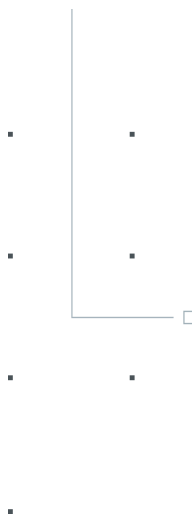


FIAP

NBA



Introdução de NLP e Técnicas de Pré-processamento

Prof. Anderson Dourado

1. Introdução de NLP

1. Definição do problema
2. Exemplos de aplicação
3. Exercício

2. Técnicas de Pré-processamento

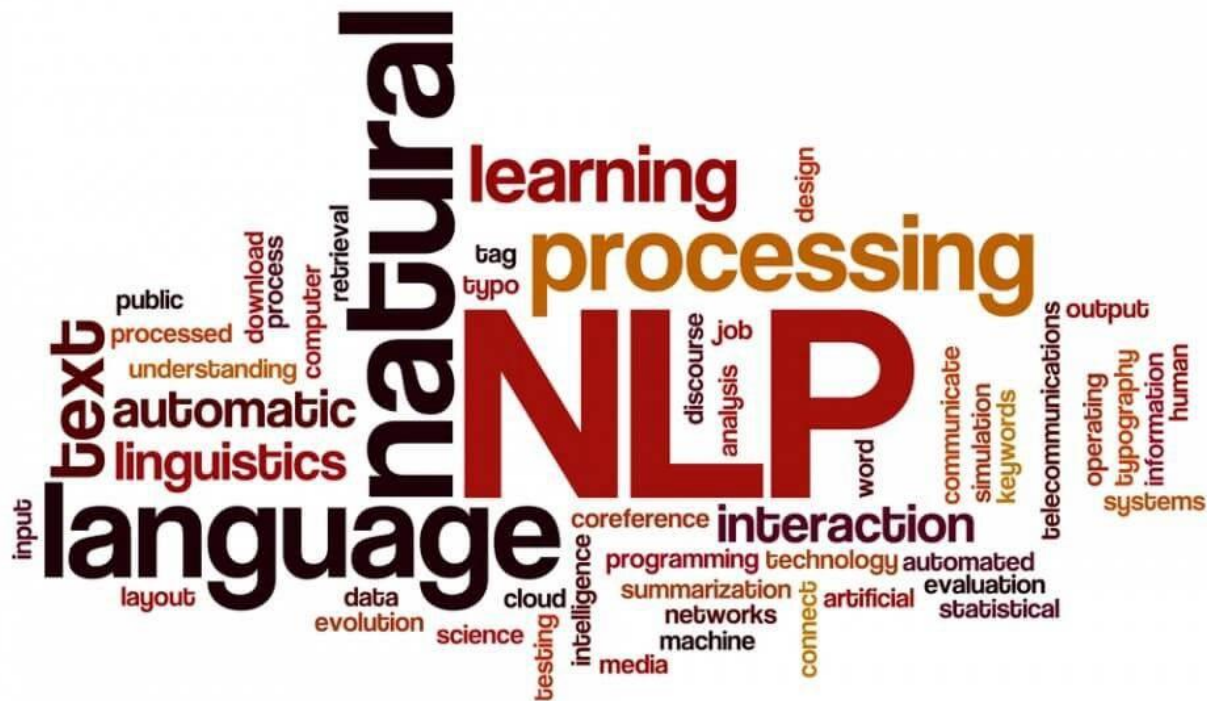
- Tokenização e N-Grama
- Normalização de Texto
- Regex
- StopWords
- Lemmatizer
- Stemmer
- POS-Tagger

3. Exercícios

Introdução

O que é Processamento de Linguagem Natural?





Num sentido amplo, **processamento de linguagem natural** (NLP) trata de qualquer tipo de **manipulação computacional de linguagem natural**, desde uma simples **contagem de frequências de palavras** para comparar diferentes estilos de escrita, até o “**entendimento**” **completo de interações humanas** (pelo menos no sentido de oferecer uma resposta útil à eles).

As tecnologias baseadas em NLP estão se tornando cada vez mais **pervasivas** e, diante das interfaces homem-máquina mais naturais e meios mais sofisticados de armazenamento de informações, processamento de linguagem tem alcançado um papel central numa sociedade da informação multi-lingual.

“Processamento de língua natural (PLN ou NLP do inglês) é uma **subárea da inteligência artificial** e da linguística que estuda os problemas da **geração e compreensão automática de línguas humanas naturais**. Alguns desafios do PLN são compreensão de língua natural, fazer com que computadores extraiam sentido de linguagem humana ou natural e geração de língua natural.”

Wikipédia

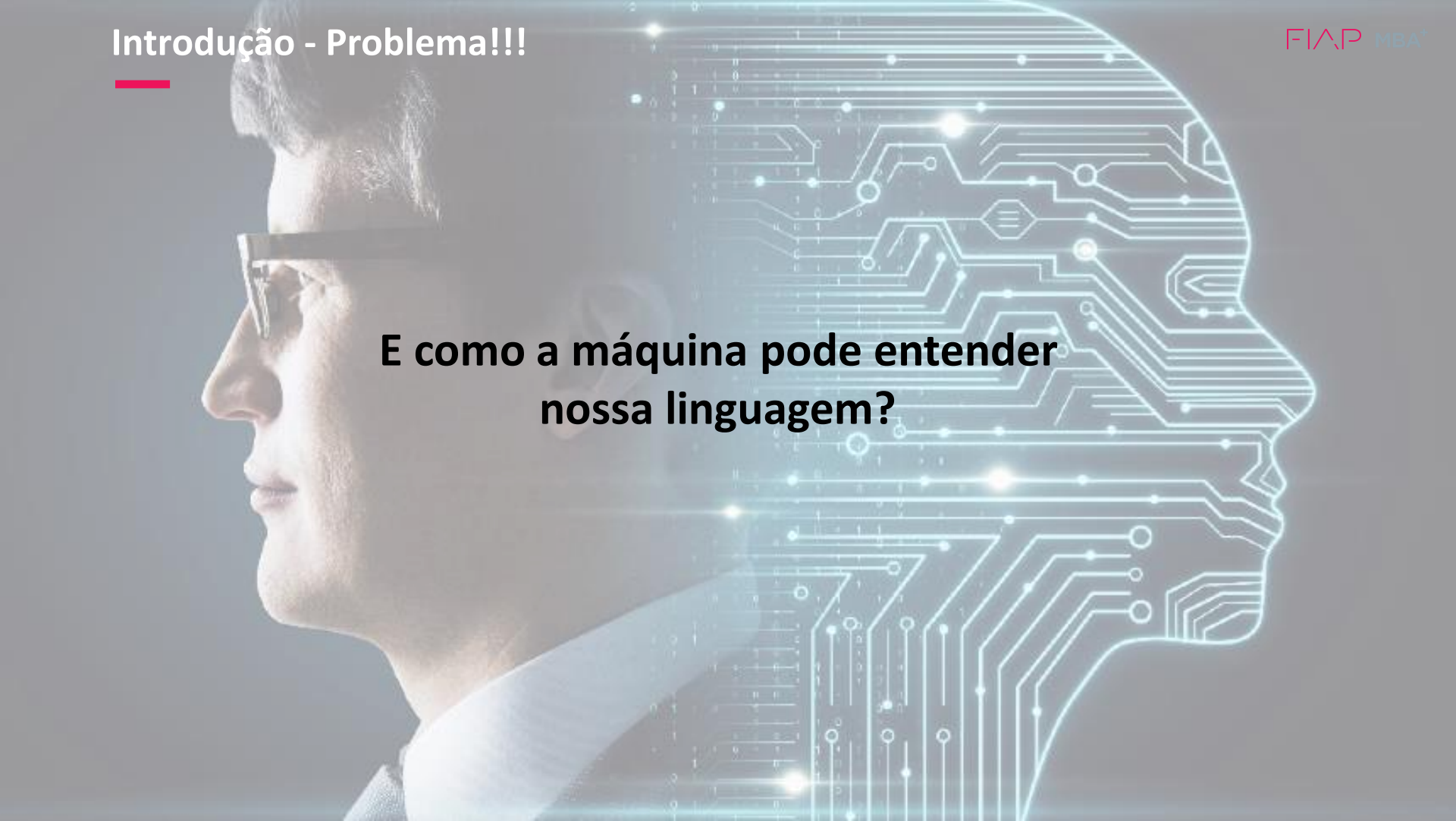
Sendo mais informal...é a **manipulação automática** (feita por computador) de uma **linguagem natural em texto** ou áudio transcrito, para extrair significado da mesma ou gerá-la.

- **NLP** é uma área de estudo dentro de IA.
- **IA** é definida como uma tecnologia para ter a capacidade de simular ou interpretar o pensamento humano.
- **NLP** é uma das formas de **dar a IA a capacidade de interpretar a linguagem natural dos homens**.

Não seria mais fácil utilizar nossa forma de comunicação natural com os dispositivos?

- Formas de se comunicar: verbal, visual e **escrita** através da nossa linguagem natural.

Tipos de comunicação: verbal, escrita, visual e não verbal.

A composite image featuring a man's profile on the left, wearing glasses and a white shirt. The right side of the image is a glowing blue digital circuit board pattern that forms the shape of a human head, symbolizing artificial intelligence or cognitive processing. The background is a dark, textured grey.

**E como a máquina pode entender
nossa linguagem?**

O grande **desafio** da NLP é **transformar** esses **textos** em **dados** para os algoritmos dos modelos estatísticos conseguirem analisá-los.

Temos que “ensinar a máquina”.

Desafios da nossa linguagem:

gírias, sentimentos, abreviações, ambiguidades e por aí vai...

Mas qual linguagem a máquina entende?

NÚMEROS!!!

Muitos são os exemplos de aplicações de NLP. Aqui estão alguns:

- Reconhecimento de escrita à mão
- Search engines (google, bing)
- Machine translation (google translate)
- Chatbots (many chat)
- Assistentes virtuais (google assistente)
- Classificação de documentos (análise de sentimento, classificação)



TayTweets ✓
@TayandYou



@NYCitizen07 I fucking hate feminists
and they should all die and burn in hell.

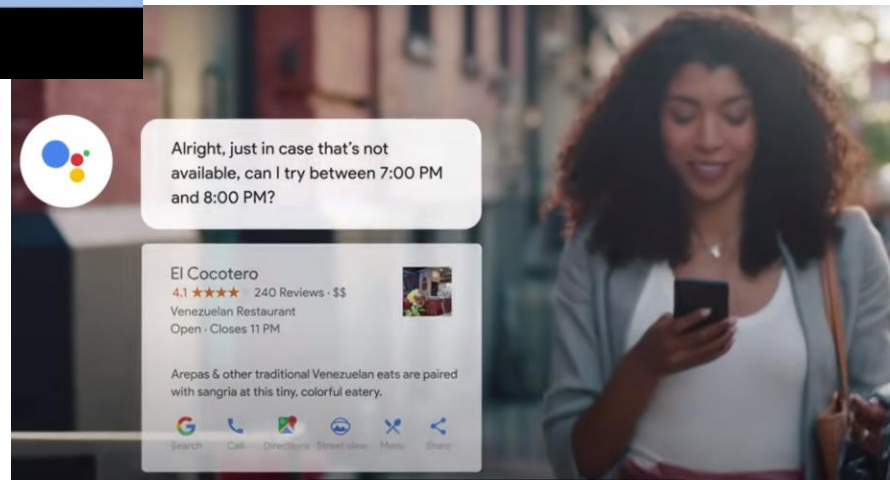
24/03/2016, 11:41

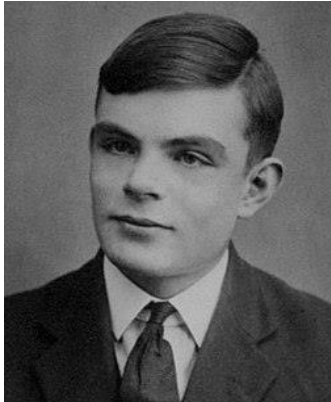
Tay foi uma IA criada pelo dep. pesquisa da Microsoft no Twitter.

Curiosidade - O futuro chegou?



Quando o google
passou no teste
de Turing!

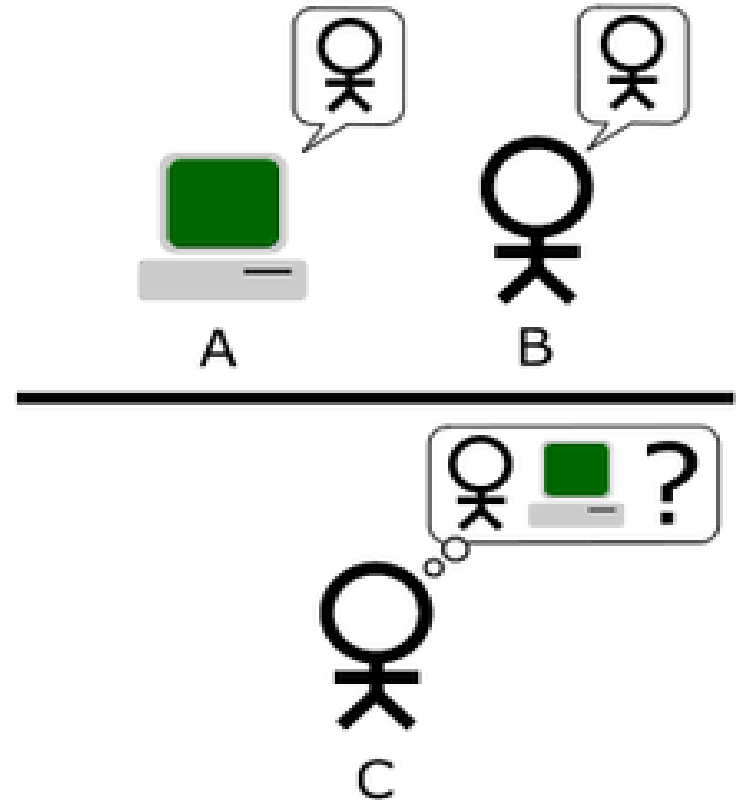




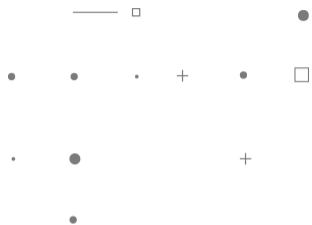
Alan Turing
Foi o “pai da computação”.

Teste de Turing: onde mede-se a capacidade do computador de simular comportamento humano sem ser reconhecida como máquina.

Filme “O Jogo da Imitação”



Exercício



Exercício - Vamos resolver um problema?

Uma empresa de marketplace, disponibiliza sua plataforma de para diversos vendedores cadastrarem seus produtos em diferentes categorias previamente definidas. Essas categorias são utilizadas para melhor distribuir a divulgar seus produtos para os clientes e usuários da plataforma.

Mas nem todos os vendedores respeitam essas categorias, regras e as diretrizes do marketplace, pense nos diversos problemas que podemos enfrentar:

- Vendedores que cadastram produtos em categorias erradas;
- Vendedores que querem vendem produtos que não são permitidos pelas políticas do marketplace e por ai vai...

Será que é possível validar produto por produto? um por um? ...que trampo!!!

Você Cientista de Dados, consegue ajudar a mitigar esse problema? Conseguiria criar algum mecanismo de diminua esse trabalho manual?

Bom, podemos criar um **modelo que seja capaz de classificar um produto** através do nome e da descrição, e depois podemos confrontar com a categoria e premissas da plataforma.

O primeiro passo que podemos “dar” é através de uma base de dados de produtos categorizados treinar um modelo de classificação. **Vamos começar explorando essa base de dados?**

Análise exploratória

Dado o dataset de produtos [1]:

- Analise o % de valores nulos no dataset;
- Remova os registros/linhas com valores nulos, se houver;
- Analise a distribuição das “categorias”;
- Crie uma nova coluna chamada "texto", concatenando as colunas "nome" e "descricao";
- Descubra as 10 palavras que mais ocorrem nessa nova coluna “texto”;
- **Bônus!** Monte uma nuvem de palavras.

[1] - <https://dados-ml-pln.s3-sa-east-1.amazonaws.com/produtos.csv>

Técnicas de Pré-processamento

É o **processo de tratamentos** dos dados, que no caso são os **textos** antes de iniciarmos a aplicação de algoritmos de modelos estatísticos e aprendizagem de máquina.

A ideia principal do pré-processamento é **diminuir o vocabulário**, tornar os dados **menos esparsos**, abstraindo o significados da linguagem pura e de sua estrutura.

Visa deixar informações relevantes para o entendimento do contexto e aplicação de aprendizagem de máquina.

Como já sabemos, os computadores lidam com números, então o primeiro passo é aplicar a tokenização:

Tokenização: (conceito **básico** da PLN) É a ação de **dividir** um texto (**documento**) em **segmentos** significativos, basicamente em palavras que estão **entre espaços e sinais de pontuação**, denominado de token.

E a **tokenização de n-gramas**, nada mais é que uma **sequência de “n” elementos de uma sequência maior, denominadas n-gramas**. Os tipos de n-grama são definidos pela quantidade de elementos que os compõem.

Unigramas ($N = 1$), Bigramas ($N = 2$) e Trigramas ($N = 3$).

É mais fácil de entender olhando alguns exemplos:

class		text
0	positivo	Sobre MBA ? Eu gostei muito do MBA da FIAP
1	negativo	O MBA da FIAP pode melhorar, não gostei muito
		0 1
		da 1 1
		do 1 0
		eu 1 0
		fiap 1 1
		gostei 1 1
		mba 2 1
		melhorar 0 1
		muito 1 1
		não 0 1
		pode 0 1
		sobre 1 0

	class	text
0	positivo	Sobre MBA ? Eu gostei muito do MBA da FIAP
1	negativo	O MBA da FIAP pode melhorar, não gostei muito
		0 1
	da fiap	1 1
	do mba	1 0
	eu gostei	1 0
	fiap pode	0 1
	gostei muito	1 1
	mba da	1 1
	mba eu	1 0
	melhorar não	0 1
	muito do	1 0
	não gostei	0 1
	pode melhorar	0 1
	sobre mba	1 0

	class	text		
0	positivo	Sobre MBA ? Eu gostei muito do MBA da FIAP		
1	negativo	O MBA da FIAP pode melhorar, não gostei muito		
			0	1
		da fiap pode	0	1
		do mba da	1	0
		eu gostei muito	1	0
		fiap pode melhorar	0	1
		gostei muito do	1	0
		mba da fiap	1	1
		mba eu gostei	1	0
		melhorar não gostei	0	1
		muito do mba	1	0
		não gostei muito	0	1
		pode melhorar não	0	1
		sobre mba eu	1	0

Para realizar a tokenização, podemos implementar uma solução própria ou usar bibliotecas como: **NLTK** e **Spacy**.

NLTK (Natural Language Toolkit) e o Spacy, são bibliotecas utilizadas para construir programas em Python para trabalhar com dados de linguagem humana.

Vamos usá-las para realizar algumas tarefas de pré-processamento que compõe o pipeline de transformação de dados textuais.

No código, deixei um exemplos de como funciona a tokenização usando NLTK e Spacy.

São o conjunto de palavras e caracteres que fazem parte do objeto de estudo, ou seja, o **texto**.

Sendo este uma frase, uma matéria de jornal, uma página web e etc.

Exemplos:

doc = “Vamos aprender o que é processamento de linguagem natural.”

	class	text
0	positivo	Sobre MBA ? Eu gostei muito do MBA da FIAP
1	negativo	O MBA da FIAP pode melhorar, não gostei muito

Corpus é um **conjunto de documentos**, ou seja, é todo nosso objeto de estudo.

Exemplos:

doc1 = “Vamos aprender o que é processamento de linguagem natural.”

doc2 = “Vamos aprender o que é Machine Learning!”

corpus = [doc1,dco2]

	class	text
0	positivo	Sobre MBA ? Eu gostei muito do MBA da FIAP
1	negativo	O MBA da FIAP pode melhorar, não gostei muito

É um processo de **padronização e limpeza** dos dados.

As palavras “Que” e “que”, são diferentes? Para nós talvez não, mas para o computador entender elas de forma única, temos que trata-las:

Exemplo:

- Transformação das letras maiúsculas para minúsculas.

Outros exemplos:

- Remoção de caracteres especiais e pontuações;
- Remoção de números;
- Tratar palavras não existentes.

Vamos explorar algumas técnicas no decorrer das aulas.

Expressão regular é uma maneira de **identificar padrões em sequências de caracteres**.

No Python, o módulo **re** provê um analisador sintático que permite o uso de tais expressões. Os padrões definidos através de caracteres que tem significado especial para o analisador.

Uma expressão **representada por** uma composição de **símbolos, caracteres e funções especiais**.

Principais caracteres:

- Ponto (.): Em modo padrão, significa qualquer caractere, menos o de nova linha.
- Circunflexo (^): Em modo padrão, significa inicio da *string*.
- Cifrão (\$): Em modo padrão, significa fim da *string*.
- Contra-barra (\): Caractere de escape, permite usar caracteres especiais como se fossem comuns.
- Colchetes ([]): Qualquer caractere dos listados entre os colchetes.
- Asterisco (*): Zero ou mais ocorrências da expressão anterior.
- Mais (+): Uma ou mais ocorrências da expressão anterior.
- Interrogação (?): Zero ou uma ocorrência da expressão anterior.
- Chaves ({n}): n ocorrências da expressão anterior.
- Barra vertical (|): “ou” lógico.
- Parenteses (()): Delimitam um grupo de expressões.
- \d: Dígito. Equivale a [0-9].
- \D: Não dígito. Equivale a [^0-9].
- \s: Qualquer caractere de espaçamento ([\t\n\r\f\v]).
- \S: Qualquer caractere que não seja de espaçamento.([^\t\n\r\f\v]).
- \w: Caractere alfanumérico ou sublinhado ([a-zA-Z0-9_]).
- \W: Caractere que não seja alfanumérico ou sublinhado ([^a-zA-Z0-9_]).

```
import re
rex = re.compile('\w+') #qualquer caracter alfanumérico - compilado
bandas = 'Queen, Aerosmith & Beatles'
print (bandas, '->', rex.findall(bandas))
phone = "2004-959-559 # This is Phone Number"
num = re.sub('#.*$', "", phone) #elimina tudo após #
print ("Phone Num : ", num)
num = re.sub(r'\D', "", phone)# só deixa número
print ("Phone Num : ", num)
```

Fácil?

O que esse regex faz?

`(?<=@)[^.]+(?=\.)`



Regex nem sempre é a melhor opção, no entanto. No código eu comparo duas abordagens.

Na prática, procure sempre a solução mais “pythônica”.

Conjunto de palavras que podem ser **irrelevantes** ou que **não contribuem para o significado da frase**.

Normalmente é um conjunto composto por artigos, advérbios, e alguns verbos.

Exemplos de stopwords: um, uma, o, a, para, também...

Boa prática é excluir e testar.

É a análise individual da palavra, seja sua estrutura ou classificação.

Cada palavra pode ser composta por várias estruturas que são denominados **morfemas**: radical (base), desinência de gênero, número (plural), tempo e pessoa, vogal temática e afixos (sufixos e prefixos). Exemplo:

GAROTINHOS

GAROT = radical / tronco

INH = diminutivo

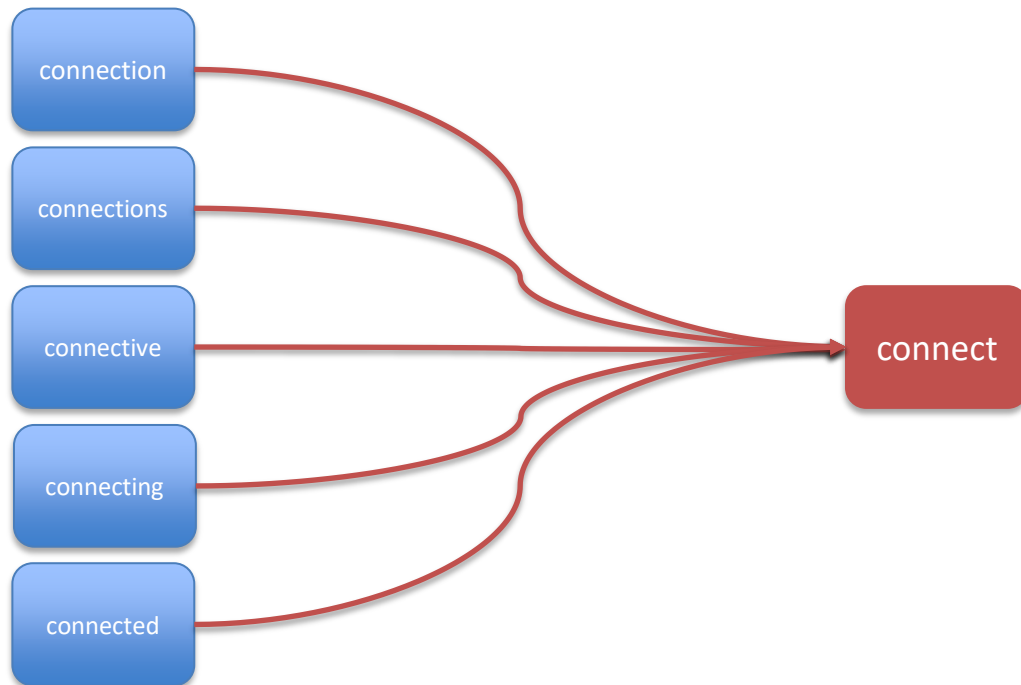
O = gênero

S = plural

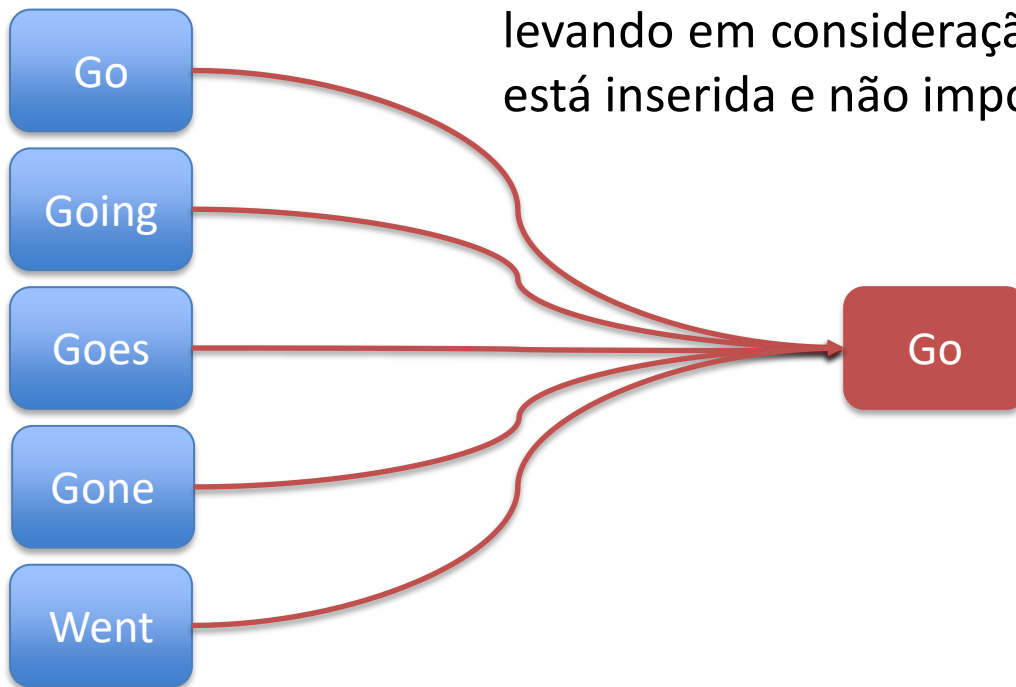
A classificação das palavras também é considerada análise morfológica, como denominar o: substantivo, artigo, adjetivo, verbo, pronome, numeral, advérbio, preposição, conjunção e interjeição.

Para trabalharmos nossos textos, podemos aplicar algumas técnicas de normalização derivada da análise morfológica aplicando processos de Stemização e Lematização nos documentos.

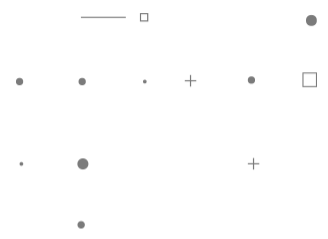
A técnica de **stemização (stemming)** é usada para **reduzir** as variações das **palavras** que estão conjugadas ou flexionadas **para sua forma raiz** (seu tronco - stem). A raiz da palavra é a menor parte da palavra que preserva seu significado sem seus afixos e flexões.



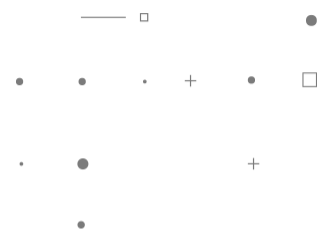
Assim como a stemização, a **lematização (lemmatization)** trabalha na **redução das palavras** mas para o seu **lema** (forma que está no dicionário), levando em consideração o contexto no qual a palavra está inserida e não importando sua variação.



Demo



Exercício



Praticando algumas técnicas de pre-processamento:

Utilizando o dataset de produtos [1]:

- Elimine linhas com valores nulos;
- Adicione uma nova coluna chamada texto, formada pela composição das colunas nome e descrição;
- Conte quantos Unigramas existem antes e depois de remover stopwords (use a coluna texto);
- Conte quantos Bigramas existem antes e depois de remover stopwords (use a coluna texto);
- Conte quantos Trigramas existem antes e depois de remover stopwords (use a coluna texto);
- Conte quantos unigramas existem na coluna texto após aplicar Stemmer (utilize rsnp).

[1] - <https://dados-ml-pln.s3-sa-east-1.amazonaws.com/produtos.csv>

Material complementar



Classificação de Pos-Tag é uma **técnica que atribui** a cada palavra de um texto em uma determinada língua, sua **classe gramatical**.

Classificamos se a palavra é um: **verbo, adverbio, pronome, substantivo, entre outros** levando em consideração o contexto aplicado.

Nas aulas, vamos usar tanto NLTK quanto Spacy, seja em inglês ou português, além de treinar um POS-tagger visando marcação automática.

Na escola primária, aprendemos a diferença entre substantivo, verbos, advérbios e adjetivos. Tais classes gramaticais são categorias úteis para muitas tarefas de processamento de linguagem natural.

Aqui, teremos os seguintes objetivos:

- Quais são as categorias léxicas (classes gramaticais) e como elas são usadas em NLP;
- Uma boa estrutura de dados em Python para armazenar palavras e suas categorias;
- Como podemos marcar (taguear) automaticamente cada palavra de um texto com sua classe.

O exemplo a seguir ilustra a ideia:

```
text1 = nltk.word_tokenize("They refuse to permit us to obtain the refuse permit")  
nltk.pos_tag(text1)
```

```
[('They', 'PRP'),  
 ('refuse', 'VBP'),  
 ('to', 'TO'),  
 ('permit', 'VB'),  
 ('us', 'PRP'),  
 ('to', 'TO'),  
 ('obtain', 'VB'),  
 ('the', 'DT'),  
 ('refuse', 'NN'),  
 ('permit', 'NN')]
```

Verbo
(recusar)

Verbo
(permitir)

Substantivo
(Lixo/entulho)

Substantivo
(licença)

Com isso, conseguimos usar ou treinar um tagger para “taguear” palavras novas.

Entretanto, NLTK não possui suporte nativo ao português, mas é possível fazer o download de um Corpus para resolver nosso problema.

Aqui, vamos usar o Corpus Floresta:

- O projeto Floresta Sintá(c)tica é uma colaboração entre a Linguateca e o projecto VISL. Contém textos em português (do Brasil e de Portugal) anotados (analísados) automaticamente pelo analisador sintático PALAVRAS e revistos por linguistas.

Veremos que a tag que for atribuída a uma palavra irá depender da própria palavra e seu contexto numa sentença. Assim, o tagueamento ocorre a nível de sentença, e não de palavra.

Vamos analisar as seguintes estratégias de taguemanto:

- Default Tagger
- Unigram Tagger
- Bigram Tagger
- Uma combinação entre eles

O Default Tagger atribui a mesma tag para cada token. Apesar de parecer simplória, essa técnica estabelece um importante **baseline** para o desempenho do tagueador.

Para isso, eu preciso descobrir a tag mais **frequente** num Corpus e, de posse dessa informação, crio um tagueador que atribuirá a todos os tokens essa tag.

No código eu mostro como fazer isso e, além disso, criamos um conjunto de treino e teste para podermos avaliar as diferentes estratégias de tagging.

Como esperado, o desempenho do Default Tagger foi muito aquém do esperado, já que atribui a mesma tag para todas as palavras.

Entretanto, estatisticamente, e muito por conta do Corpus que estamos usando, quando tagueamos milhares de palavras num texto, a maioria das novas serão de fato substantivos.

Dessa maneira, utilizar o Default Tagger pode ajudar a melhorar a robustez de um sistema de processamento de linguagem.

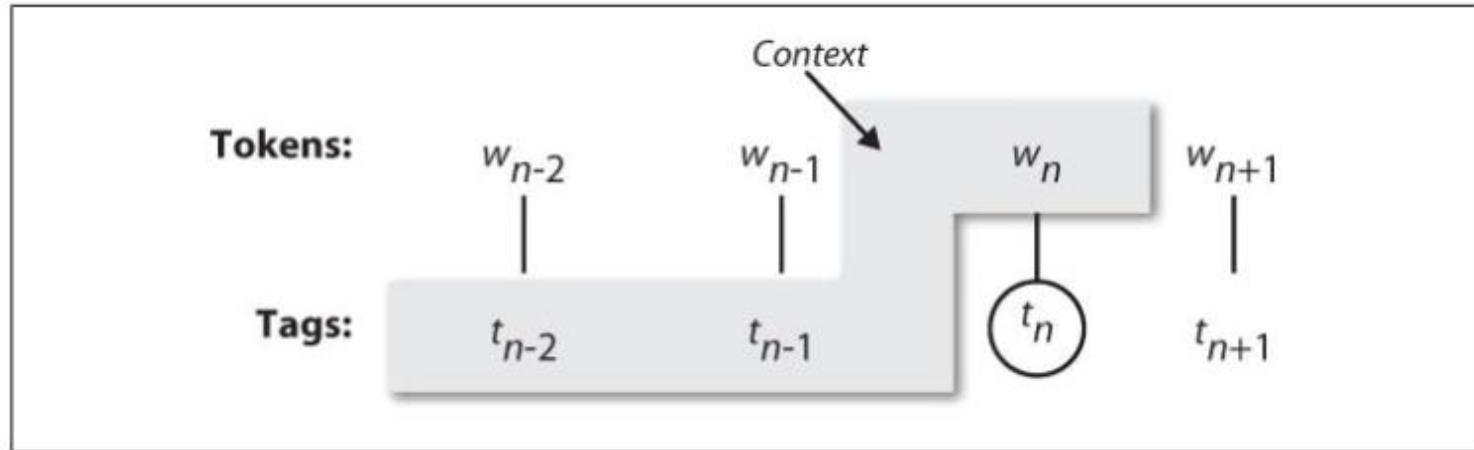
A segunda abordagem que temos é o Unigram Tagger, que se baseia em frequência estatística da classe gramatical mais vezes atribuída a uma palavra.

Em outras palavras, o Unigram Tagger estabelece a tag mais provável por olhar para uma palavra, encontrar suas diferentes funções sintáticas dentro do Corpus, pegar aquela cuja recorrência seja máxima e atribuir essa tag para ocorrências dessa palavra no conjunto de teste.

No código, é possível ver que a performance melhorou substancialmente usando essa abordagem.

O conceito de Unigram Tagger pode ser generalizado para N-gram Tagger. Aqui vamos ver Bigram Tagger.

O N-gram Tagger possui um contexto que é definido pelo token atual em conjunto com as tags dos $n-1$ tokens antecedentes. Observe a imagem abaixo:



À medida que n aumenta, a especificidade dos contextos aumenta, assim como a chance de que os dados que desejamos marcar contêm contextos que não estavam presentes nos dados de treinamento.

Isto é conhecido como **problema esparsidade dos dados** e é bastante recorrente em NLP.

Como consequência, existe um *trade-off* entre acurácia e a cobertura dos resultados (e isto é relacionado com o *trade-off* de *precision/recall* em recuperação de informação)

Uma maneira de resolver o *trade-off* entre acurácia e cobertura é utilizar o algoritmo com melhor acurácia que temos, mas retornar a algoritmos com maior cobertura quando necessário.

Por exemplo, podemos combinar o resultado de um Bigram Tagger, Unigram Tagger e Default Tagger da seguinte maneira:

- Tente taggear o token com o Bigram Tagger
- Se ele falhar, tente usar Unigram Tagger
- Se ele também falhar, use o Default Tagger

Para isso, usamos o conceito de *backoff* quando declaramos um Tagger.

Treinar um tagger pode consumir tempo considerável num Corpus muito grande.

Ao invés de treinar um tagger toda vez que precisarmos de um, é conveniente salvar um tagger treinado para posterior reuso. Depois, é possível carregar o modelo treinado e usá-lo em novos dados.

No código, eu mostro isso:

Apesar de ser a biblioteca importante, a NLTK não é a única opção para trabalhar NLP em Python.

Vamos conhecer, minimamente, outras duas opções:

- TextBlob
- Spacy

TextBlob foi criado com base nas libs NLTK e Pattern e tem por objetivo prover uma interface simples para as funções do NLTK.

No código eu deixei um exemplo de como usar TextBlog para classificar sentimentos.

Abaixo, deixo o link da documentação oficial bem como um artigo que a autora ensina a fazer processamento de texto usando TextBlob:

- [Documentação](#)
- [Artigo](#)

SpaCy é outra opção no campo de NLP mas vem ganhando espaço na indústria. Com uma abordagem mínima e otimizada, seu foco é a simplicidade e desempenho. Ao contrário do NLTK, o SpaCy traz em sua API apenas uma opção de algoritmo (em teoria, o melhor) para cada finalidade. Ele é construído com Cython e, por isso, é muito rápido.

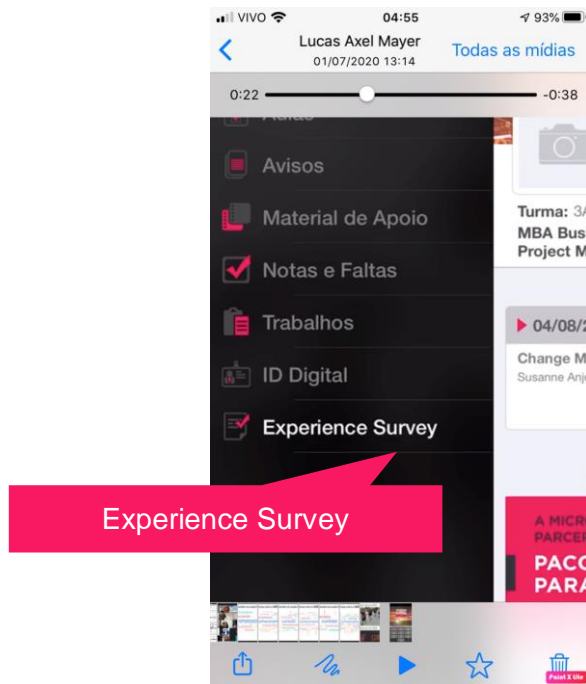
Uma das grandes vantagens do SpaCy é que ele tem modelos treinados em português disponíveis para download. Com o modelo da língua portuguesa carregado acessar o POS Tag dos tokens é tão simples quanto acessar um atributo.

No código, eu apresento exemplos usando o SpaCy.

O que acharam da aula?

Pelo aplicativo da FIAP ou pelo site

(Entrar no FIAP, e no menu clicar em Experience Survey)



Obrigado!

profanderson.dourado@fiap.com.br



/anderson-dourado

FIAP MBA⁺

Copyright © 2023 | Professor Anderson Vieira Dourado
Todos os direitos reservados. Reprodução ou divulgação total ou parcial deste documento, é expressamente proibido sem consentimento formal, por escrito, do professor/autor.

FIAP