DHENY R. FERNANDES

PÓS TECH

DATA ANALYTICS

DADOS GERADOS POR HUMANOS

AULA 02

SUMÁRIO

O QUE VEM POR AÍ?

Nesta aula, veremos o que é Processamento de Linguagem Natural, alguns exemplos de aplicações e quais as principais técnicas de pré-processamento de texto.

HANDS ON

Veremos quais técnicas usamos para trabalhar com texto. Durante a aula, mesclaremos conteúdo teórico e prático, através do Jupyter Notebook, a fim de entendermos e praticarmos sobre o conteúdo proposto.

SAIBA MAIS

Introdução

O campo de processamento de linguagem natural (NLP) passou por uma mudança dramática nos últimos anos, tanto em termos de metodologia quanto de aplicativos suportados. Os avanços metodológicos têm variado, desde novas formas de representar documentos a novas técnicas de síntese de linguagem. Com eles, surgiram novos aplicativos que vão desde sistemas de conversação abertos até técnicas que usam linguagem natural para a interpretabilidade do modelo. Por fim, esses avanços permitiram que a NLP ganhasse espaço em áreas relacionadas, como visão computacional e sistemas de recomendação. Esse último será nosso objeto de estudo futuramente.

Mas o que é NLP? 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 a eles).

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

Por conta disso, a NLP está rapidamente se tornando uma habilidade necessária, exigida por engenheiros, gerentes de produto, cientistas, estudantes e entusiastas que desejam construir aplicativos com base em dados de linguagem natural. Por um lado, novas ferramentas e bibliotecas para NLP e aprendizado de máquina tornaram a modelagem de linguagem natural mais acessível do que nunca. Mas, por outro lado, os recursos para aprender NLP devem visar esse público diversificado e sempre crescente.

Como dito, NLP permite interação com sistemas computacionais em linguagem humana. Entretanto, computadores entendem apenas dados binários, por exemplo, 0 e 1. Embora possamos representar dados de linguagem em binário, como fazemos as máquinas entenderem a linguagem? É isso que aprenderemos aqui.

Para exemplificar o quão importante NLP se tornou em nossa vida, aqui vão algumas aplicações:

1. Plataformas de e-mail usam NLP na classificação de mensagens (spam ou legítimas), priorização na caixa de entrada e auto-complete;
2. Assistentes baseados em voz, tais como Amazon Alexa, Apple Siri, Google Assistant ou Microsoft Cortana, utilizam técnicas de NLP para interagir com os usuários, entendendo-os e respondendo-os corretamente;
3. Plataformas de busca (Search engines), como Google ou Bing, usam NLP para entendimento de query (informação que o usuário digitou), recuperação da informação e ranqueamento, para citar alguns;
4. Tradução de máquina, como o Google Translate, é construído a partir de técnicas de NLP;
5. NLP também pode ser usado em uma variedade de campos, como jurídico, saúde, varejo, atendimento, marketing e outros.

Ao longo da fase, trabalharemos com alguns exemplos e veremos sua aplicação prática no mundo real.

Modelar problemas de NLP não é uma tarefa trivial. Pelo contrário, é bem desafiador. Veremos dois exemplos. O primeiro é usado em sistemas de buscas:

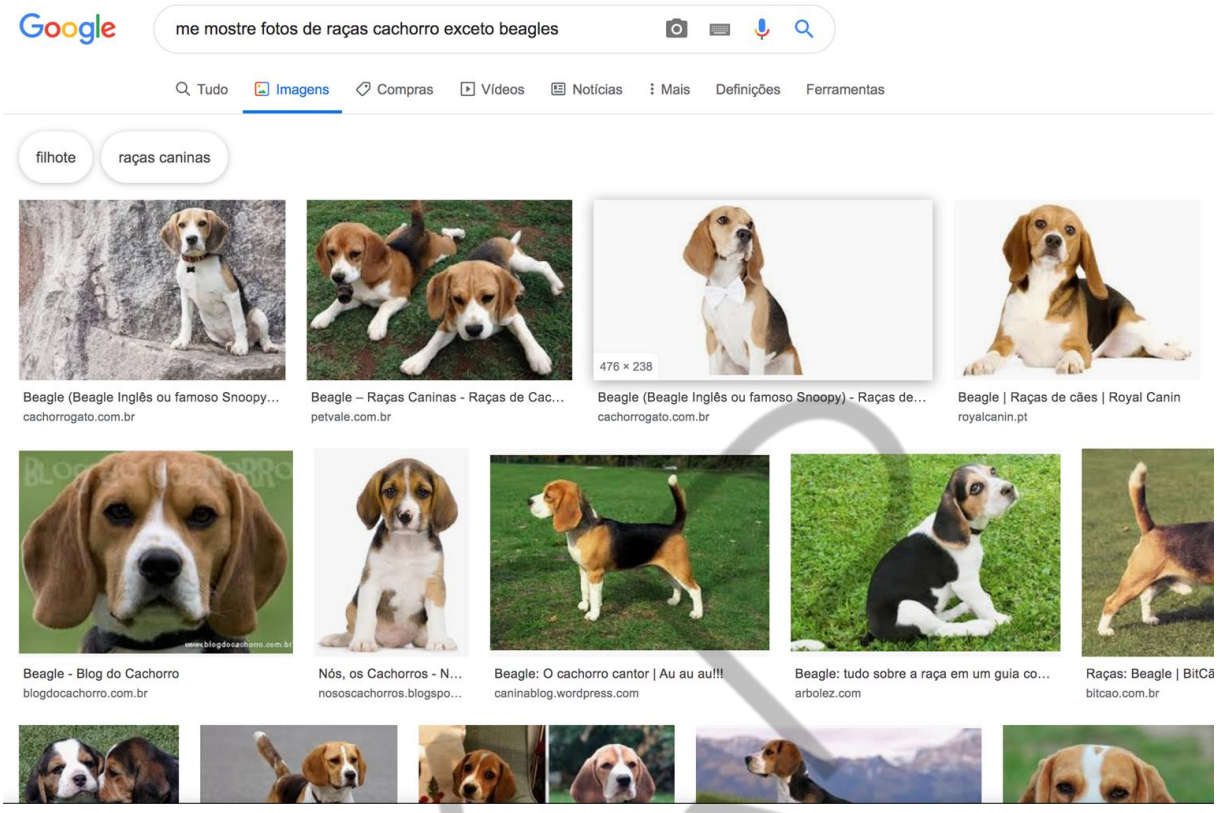


Figura 1 – Exemplo da complexidade do entendimento de linguagem
Fonte: Google (2023)

Esse caso deixa claro que interpretar linguagem humana não é uma tarefa trivial para as máquinas. Podemos imaginar como seria para o computador interpretar ironia, não é mesmo?

Esse outro caso é muito comum. Trata-se da ambiguidade:

Figura 2 – Exemplo de ambiguidade
Fonte: Elaborado pelo autor (2023)

Qual frase possui a interpretação correta? Como você justificaria sua escolha?

Perceba que a tarefa é mais complexa do que imaginamos. Criar um sistema de NLP, que consiga interpretar tais textos e fornecer uma resposta coerente, é realmente um trabalho árduo. Na sessão a seguir, discutiremos quais são as principais técnicas de pré-processamento usadas para preparar o texto, para que possam ser utilizados em sistemas de inteligência Artificial.

PIPELINE DE NLP

Quando falamos de NLP, geralmente usamos técnicas de Machine Learning. Elas são aplicadas a dados textuais da mesma maneira que são usadas em outros tipos de dados, como imagens ou dados estruturados.

Toda abordagem de Machine Learning para NLP, seja ela supervisionada ou não supervisionada, pode ser descrita em três passos comuns:

1. Extrair features de um texto.
2. Usar uma representação dessas features para aprender um modelo.
3. Avaliar e melhorar o modelo.

De maneira mais geral, precisamos de mais algumas etapas. O diagrama abaixo apresenta um pipeline genérico para NLP:

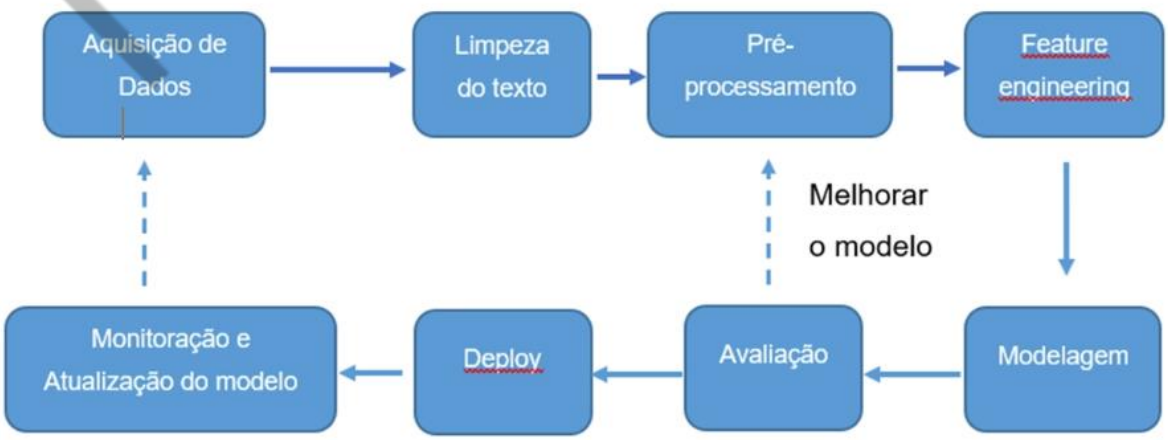


Figura 3 – Diagrama
Fonte: elaborado pelo autor (2023)

O primeiro passo é o de “aquisição de dados”, que consiste em obter os dados textuais que serão usados para treinar nosso modelo de Machine Learning. Geralmente, os dados estão disponíveis em bancos de dados nas próprias empresas. Às vezes, você encontra algumas bases públicas, em outros casos, você precisa ir capturando as informações aos poucos. Essa parte é importante, apesar de não ser o foco da disciplina, pois quanto mais dados, melhor será o nosso modelo.

No segundo passo, iremos fazer a limpeza desse texto, removendo qualquer coisa que não seja texto, tais como metadados, links, entre outros. O terceiro passo é o pré-processamento. Às vezes, ele é executado junto com a limpeza dos dados, consistindo numa série de técnicas que serão objeto de estudo dessa aula.

O próximo passo, também conhecido como “feature extraction”, consiste em extrair do texto quais características melhor o descrevem e imputá-las em algoritmos de machine learning. A parte de modelagem consiste em escolher uma técnica de machine learning e usar os dados tratados para resolver um problema específico. Logo em seguida, devemos avaliar esse modelo, a partir de alguns critérios que apresentaremos no decorrer da disciplina.

Por fim, faremos o deploy do nosso modelo, disponibilizando o modelo treinado em ambiente produtivo e, frequentemente, monitorando desvios, erros e realizando correções, quando necessário, treinando o modelo novamente.

Técnicas de Pré-processamento

Como dito anteriormente, o primeiro passo a ser feito em NLP é extrair features de um texto. Mais precisamente, precisamos adquirir os dados, limpá-los, fazer o pré-processamento e extrair as características, transformando os dados para que possam ser imputados em modelos de machine learning. Vamos tratar desses passos nessa sessão. Importante dizer que, a título de didática, consideramos que já temos os dados disponíveis.

Como sabemos, os computadores lidam com números. Qualquer outro tipo de dado (como áudio, texto, imagens, etc.) precisa passar por um processo de transformação para números.

Para realizar essa transformação, antes precisamos fazer o pré-processamento de dados, cujo objetivo é preparar os textos para criar um espaço de características adequado.

O primeiro passo é a tokenização dos dados. A tokenização nada mais é que uma sequência de “n” elementos de uma sequência maior, denominadas n-gramas. Seu objetivo é transformar os textos em números. Os tipos de n-grama são definidos pela quantidade de elementos que os compõem. Unigramas (N = 1), Bigramas (N = 2) e Trigramas (N = 3).

A ideia é dividir o texto, geralmente usando “espaço” como separador, em tokens, para realizar outras atividades.

Considere as seguintes frases como exemplos:

- Eu gosto de assistir jogos de futebol.
- Já eu, prefiro assistir jogos de basquete.

Quando falamos de unigrama, este é o tipo de representação que obtemos:

	0	1
assistir	1	1
basquete	0	1
de	2	1
eu	1	1
futebol	1	0
gosto	1	0
jogos	1	1
já	0	1
prefiro	0	1

Figura 4 – Representação vetorial - unigram
Fonte: Elaborado pelo autor (2023)

Quando falamos de bigramas e trigramas, estas são as representações que obtemos:

Bigramas

	0	1
assistir jogos	1	1
de assistir	1	0
de basquete	0	1
de futebol	1	0
eu gosto	1	0
eu prefiro	0	1
gosto de	1	0
jogos de	1	1
já eu	0	1
prefiro assistir	0	1

Trigramas

	0	1
assistir jogos de	1	1
de assistir jogos	1	0
eu gosto de	1	0
eu prefiro assistir	0	1
gosto de assistir	1	0
jogos de basquete	0	1
jogos de futebol	1	0
já eu prefiro	0	1
prefiro assistir jogos	0	1

Figura 5 – Representação vetorial – bigramas e trigramas
Fonte: Elaborado pelo autor (2023)

Para realizar a tokenização, podemos implementar uma solução própria ou usar a biblioteca NLTK. NLTK, ou Natural Language Toolkit, é uma plataforma para construir programas em Python para trabalhar com dados de linguagem humana. Vamos usá-la para realizar a maioria das tarefas de pré-processamento que compõe o pipeline de transformação de dados textuais.

Outra técnica que auxilia muito no pré-processamento é o Regex. “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 têm significado especial para o analisador.

A tabela abaixo resume os principais caracteres que são usados para definir um padrão a ser procurado numa string:

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_]).

Figura 6 – Padrões de Regex
Fonte: Natural Languagem Processing with Python (2009)

Essa solução é muito usada como um complemento para soluções mais complexas de NLP, principalmente quando é necessário capturar informações de padrões bem definidos e estabelecidos, como CPF, RG, e-mail, entre outros.

Quando lemos um texto, percebemos que algumas palavras sempre aparecem, mas elas não contribuem para uma interpretação mais sólida de uma frase. Tais palavras são chamadas "stop-words". Normalmente, são artigos, advérbios, preposições, conectivos e até alguns verbos.

Geralmente, quando nos deparamos com um texto, optamos por remover tais palavras, fazendo com que nosso modelo se concentre no que, de fato, é importante dentro de uma frase. Tecnicamente falando, estamos reduzindo o espaço de características.

Por falar nele, outra técnica que auxilia o modelo é a normalização, que consiste em fazer com que as palavras tenham a mesma representação morfológica, a fim de não gerar vieses. Por exemplo, caso eu tenha a palavra "Que" e, posteriormente, a

palavra “que”, elas não podem ser tratadas como distintas. Para resolver isso, colocamos todas as palavras em letra minúscula.

Outro fator de normalização é o tratamento que damos aos tempos verbais das palavras. Num sentido simples, “foi” e “será” têm origem na mesma palavra: “é”. Logo, em alguns casos, fazemos com que as palavras sejam reescritas em sua forma raiz. Esse processo é conhecido como lematização. Uma técnica mais simples, mas similar, é a stemização, que consiste em podar (ou, pelo menos, tentar podar) a palavra até um radical mais próximo.

Para entendermos o funcionamento dessas três técnicas, vamos analisar as seguintes frases:

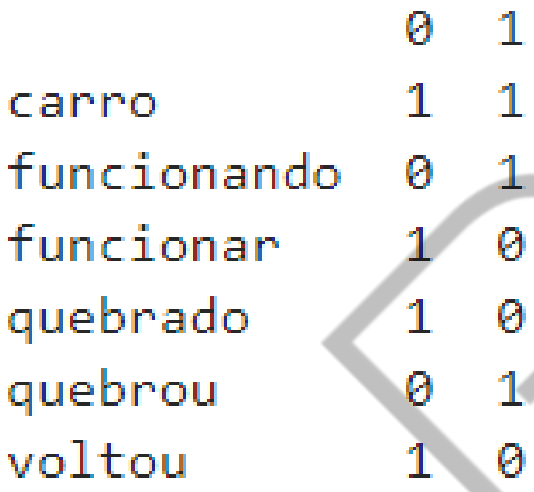
- O carro que estava quebrado voltou a funcionar.
- Meu carro quebrou e não está funcionando.

O primeiro passo é realizar a vetorização dos dados, ou seja, transformar palavras em números. Nesse caso, é colocado “0” para quando a palavra não está na frase e “1” para quando está. O conjunto de palavras distintas forma o que chamamos de dicionário. O resultado obtido é o seguinte:

	0	1
carro	1	1
estava	1	0
está	0	1
funcionando	0	1
funcionar	1	0
meu	0	1
não	0	1
que	1	0
quebrado	1	0
quebrou	0	1
voltou	1	0

Figura 7 – vetorização
Fonte: elaborado pelo autor (2023)

O próximo passo é remover as stop-words, obtendo o seguinte resultado:



	0	1
carro	1	1
funcionando	0	1
funcionar	1	0
quebrado	1	0
quebrou	0	1
voltou	1	0

Figura 8 – remoção de stop-words
Fonte: elaborado pelo autor (2023)

Entretanto, ainda temos variações de uma mesma palavra dentro do meu conjunto de características:

- Funcionar: funcionar, funcionando.
- Quebrar: quebrado, quebrou.

Podemos melhorar a representação de texto usando stemming e lemmatization, para transformar palavras/tokens num formato padrão. O que as diferencia é a maneira com que elas atingem esse objetivo. A decisão de qual usar depende do trabalho e, muitas vezes, isso é decidido baseado em testes. No caso, eu usei stemming, obtendo o seguinte resultado:

	0	1
carr	1	1
est	1	1
funcion	1	1
quebr	1	1
volt	1	0

Figura 9 – Aplicação de Stemming
Fonte: elaborado pelo Autor (2023)

Ou seja, depois de aplicar todo esse processo de normalização, reduzi meu espaço de característica, o que contribui para que meu modelo se concentre apenas no essencial e tenha maiores chances de produzir um ótimo resultado.

POS-Tagging

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. POS-Tagging é uma ferramenta usada no processamento de linguagem natural (NLP), que permite que os algoritmos entendam a estrutura gramatical de uma frase e desambiguam palavras que têm vários significados.

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.

Observe o exemplo a seguir:

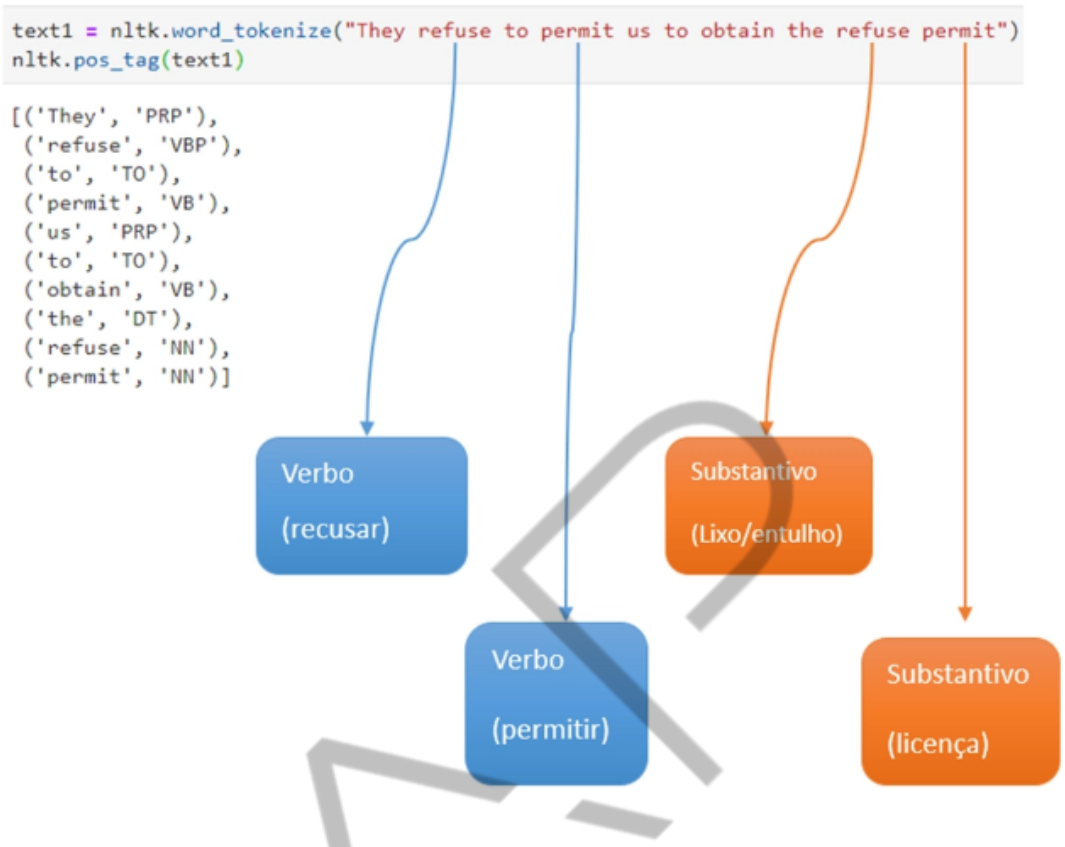


Figura 10 – Identificação automática de classe gramatical
Fonte: elaborado pelo autor (2023)

Mas, afinal, qual a real utilidade disso? Considere as seguintes palavras: Woman (substantivo), bought (verbo), over (preposição) e The (determinante).

Com o POS-tagging, é possível encontrar palavras que pertençam à mesma classe gramatical.

```
nltk.download('brown')
text = nltk.Text(word.lower() for word in nltk.corpus.brown.words())
text.similar('woman')
```

```
[nltk_data] Downloading package brown to
[nltk_data]     C:\Users\Dheny\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\brown.zip.
man time day year car moment world house family child country boy
state job place way war girl work word
```

```
text.similar('bought')
```

```
made said done put had seen found given left heard was been brought
set got that took in told felt
```

```
text.similar('over')
```

```
in on to of and for with from at by that into as up out down through
is all about
```

```
text.similar('the')
```

```
a his this their its her an that our any all one these my in your no
some other and
```

Figura 11 – Aplicações de POS-tagging
Fonte: elaborado pelo autor (2023)

Com isso, conseguimos 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 (analisados) automaticamente pelo analisador sintático PALAVRAS e revistos por linguistas.

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

Vamos analisar as seguintes estratégias de tagueamento:

- Default Tagger.
- Unigram Tagger.
- Bigram Tagger.

- Uma combinação entre eles.

O Default Tagger atribui a mesma tag para cada token. Apesar de parecer simples, essa técnica estabelece um importante baseline para o desempenho do tagueador. Para isso, é preciso descobrir a tag mais frequente num Corpus e, com essa informação, criar um tagueador que atribuirá a todos os tokens essa tag.

```
tagger0 = nltk.DefaultTagger('n')
print(tagger0.evaluate(test))
```

```
0.17800040072129833
```

Figura 12 – Teste de um Default Tagger
Fonte: elaborado pelo autor (2023)

O desempenho do Default Tagger é muito aquém do esperado, já que atribui a mesma tag para todas as palavras. Entretanto, estatisticamente e 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.

```
tagger1 = nltk.UnigramTagger(train)
print(tagger1.evaluate(test))
```

```
0.8522139851733119
```

Figura 13 – Teste de um Unigram Tagger
Fonte: elaborado pelo autor (2023)

O conceito de Unigram Tagger pode ser generalizado para N-gram Tagger. Aqui, veremos o 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 a seguir:

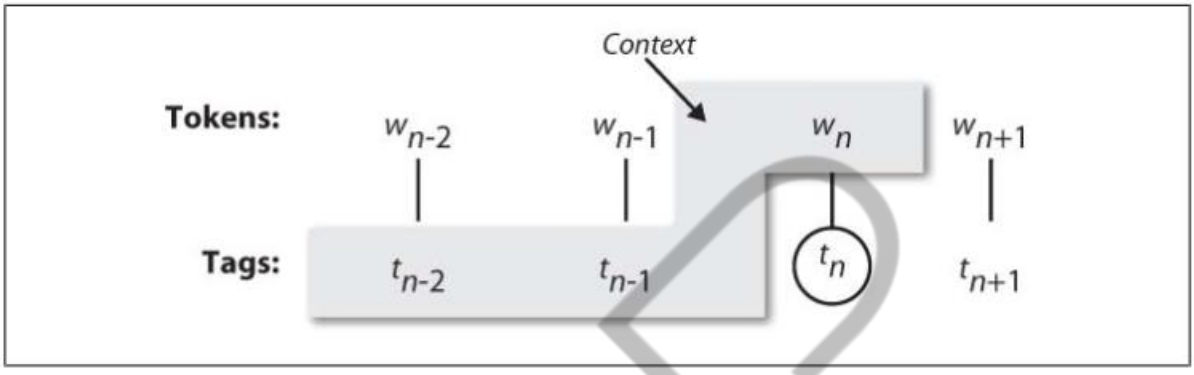


Figura 14 – Conceito do N-gram Tagger
Fonte: Natural Language Processsing With Python (2009)

Após treinar um bigram, entretanto, os resultados são bem aquém do esperado. Veja:

```
tagger2 = nltk.BigramTagger(train)
print(tagger2.accuracy(test))
```

```
0.14626327389300742
```

Figura 15 – Teste de um Bigram Tagger
Fonte: Elaborado pelo autor (2023)

Por qual motivo isso ocorreu? Veja a seguinte ilustração:

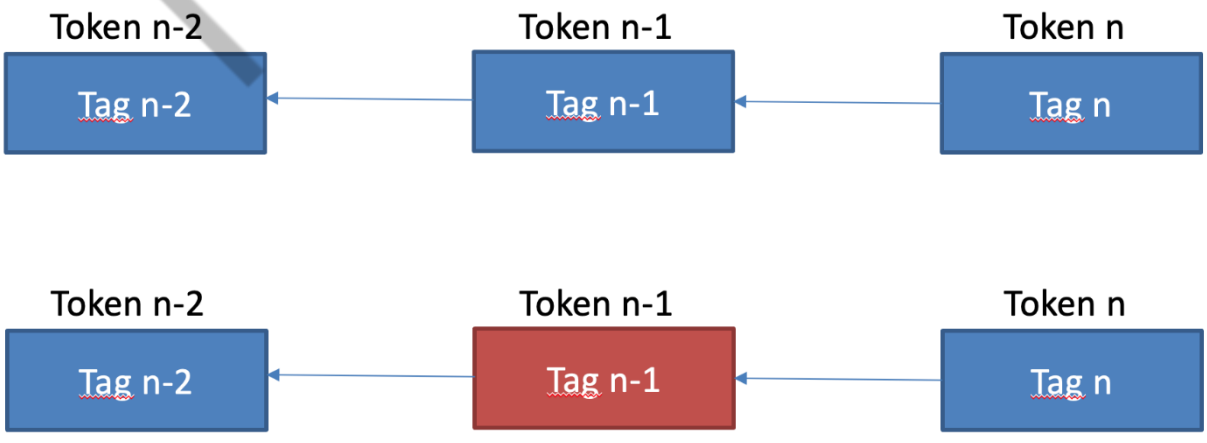


Figura 16 – Problemas no Bigram Tagger

Fonte: elaborado pelo autor (2023)

Mesmo que a “tag n” seja a mesma no conjunto de treino e teste, ela não será rotulada corretamente, já que a tag n-1 é diferente. Consequentemente, o tagger falha em taguear o resto da sentença.

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

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

Como consequência, existe um trade-off entre acurácia e a cobertura dos resultados (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 taguear 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.

```
tagger1 = nltk.UnigramTagger(train, backoff=tagger0)
print('tagger1: ',tagger1.accuracy(test))
tagger2 = nltk.BigramTagger(train, backoff=tagger1)
print('tagger2: ',tagger2.accuracy(test))
```

```
tagger1:  0.8740532959326788
tagger2:  0.8900420757363254
```

Figura 17 – Teste de um Tagger usando backoff
Fonte: Elaborado pelo autor (2023)

Por fim, treinar um tagger pode consumir um 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. Assim, é possível carregar o modelo treinado e usá-lo em novos dados.

O QUE VOCÊ VIU NESTA AULA?

Nesta aula, apresentamos o NLP, os desafios que temos, quais aplicações usam essa técnica e como machine learning pode nos auxiliar. Depois, entramos em detalhes para entender como realizar o tratamento de dados textuais, preparando-os para, futuramente, serem introduzidos no nosso modelo.

O que achou do conteúdo? Conte-nos no Discord! Estamos disponíveis na comunidade para fazer networking, tirar dúvidas, enviar avisos e muito mais. Participe!

REFERÊNCIAS

BIRD, S; KLEIN, E; LOPER, E. **Natural Language Processing with Python.** Sebastopol: O'Reilly Media, 2009.

VAJJALA, S et al. **Practical Natural Languagem Processing: A Comprehensive Guide to Building Real-World Nlp Systems.** Sebastopol: O'Reilly Media, 2020.

PALAVRAS-CHAVE

Natural Language Processing. Pré-processamento. NLTK.