 | | | | | | | | |
 * E X E C U T I O N
 d s s i s

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

Distância mínima de edição

- Observações:
 - Permite medir de forma “ótima” a similaridade entre duas sequências por meio de programação dinâmica
 - Os custos das operações podem ser customizados dependendo da necessidade:
 - Sistema de correção que leva em conta digitação pode considerar letras adjacentes menos custosas
 - Outras distâncias consideram outras operações
 - Damerau-Levenshtein permitem transposições (trocas de caracteres adjacentes)
 - A distância de Levenshtein possui complexidade de $O(m * n)$ em tempo e espaço, onde m e n são os comprimentos das duas strings

Material complementar

- Bibliotecas em python sobre expressões regulares:
 - <https://docs.python.org/3/library/re.html>
 - <https://pypi.org/project/regex/>
- Ferramentas online para testar regex:
 - <https://regex101.com/>
 - <https://regexr.com/>
- Pattern Matching with Regular Expressions in Python (automatetheboringstuff):
 - <https://automatetheboringstuff.com/2e/chapter7/>
- Tutoriais regex:
 - <https://realpython.com/regex-python/>
 - https://www.w3schools.com/python/python_regex.asp
 - <https://www.programiz.com/python-programming/regex>

Próximas aulas

- Aula teórica:
 - Introdução à representação de textos
 - Modelos tradicionais de representação:
 - Bag of words
 - TF-IDF
 - Representação de documentos
 - Medidas de similaridade entre textos
 - Distância euclidiana
 - Distância de cosseno
 - Problemas de representações tradicionais

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Instituto de Informática
Departamento de Informática Aplicada

Obrigado pela atenção!
Dúvidas?

Prof. Dennis Giovani Balreira
(Material adaptado da Profa. Viviane Moreira e do Prof. Dan Jurafsky)



INF01221 - Tópicos Especiais em Computação XXXVI:
Processamento de Linguagem Natural

