Estratégia Concursos

Aula 03

Banco do Brasil (Escriturário - Agente de Tecnologia) Passo Estratégico de Tecnologia de Informação - 2023 (Pós-Edital)

Autor:
Thiago Rodrigues Cavalcanti

24 de Janeiro de 2023

NOÇÕES DE PROCESSAMENTO DE LINGUAGEM NATURAL

Sumário

ANÁLISE ESTATÍSTICA

Inicialmente, convém destacar os percentuais de incidência de todos os assuntos previstos no nosso curso – quanto maior o percentual de cobrança de um dado assunto, maior sua importância:

Assunto	Quantidade	Grau de incidência em concursos similares CESGRANRIO
Modelagem conceitual de dados (a abordagem entidade-relacionamento); Modelo relacional de dados (conceitos básicos, normalização);	55	23,40%
Linguagem SQL2008;	49	20,85%
5. Estrutura de dados e algoritmos: Busca sequencial e busca binária sobre arrays; Ordenação (métodos da bolha, ordenação por seleção, ordenação por inserção), lista encadeada, pilha, fila e noções sobre árvore binária.	42	17,87%
6. Ferramentas e Linguagens de Programação para manipulação de dados: Ansible; Java (SE 11 e EE 8); TypeScript 4.0;	34	14,47%
Data Warehouse (modelagem conceitual para data warehouses, dados multidimensionais);	31	13,19%
Python 3.9.X aplicada para IA/ML e Analytics (bibliotecas Pandas, NumPy, SciPy, Matplotlib e Scikit-learn).	11	4,68%
2. Banco de Dados: Banco de dados NoSQL (conceitos básicos, bancos orientados a grafos, colunas, chave/valor e documentos);	4	1,70%
4. Desenvolvimento Mobile: linguagens/frameworks: Java/Kotlin e Swift. React Native 0.59; Sistemas Android api 30 e iOS xCode 10.	4	1,70%
3. Big data: Fundamentos; Técnicas de preparação e apresentação de dados.	3	1,28%
1. Aprendizagem de máquina: Fundamentos básicos; Noções de algoritmos de aprendizado supervisionados e não supervisionados	1	0,43%
Postgre-SQL;	1	0,43%
Noções de processamento de linguagem natural.	0	0,00%
Conceitos de banco de dados e sistemas gerenciadores de bancos de dados (SGBD); MongoDB;	0	0,00%

ROTEIRO DE REVISÃO E PONTOS DO ASSUNTO QUE MERECEM DESTAQUE

A ideia desta seção é apresentar um roteiro para que você realize uma revisão completa do assunto e, ao mesmo tempo, destacar aspectos do conteúdo que merecem atenção.

Para revisar e ficar bem-preparado no assunto, você precisa, basicamente, seguir os seguintes tópicos:

Dicionário

Faremos uma lista de termos que são relevantes ao entendimento do assunto desta aula! Se durante sua leitura texto, você tenha alguma dúvida sobre conceitos básicos, esta parte da aula pode ajudar a esclarecer.

Conceitos de PLN

Análise sintática ou parsing é a tarefa de analisar strings como símbolos e garantir sua conformidade com um conjunto estabelecido de regras gramaticais.

Análise semântica está interessada em determinar o significado das seleções de texto (sequências de caracteres ou palavras). Depois que uma seleção de entrada de texto é lida e analisada (analisada sintaticamente), a seleção de texto pode então ser interpretada quanto ao significado.

Análise de sentimento é o processo de avaliação e determinação do sentimento capturado em uma seleção de texto, com sentimento definido como sentimento ou emoção. Esse sentimento pode ser simplesmente positivo (feliz), negativo (triste ou zangado) ou neutro, ou pode ser alguma medida mais precisa ao longo de uma escala, com neutro no meio e positivo e negativo aumentando em qualquer direção.

Corpus é definido como uma coleção de documentos de texto, por exemplo, um conjunto de dados contendo notícias é um corpus ou os tweets contendo dados do Twitter são um corpus.

Expressões regulares, geralmente abreviadas *regexp* ou *regexp* , são um método testado e comprovado de descrever de forma concisa os padrões de texto. Uma expressão regular é representada como uma string de texto especial e se destina ao desenvolvimento de padrões de pesquisa em seleções de texto.

Processamento de linguagem natural - a aplicação de algoritmos de aprendizado de máquina para a análise, compreensão e manipulação de exemplos escritos ou falados da linguagem humana.

Lematização é um processo sistemático passo a passo para remover as formas de inflexão de uma palavra. Ele faz uso de vocabulário, estrutura de palavras, tags de classes gramaticais e relações gramaticais. A saída da lematização é uma palavra raiz chamada **lema**. Por exemplo:

Executando, Executado, Execução >> Executar

Marcas de parte da fala ou marcas de PoS (part of speech) são as propriedades das palavras que definem seu contexto principal, sua função e o uso em uma frase. Algumas das tags de classes gramaticais comumente usadas são: **substantivos**, que definem qualquer objeto ou entidade; **verbos**, que definem alguma ação; e **adjetivos ou advérbios**, que atuam como modificadores, quantificadores ou intensificadores em qualquer frase.

n-gramas é um modelo de representação para simplificar a seleção de no texto. Ao contrário da representação sem ordem do saco de palavras (ver definição abaixo), a modelagem de n-gramas está interessada em preservar sequências contíguas de N itens da seleção de texto.

Saco de palavras (Bag of Words) é um modelo de representação particular usado para simplificar o conteúdo de uma seleção de texto. O modelo do saco de palavras omite a gramática e a ordem das palavras, mas está interessado no número de ocorrências de palavras no texto.

Stemming é um processo elementar baseado em regras para remover formas flexionais de um token e as saídas são o radical da palavra.

Stop Words são aquelas palavras que são filtradas antes do processamento posterior do texto, uma vez que essas palavras contribuem pouco para o significado geral, visto que geralmente são as palavras mais comuns em um idioma.

Tokenização - é um processo de divisão de um objeto de texto em unidades menores, também chamadas de tokens. Exemplos de tokens podem ser palavras, números, engramas ou mesmo símbolos.

Introdução

Existem várias formas didáticas de estudar o assuntos de processamento de linguagem natural, no passo estratégico resolvi segmentar o assunto em tópicos que refletem momentos distintos de manipulação dos dados em um processo de PLN. Vejamos a alista de tópicos que veremos na figura abaixo:

Representação de Texto

One-Hot encoding

Vamos começar examinando a representação de texto, ou essencialmente, como podemos carregar e processar dados de texto de uma maneira que os torne passíveis de serem processados e analisados facilmente usando uma linguagem de programação como Python ou qualquer outro método computacional. A primeira etapa é examinar a codificação one-hot, que é uma das maneiras fundamentais de representamos tokens textuais de uma forma que pode ser analisada computacionalmente.

Antes que possamos fazer qualquer coisa, o primeiro passo é conhecido como **tokenização**. E esta é apenas uma maneira elegante de dizer que queremos **dividir nosso texto**, seja um livro ou um site inteiro, ou até mesmo um twitte, é possível separar os textos em pequenos pedaços que sejam significativos de alguma forma. Normalmente, estes correspondem a palavras ou abreviaturas ou assim por diante. E queremos fazer isso de uma forma que seja **fácil e intuitivo de entender, que faça sentido e se adapte à nossa aplicação específica**.

Os tokens individuais **normalmente correspondem a palavras, números, pontuação, às vezes URLs, hashtags, nomes de usuário e assim por diante**. E, claro, há tokens que decidirmos manter e outros que você decidir descartar, geralmente chamamos isso de **remoção de stop words**. Outro ponto importante é que o número de tokens que escolhemos manter é conhecido também como **tamanho do vocabulário**, o número de palavras únicas ou número de tokens únicos também tem um custo computacional. Quanto mais tivermos, mais processamento teremos que fazer. E apenas para simplificar, é comum usar palavra e token neste contexto mais ou menos de forma intercambiável. Para dar um exemplo simples, aqui está uma frase.

"The Earth is estimated to be 4.54 billion years old, plus or minus about 50 million years."

(Traduzindo: "Estima-se que a Terra tenha 4,54 bilhões de anos, mais ou menos cerca de 50 milhões de anos.")

Há muitas maneiras diferentes de tokenizar esta frase, então vou mostrar apenas quatro e destacar as diferenças e os pontos em que elas mudam.

- ["The", "Earth", "is", "estimated", "to", "be", "4.54", "billion", "years", "old", ",", "plus", "or", "minus", "about", "50", "million", "years", "."]
- ["The", "Earth", "is", "estimated", "to", "be", "4.54", "billion", "years", "old", "plus", "or", "minus", "about", "50", "million", "years"]
- ["The", "Earth", "is", "estimated", "to", "be", "4.54", "billion", "years", "old,", "plus", "or", "minus", "about", "50", "million", "years."]
- ["The", "Earth", "is", "estimated", "to", "be", "billion", "years", "old,", "plus", "or", "minus", "about", "million", "years."]

No primeiro exemplo, estamos tratando cada caractere de pontuação individual como seu próprio token. Então, você pode ver que a vírgula e o ponto são tokens independentes. No segundo, estamos simplesmente removemos os tokens de pontuação completamente, então não estamos interessados em analisá-los mais. E no terceiro, estamos apenas combinando a pontuação com a palavra ao lado, então a vírgula aparece dentro do token de **old** e o ponto aparece dentro do token de ***years***. Também estamos mantendo todos os números, então 50 e 4,54. Na versão final, não estamos apenas agregando a pontuação com as palavras, mas também removendo os próprios números. Desta forma, você pode ver todas essas versões, e claro, você pode pensar em outras e combinar diferentes partes da frase, você pode remover os números, mas mantendo a pontuação separada, você pode manter tudo e agregar as pontuações ou não e assim por diante.

As escolhas que você fizer aqui, é claro, terão um impacto no futuro e limitarão a análise que você pode fazer. Portanto, dependendo da sua aplicação, uma opção pode ser mais aplicável ou mais acessível do que a outra. Um dos pacotes que usaremos extensivamente nesta aula é o NLTK, significa o pacote de ferramentas de linguagem natural, ele dará um suporte bastante extenso para tokenização que permite alternar facilmente de uma definição para outra. Tudo isso fica dentro do submódulo **nltk.tokenize**. E tem duas funções utilitárias que facilitam muito a sua vida. Uma é **word_tokenize**, e isso simplesmente dividirá o texto em todas as palavras e pontuações, para que as pontuações sejam tokens individuais. Enquanto o **sent_tokenize** apenas dividirá seu texto em frases individuais. Então, se você está tentando fazer algum tipo de análise de nível de frase, você usará **sent_tokenize**, se você quiser fazer análise de nível de palavra, você usará **word_tokenize**.

Na verdade, essas são apenas funções utilitárias que, em segundo plano, usam um dos muitos objetos tokenizer, módulos tokenizer que o nltk inclui. Vou listar abaixo algumas outras funções que podem ser úteis para aplicações específicas:

RegexTokenizer

- Especifica que tipo de expressão regular usar

TweetTokenizer

- Especialmente projetado para lidar com tweets e outros textos de mídia social
- >>> tknzr = TweetTokenizer()
- >>> s0 = "This is a cooool #dummysmiley: :-) :-P <3 and some arrows < > -> <--"
- >>> tknzr.tokenize(s0)
- ['This', 'is', 'a', 'cooool', '#dummysmiley', ':', ':-)', ':-P', '<3', 'and', 'some', 'arrows', '<', '>', '->', '<--']

WhitespaceTokenizer

- Apenas divide a string ou seu texto quando tudo vem no caractere de espaço em branco e assim por diante
- >>> tk = WhitespaceTokenizer()
- >>> gfg = "The price**\t** of burger **\n**in BurgerKing is Rs.36. "
- >>> tk.tokenize(gfg)
- ['The', 'price', 'of', 'burger', 'in', 'BurgerKing', 'is', 'Rs.36.']

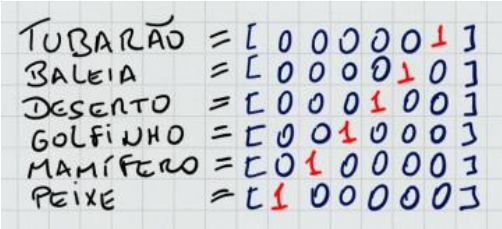
PunktSentenceTokenizer

- Esse tokenizer divide um texto em uma lista de frases usando um algoritmo não supervisionado para construir um modelo para abreviação de palavras, colocações e palavras que iniciam frases. Ele deve ser treinado em uma grande coleção de texto simples no idioma de destino antes que possa ser usado.O divisor de sentenças deve remover o espaço branco posterior ao limite da sentença.
- >>> pst = PunktSentenceTokenizer()
- >>> pst.tokenize('See Section 3). Or Section 2). ')
- ['See Section 3).', 'Or Section 2).']

Agora que definimos quais são nossos tokens, de acordo com qualquer definição que desejarmos, podemos seguir em frente e descobrir como podemos representar isso de uma maneira significativa, de uma maneira que economize tempo e facilite nossa vida. Normalmente, o que queremos fazer é encontrar **algum tipo de representação numérica para cada uma das palavras ou cada um dos tokens que estamos considerando**.

A razão para isso, é claro, é que os computadores são muito bons em processar números, mas não tão bons em lidar com texto. Então, queremos trapacear um pouco e substituir o texto por números para facilitar nossas vidas e tornar nossos algoritmos e nossos aplicativos mais eficazes. Podemos simplesmente **atribuir um ID numérico para cada palavra**, o problema com isso é que os computadores estão acostumados com números e eles não necessariamente entendem que estamos apenas usando os números como uma referência, então é possível, acidentalmente, fazer coisas como subtrair um ID numérico de outro, e isso resultaria no terceiro ID numérico. Assim, você pode criar erros sutis que não necessariamente notaria, a menos que estivesse analisando seu código com muito cuidado.

Uma maneira de evitar isso é **usar vetores em vez de números**. E é aí que entra a **codificação one-hot**. Então aqui, essencialmente o que estamos fazendo é definindo que nossas palavras **vivem neste espaço vetorial de alta dimensão**. Esse espaço é tão dimensionalmente alto que tem o mesmo número de dimensões que nosso vocabulário tem tokens, ou vocabulário tem palavras. Assim, **cada palavra corresponderá a uma única dimensão, e nossos vetores que representam nossa palavra terão zeros em todos os lugares, exceto na posição correspondente a essa palavra**.



Então, temos aqui alguns exemplos, o vetor para a palavra "baleia" tem zeros em todos os lugares, exceto no quinto elemento do vetor, enquanto o vetor para a palavra "deserto" tem zeros em todos os lugares, exceto na quarta posição. Desta forma, esta seria a representação codificada dessas palavras em algum contexto ou em algum corpus ou em alguns textos específicos. Uma coisa a ter em mente aqui é que **os vetores de codificação one-hot na verdade não têm informações semânticas**, certo? **Não há nada sobre o significado da palavra codificada nessa representação**.

Desta forma, o que você está fazendo é criar um espaço vetorial com o mesmo número de dimensões que o número de palavras ou tokens exclusivos que você possui. Você está atribuindo uma ordem a todas as palavras, de forma que cada palavra tenha uma posição única no espaço vetorial e, em seguida, você está gerando esses vetores de zeros e uns com os elementos no lugar certo.

Se formos analisar um texto com várias palavras, toda vez que uma palavra é repetida, você repete o valor 1 na mesma posição do vetor. Você está percebendo que esta é uma representação muito inútil. Temos cerca de meia dúzia de palavras únicas, o que significa que temos muitos zeros em cada uma das linhas. Então usando a codificação one-hot você desperdiçará muito espaço apenas armazenando zeros. Uma forma de obter a representação codificada do texto completo e somar todos os vetores de cada palavra do texto. Esta é a soma vetorial de todas as palavras, então na representação esta será apenas a soma ao longo das colunas, o que essencialmente resulta em um vetor que mostra para cada posição a quantidade de vezes que uma palavra aparece no texto.

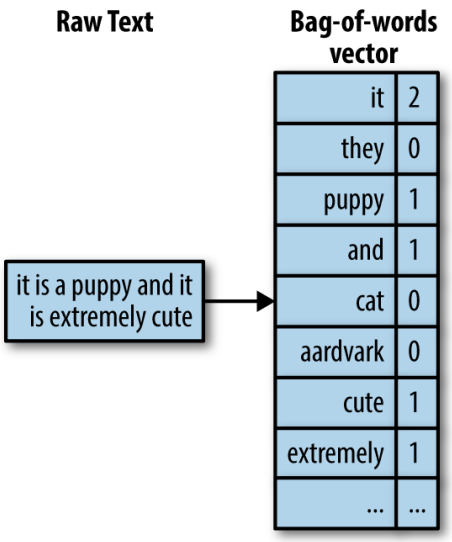
Aqui notamos uma das muitas trocas (tradeoffs) que muitas vezes temos que fazer, que é entre armazenar mais informações e tornar as coisas mais eficazes. Podemos representar nosso texto nessa representação simplificada onde estamos mantendo apenas as contagens de cada palavra desse vetor, mas estamos perdendo qualquer informação que tínhamos sobre a sequência em que a palavra aparece, certo? Estamos apenas mantendo quantas vezes cada palavra ocorre no texto, não onde ela ocorre. Assim, estamos economizando muita memória, estamos perdendo algumas informações, o que novamente, dependendo da sua aplicação, pode ou não ser um problema.

A codificação one-hot é um procedimento de PLN padrão, portanto, você tem funcionalidade para ela em muitas bibliotecas. Pandas tem um método **pd.get_dummies** que irá gerar representações codificadas one-hot de séries, onde essencialmente, ele transformará uma série ou valores em um DataFrame onde cada coluna corresponderá a um valor exclusivo dessa categoria.

sklearn tem o transformador codificado one-hot (**OneHotEncoder**) que faz algo semelhante. Ambas as versões transformarão **um array em uma matriz de valores**, portanto, transformarão uma sequência de rótulos ou uma sequência de valores categóricos em um matriz.

nltk em particular faz isso de uma maneira diferente porque está otimizando as coisas ainda mais, e está representando **os vetores codificados one-hot ou simplesmente dicionários** onde a chave (key) é o token ou a palavra, e é mapeado para um valor verdadeiro.

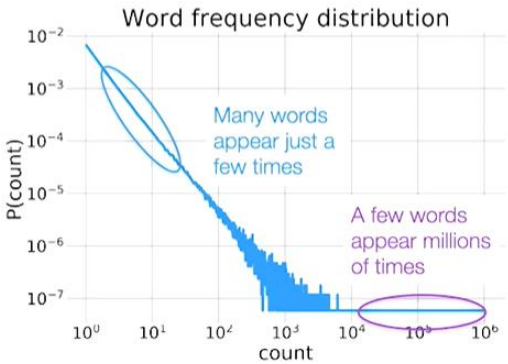
Bag of Word (saco de palavras)



O próximo passo em termos de refinamento é conhecido como **bag-of-words**. Quando mencionamos a forma como o NLTK representa um vetor codificado, configurando-os como um dicionário, esta é uma forma de economizar espaço, de economizar novamente memória. Isso torna sua vida significativamente mais fácil, mas, novamente, você também está perdendo algumas informações. Então, agora, se você apenas usar o dicionário para combinar as chaves individuais com valores verdadeiros ou falsos ou 20 outros valores, você está perdendo qualquer informação que possa ter sobre a totalidade do vocabulário, quais são as diferentes dimensões, quantas você tem, embora você seja mais eficaz e mais eficiente no processamento de seu texto.

Isso é conhecido como saco de palavras, porque essencialmente **você está apenas jogando todas as palavras em um saco e perdendo qualquer informação sobre qualquer relacionamento entre elas**. E isso, claro, é muito mais conveniente. Portanto, a representação do conjunto de palavras, que estamos mostrando aqui acima, é simplesmente um dicionário ou, um Data Frame que **mapeia uma palavra individual para o número de vezes que essa palavra acontece**. Não há informação da ordem em que as chaves deste dicionário aparecem. Entretanto, você pode gerar representações semelhantes de outros documentos, e isso facilita a comparação simplesmente fazendo junções numéricas nos diferentes dicionários.

Stop-Words



Vamos tratar agora do conceito de palavras de parada (stop-words). Naturalmente, esperamos que algumas palavras sejam **muito mais comuns do que outras**. Algumas ocorrem em praticamente todas as frases, enquanto outras são muito, muito, muito mais raros. Aqui no lado direito, traçamos a distribuição das palavras em inglês. No histograma normalizado de frequência de palavras, podemos ver que muitas palavras aparecem poucas vezes.

Esse gráfico é baseado em um rico corpus de 17 milhões de palavras coletadas da edição em inglês da Wikipedia. E vemos no canto inferior direito, que as poucas palavras **aparecerão milhões de vezes**. Claro que são palavras que reproduzem algum tipo de linha gramatical como

the, of, and, one, in etc. Estas são conhecidas como palavras de parada ou stop words. Essas palavras que estão incluídas no texto, são necessárias para fins gramaticais, mas não carregam nenhum significado, e não fornecem nenhuma informação sobre qual pode ser o tópico do texto, do que está falando e assim por diante.

Portanto, muitas vezes o que fazemos é **remover ou descartar essas palavras**. Se no exemplo anterior do corpus com 17 milhões de palavras, simplesmente removemos as **100 palavras mais comuns**. Reduzimos nosso conjunto de dados de 17 milhões de palavras para apenas 9 milhões. Portanto, quase 50% de redução, **perdendo muito pouca informação**. Claro, dependendo do nosso aplicativo, você não quer apenas remover as X palavras principais, as palavras mais comuns. Você quer fazê-lo de uma maneira mais cuidadosa. E ao longo dos anos, linguistas computacionais e cientistas da computação publicaram listas de palavras de parada para vários idiomas e para vários propósitos.

Você pode até mesmo gerar seu próprio conjunto de palavras de parada para sua aplicação específica, ou pode simplesmente decidir não usar nenhuma palavra de parada. O objetivo de removê-los, é claro, é reduzir o número total de palavras, reduzir o tamanho do seu vocabulário e, ao mesmo tempo, perder o mínimo de informações possível. Obviamente, as palavras de parada dependerão do idioma que você está usando. A palavra **the** é uma palavra de parada em inglês, mas provavelmente não em mandarim. Então você tem que ter listas para todos os idiomas que você está interessado. **O NLTK facilita nossas vidas e nos fornece palavras de parada para 23 idiomas diferentes que você pode acessar com muita facilidade.** Você também pode adicionar sua própria lista de palavras de parada simplesmente adicionando um arquivo de texto ao diretório correto em seu disco rígido. E então você pode acessar a lista de palavras de parada usando o nome do arquivo.

TF-IDF

Vamos discutir TF-IDFs (term frequency - document inverse frequency), que são dois conceitos muito inter-relacionados que **nos permitem quantificar o quão importante ou relevante uma palavra específica é para um determinado documento**, ou mesmo para um corpus de documentos. Já estamos um pouco familiarizados com a ideia de frequência de termo. A frequência do termo é **simplesmente o número de vezes que uma palavra ocorre dentro do documento**. Já verificamos que existem algumas palavras que são muito mais comuns que outras, como vimos em nosso grande corpus de 17 milhões de palavras, então já calculamos essencialmente a frequência do termo.

A ideia de frequência de termos, ao usá-la como forma de **quantificar a importância das palavras** para um determinado documento, é simplesmente que **as palavras que serão mais comuns dentro do documento serão as mais relacionadas após removermos as palavras de parada**, ao tema do texto. Assim, no livro sobre programação Python, as palavras "código" e "script", "print", "import" e assim por diante aparecerão muitas vezes. Bem, você provavelmente não verá muitas dessas palavras em um livro sobre futebol ou futebol. Frequência do termo, TF, nos dá uma ideia de quão popular um termo específico é dentro do documento. Mas então a questão permanece, como podemos comparar informação entre documentos?

Então ... quando temos um corpus de vários documentos, que pode ser até dezenas de milhares ou milhões de documentos, se você pensar em termos de rede mundial, como podemos compará-los e ver quais palavras são mais importantes para quais documentos? É aí que entra a **frequência inversa de documentos**, que essencialmente nos diz, ou tenta quantificar, quão incomum é que em nosso corpus haja um documento com essa palavra. Então é chamado de frequência inversa do documento porque é um sobre a frequência do documento, e a frequência do documento é **o número de documentos em que essa palavra aparece**. E para isso, o número de vezes que aparece no documento não é relevante, apenas em quantos documentos diferentes ela aparece.

A ideia, claro, é que se você tem palavras que aparecem em apenas alguns documentos, elas são muito mais importantes para esses documentos do que para todos os outros, e elas são uma pista importante para nos ajudar a distinguir esses documentos dos outros. Então TF-IDF é quando combinamos esses dois termos via. Essencialmente, vamos multiplicar o termo frequência, quão importante é a palavra para este documento específico, vezes a frequência inversa do documento, o que nos diz o quão incomum é que essa palavra exista em um documento do nosso corpus. Portanto, idealmente, você deseja identificar palavras importantes no documento e palavras que o ajudem a distinguir este documento de outros.

Naturalmente, existem diferentes maneiras de definir a frequência do termo e a frequência do termo inverso. A definição que usamos até agora de frequência de termo é simplesmente o número de vezes que uma palavra aparece em documentos individuais, então estamos chamando isso de N_t de d. Você também pode usar uma versão normalizada, que é simplesmente a fração do número total de palavras neste documento que essa palavra específica representa, portanto, apenas N_t de d dividido pela soma total em todos os termos deste documento.

$$TF(t,d) = \frac{Número\ de\ ocorrências\ do\ termo\ \boldsymbol{t}\ no\ documento\ \boldsymbol{d}}{Total\ de\ termos\ no\ documento\ \boldsymbol{d}}$$

Como vimos, a frequência inversa do documento é simplesmente um sobre a frequência, o que significa que é um sobre o termo do número de documentos, T, aparece dividido pelo número total de documentos. Por conveniência matemática, porque em geral N, o número total de documentos, será muito maior que o número de documentos em que uma palavra específica aparece, este será um número relativamente grande quando você dividir N por N_t, então aplicamos um logaritmo para nos ajudar a reduzir esse número. O logaritmo tem uma consequência importante, é que se uma palavra aparece em cada documento, então N_t é igual a N, logo N sobre N_t será um, e o logaritmo de um é sempre zero. Então, o que isso significa é que uma palavra que aparece em cada documento em nosso corpus terá um peso zero, terá uma pontuação TF-IDF zero. Ou seja, intrinsecamente estamos removendo palavras que aparecem em todos os documentos, então é como se as tratássemos como se fossem palavras de parada.

$$IDF(t) = log_e \frac{Número\ total\ de\ documentos\ em\ um\ corpos}{Número\ de\ documentos\ em\ que\ o\ termo\ \boldsymbol{t}\ aparece}$$

Se, por algum motivo, isso não for algo que queremos, então, se quisermos manter todas as palavras e garantir que todas tenham um peso diferente de zero, o que pode ser importante dependendo exatamente de qual aplicativo estamos fazendo, modificamos nossa definição ligeiramente, e em vez de fazer o logaritmo de N sobre N_t, fazemos o logaritmo de um mais N sobre N_t. E isso garante que o valor mínimo dentro do logaritmo, o argumento mínimo do logaritmo, será dois em vez de um.

Então, quando combinamos esses dois termos, criamos esse conceito, o conceito de TF-IDF, que nos dá **uma definição ou um peso e a importância relativa de quão importante uma palavra específica é para um documento específico**. Como mencionei, as palavras que aparecem em todos os documentos não têm sentido quando estamos tentando distinguir os documentos, então não precisamos nos preocupar tanto com elas.

Limpeza de Texto (Text Cleaning)

Stemming

Vimos algumas representações básicas e não tão básicas de texto e palavras para que pudéssemos analisá-las e processá-las de maneira mais fácil e automatizada. Neste bloco da aula, veremos como podemos limpar o texto, reduzindo **a complexidade**, **o tamanho do vocabulário**, ou **o número de tokens** que temos que manipular. Vou começar analisando o stemming, que é essencialmente **a ideia de "desconjugar" palavras**.

O conceito é muito simples. Encontramos ao longo de nossas vidas, sempre que estamos processando ou lendo uma grande quantidade de variações de uma palavra individual. Portanto, isso pode ser **plurais, substantivos, advérbios, conjugações verbais para tempo e assim por diante**. E na maioria das aplicações, não precisamos dessa quantidade de detalhes. Não precisamos saber qual é o tempo verbal, ou se está sendo usado como um advérbio, ou um substantivo, ou assim por diante.

Stemming é uma ferramenta e essencialmente **um conjunto de heurísticas** que nos permite "desconjugar" uma palavra, de modo a mapear a palavra da variante conjugada na palavra **raiz**, ou **o radical da palavra**, que é mais significativo, e nos permite **reduzir** significativamente o **tamanho do nosso vocabulário** se reduzirmos a quantidade de tokens que temos, enquanto novamente perdemos o mínimo de informação possível.

Vocabulários, é claro, podem ser extremamente grandes. Portanto, estima-se que o inglês tenha mais de um milhão de palavras únicas, com todas as variantes. Várias técnicas foram desenvolvidas para minimizar a quantidade de palavras para minimizar o tamanho do vocabulário. Stemming, que é o que estamos considerando agora, está tentando usar algumas heurísticas. E aqui, heurística significa que são **regras práticas e não as regras gramaticais precisas**, mas que funcionam bem. E não é garantido que o resultado seja uma palavra natural, mas é garantido que será único e consistente, de modo que a mesma lei da palavra original seja mapeada para o mesmo radical. Uma abordagem mais sofisticada que veremos na próxima parte desta aula é a lematização. A lematização tenta usar parte da análise de fala também, e outras técnicas mais sofisticadas para realmente identificar qual é a origem da palavra, qual é a palavra original real, então é garantido sempre mapeá-la para **o lema correto**, sempre mapeá-la para o palavra originária. Claro, é mais complexo, mais computacionalmente intensivo e requer essencialmente um grande mapeamento de palavras e suas conjugações. E, finalmente, palavras de parada, que já vimos, que essencialmente tentam remover todas as palavras comuns que não são semanticamente significativas.

Stemming, como está tentando "desconjugar" palavras, é claro que depende extremamente do idioma real que estamos considerando, o que significa que as regras que funcionam para o inglês não funcionarão para o espanhol, mandarim ou russo. Assim, cada linguagem terá seu próprio algoritmo específico ou seu próprio modelo treinado especificamente para aquela linguagem.

Para o inglês, o primeiro algoritmo e o mais conhecido é o **PorterStemmer**, mas é claro que existem muitas outras variantes. Em particular, o **nltk** suporta um punhado de diferentes lematizadores, alguns deles trabalhando e se especializando apenas em um ou dois idiomas, outros sendo capazes de lidar com mais de um. Em particular, vamos olhar com mais cuidado para os quatro que são capazes de lidar com inglês, então **LancasterStemmer**, **PorterStemmer**, **RegexpStemmer**, que basicamente usa expressões regulares, e finalmente o **SnowballStemmer**, que é uma abordagem mais geral que é realmente capaz de lidar com vários idiomas que são muito diferentes entre si.

	LancasterStemmer	PorterStemmer	RegexpStemmer	SnowballStemmer
playing	play	play	play	play
loved	lov	love	loved	love
ran	ran	ran	ran	ran
river	riv	river	river	river
friendships	friend	friendship	friendship	friendship
misunderstanding	misunderstand	misunderstand	misunderstand	misunderstand
trouble	troubl	troubl	troubl	troubl
troubling	troubl	troubl	troubl	troubl

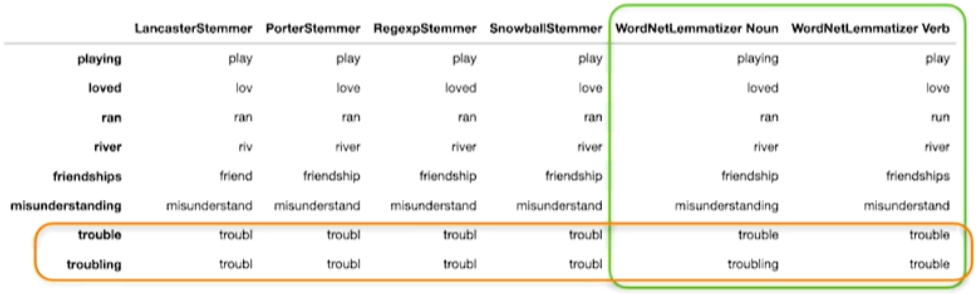
Na tabela acima, o que temos são oito palavras que são, é claro, conjugações de algum conceito, e como esses diferentes stemmers processam essas palavras e o que elas obtêm. Uma coisa interessante a ser observada, é claro, e estamos destacando isso nas duas linhas inferiores, é que o resultado da divisão não é necessariamente uma palavra real. Então você vê aqui como "trouble" e "troubling" são mapeados sempre para "troubl" sem o "e" no final. Por outro lado, você também pode ver como alguns stemmers parecem funcionar e fazer um trabalho melhor do que outros. Por exemplo, temos na segunda linha o LancasterStemmer e o PorterStemmer são capazes de processar "loved" e estão mapeando-o para "love", enquanto o RegexpStemmer, por exemplo, apenas o mantém como "loved".

Lemmatization

Na seção, veremos a lematização, que é essencialmente **uma maneira mais sofisticada de fazer a derivação de palavras não conjugadas**. A forma como a lematização funciona é que ela tem um **entendimento interno de como as palavras se relacionam** com outras palavras e como elas são conjugadas com base em sua função gramatical específica. Essencialmente, o que eles fazem parte do discurso a que correspondem é a palavra sendo usada como substantivo, como advérbio ou verbo e assim por diante. E o nltk nos fornece o **WordNetLematizer**, que essencialmente lematizador que usa o **WordNet**, que é um grande banco de dados de como conceitos e palavras se relacionam entre si para fazer esse processo de lematização para nós.

Na figura ao lando, temos um meme onde ele mostra meio que brincando como o fato de a palavra estar sendo usada como verbo ou adjetivo é realmente muito importante e pode mudar significativamente o significado do que você está dizendo. Não há lematizadores em que seja garantido retomar a palavra "real". Mas é claro que os resultados dependem de você fornecer a parte correta da **marcação de fala**. Depende de você passar os parâmetros corretos ao algoritmo, informando ao lematizador se esta palavra está sendo usada como verbo, substantivo, advérbio ou assim por diante. Porque isso produzirá resultados diferentes e o "desconjugará" de maneiras diferentes.

Na tabela abaixo, estendemos essencialmente a tabela que mostramos antes e agora com duas colunas extras que destacam a importância dessa parte da marcação de fala para os resultados obtidos com o WordNetLematizer.



	LancasterStemmer	PorterStemmer	RegexpStemmer	SnowballStemmer	WordNetLemmatizer Noun	WordNetLemmatizer Verb
playing	play	play	play	play	playing	play
loved	lov	love	loved	love	loved	love
ran	ran	ran	ran	ran	ran	run
river	riv	river	river	river	river	river
friendships	friend	friendship	friendship	friendship	friendship	friendships
misunderstanding	misunderstand	misunderstand	misunderstand	misunderstand	misunderstanding	misunderstand
trouble	troubl	troubl	troubl	troubl	trouble	trouble
troubling	troubl	troubl	troubl	troubl	troubling	trouble

Então temos o mesmo conjunto de palavras, as mesmas oito palavras ou algo assim. Mas agora em uma coluna que destacamos aqui na caixa verde, estamos dizendo ao WordNetLematizer que esta palavra está sendo usada como um substantivo, enquanto no lado direito estamos dizendo que está sendo usada como um verbo. Então, você pode ver o quão importante é, que você tenha a parte correta da tag de fala para a palavra. Assim, você tem, por exemplo, na terceira linha, a palavra "ran". Então é claro que corresponde ao passado do verbo "to run". Se você disser que é um substantivo, então ele não sabe como desconjugar e o preserva como executado.

Por outro lado, se você disser que é um verbo, ele retorna corretamente a versão definitiva do verbo. Duas linhas abaixo. você tem a palavra "freindships" (amizades). Então, no plural, você notará que alguns lematizadores como o Lancaster Stemmer são realmente capazes de obter a palavra amigo em vez de apenas amizade. Para o lematizador, se você disser que é um substantivo, então ele apenas diz, ok, isso é um substantivo. Então o "s" no final representa um plural. Então é capaz de remover o plural e apenas retornar "frindship" enquanto se você disser que é um verbo, então, novamente, ele não sabe como desconjugar ele mantém como está.

O poder dessa abordagem é ainda mais óbvio quando olhamos para as duas linhas finais onde você tem a interseção entre a caixa laranja e a caixa verde. Essas são as duas variantes do mesmo verbo. Então, quando você está tentando lemitizar a palavra problema e dizendo ao lematizador que isso é um substantivo, ele mantém isso constante. Se você disser que é um verbo, ele tem condições de encontrar o elemento base da palavra.

Você ainda está tentando essencialmente desconjugar palavras e obter o radical ou a raiz da palavra, mas dependendo do algoritmo que estiver usando, alguns funcionarão melhor que outros em circunstâncias específicas. Portanto, é sempre importante que você experimente e experimente diferentes e esteja familiarizado com as diferentes abordagens e seus prós e contras.

Expressão regulares:

Nesta seção final, veremos expressões regulares. Expressões regulares, é claro, são um mundo a parte. Elas são extremamente complexas e permitem que você faça coisas muito complexas e muito sofisticadas. Aqui só temos tempo para uma rápida visão geral das ideias mais fundamentais, mas isso deve ser mais do que suficiente para colocá-la em funcionamento.

Expressões regulares, às vezes chamadas de REs, regexes, padrões regex e assim por diante, são essencialmente uma própria linguagem de programação altamente especializada. Então você está essencialmente escrevendo este pequeno pedaço de código em uma linguagem totalmente diferente, que na verdade é suportada, praticamente por todas as outras linguagens de programação. Então você pode fazer regexes em Python, Perl, C, PHP, Java, qualquer linguagem que você queira.

Esta é uma funcionalidade central em muitos desses idiomas, porque permite que você faça um processamento de texto bastante sofisticado. Python em particular usa o módulo RE ou nos fornece o módulo RE, que tem essa funcionalidade. Uma das vantagens e por que você gostaria de gastar o tempo para aprender sobre expressões regulares, é que, em segundo plano, todas elas são implementadas em C. Portanto, elas **são extremamente rápidas**. Assim, se você precisar **processar texto e encontrar correspondências e substituir coisas**, essa é a maneira mais rápida de fazer isso normalmente.

As regexes funcionam da seguinte forma, você especifica ou define escrevendo seu próprio pequeno pedaço de código como **uma string especial**. A string especifica quais são as regras para identificar correspondências em um pedaço de texto. Em geral, a maioria dos caracteres dentro dessa string mágica e especial do projeto, apenas representam a si mesmos. Mas é claro que você tem vários metacaracteres ou vários caracteres especiais, que realmente desempenham um papel importante. Nesta tabela, temos um breve resumo de alguns dos metacaracteres ou caracteres especiais mais comuns em regexes.

Padrão	Significado
^	Nega a expressão OU marca o início de uma string
$	Marca o final de uma string
+	Prolonga o caractere anterior
*	Prolonga o caractere anterior, mas ele também pode não existir
?	Diz que há dúvida se o caractere anterior a ele existe
.	Substitui qualquer caractere (apenas um)
[A-Z]	Procura qualquer caractere maiúsculo de **A** a **Z**
[a-z]	Procura qualquer caractere minúsculo de **a** a **z**
[0-9]	Procura qualquer dígito de **0** a **9**
[125]	Procura os dígitos **1**, **2** ou **5**
[^0-9]	O ^ nega a expressão a seguir, logo, procura tudo que não for número.
[abc]	Procura os caracteres **a**, **b** ou **c**
[^A-Z]	Procura tudo que não for letra maiúscula
[a-zA-Z]	Procura tudo que for letra
\s	Procura espaços
\w	Procura caracteres, exceto os especiais
\d	Também procura qualquer dígito de **0** a **9**

Os colchetes permitem a inserção de um conjunto de caracteres que se deseja procurar uma correspondência. Dentro dos colchetes são colocados os caracteres permitidos para a correspondência.

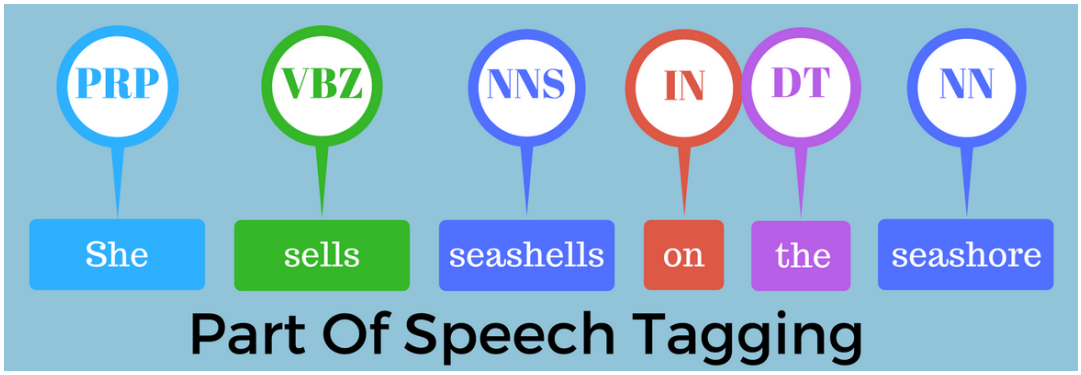
Named Entity Recognition

Part of Speech Tagging (PoS - Marcação de fala)

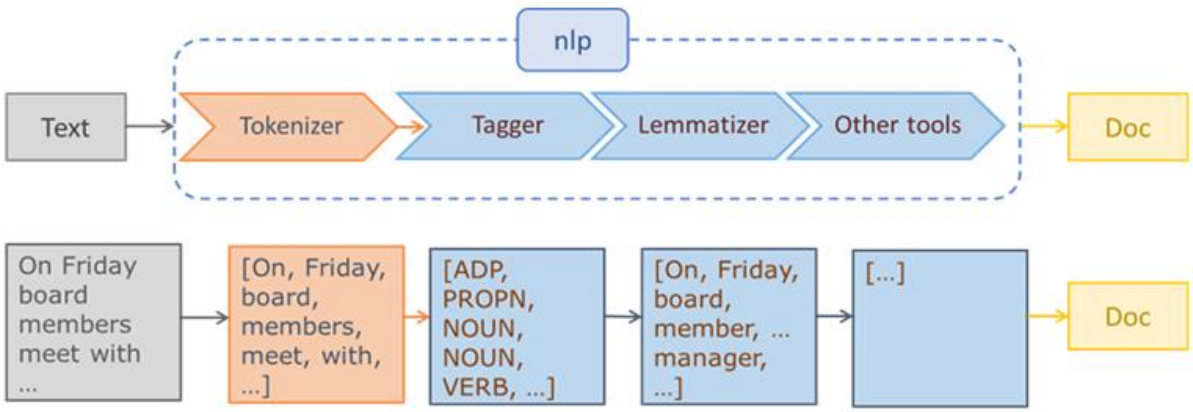
Nesta parte da aula, veremos como podemos realizar o reconhecimento de entidade nomeada (NER). E o que isso significa essencialmente é: **como podemos processar automaticamente um pedaço de texto e extrair entidades nomeadas?** Essas podem ser pessoas, empresas, países e assim por diante. O primeiro passo é olhar para a parte da **marcação de fala**, porque esse é um dos componentes fundamentais antes que possamos chegar ao reconhecimento de entidade nomeada completo.

Como você deve se lembrar da escola primária, diferentes partes de uma frase desempenhavam diferentes papéis gramaticais. Então, neste exemplo simples, "o cachorro chutou a bola", talvez tenhamos a estrutura de frases mais comum, que é sujeito, verbo, objeto, certo? O sujeito faz alguma coisa, que é o verbo, a alguma coisa, que é o objeto. E cada um deles, de alguma forma, desempenha um papel diferente na transmissão da mensagem do que está acontecendo. Parte da marcação de fala é essencialmente um conjunto de algoritmos que tenta **identificar automaticamente esse papel específico que cada uma dessas peças está desempenhando**.

Portanto, é uma etapa comum de pré-processamento que identifica e marca cada token, cada palavra como o papel que eles estão desempenhando ativamente nessa frase específica. E isso torna outros algoritmos a jusante (que vem depois na sequência) significativamente mais eficientes. Já vimos, por exemplo, que a lematização exige que saibamos qual é a parte do discurso. Portanto, temos que fornecer as partes da marcação de fala para esse algoritmo. E o nltk tem uma função utilitária chamada **pos_tag** que automaticamente fará parte da marcação de fala para nós. No fundo, esta é apenas uma função de conveniência para o conjunto de tags **Penn Treebank** para inglês. Há também a versão russa do conjunto de tags **Russian National Corpus**.



Você também pode usar outros conjuntos de tags ou treinar seus próprios modelos. O conjunto de tags Penn Treebank, como mencionei, é o padrão para nltk, é capaz de reconhecer várias partes muito específicas do discurso. Aqui estão apenas alguns dos mais comuns. DT, a tag que corresponde ao determinador. VB corresponde a um verbo. NN representa um substantivo. RB é um advérbio. E JJ representa adjetivos. Você pode obter uma descrição completa de todas as tags, como mencionei, existem várias dezenas delas, apenas chamando **nltk.help.upenn_tagset**.



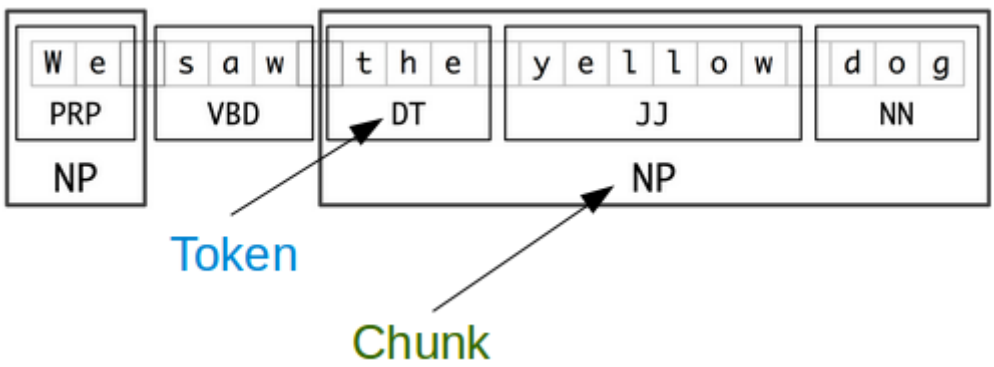
POS_tag espera receber tokens diretamente, entradas tokenizadas em vez de strings brutas. Portanto, você verá frequentemente POS_tag(word_tokenize). Para cada palavra ou para cada token, ele atribui uma tag que especifica qual é a parte do discurso, qual é o papel que esse token específico está desempenhando nessa frase. Novamente, devo mencionar que todos esses algoritmos são específicos para cada linguagem. Portanto, se você quiser fazer parte da marcação de fala no idioma que não seja inglês ou russo, terá que usar algumas outras partes do marcador de fala, que você pode encontrar também para nltk.

O Nltk vem com uma ampla variedade de outras partes de taggers de fala que você pode especificar e usar por conta própria. Você tem o BrillTagger, ContextTagger, HiddenMarkovTagger, PerceptronTagger, que usa redes neurais em segundo plano e assim por diante. Você pode treinar seu próprio HiddenMarkovModel e BrillTaggers usando o objeto trainer. Para treinar qualquer tipo de modelo você precisa de um Corpus pré-marcado. Então você precisa do Corpus para cada token, onde você sabe qual é a parte correta da tag de fala, para que você possa usar isso para ensinar seu algoritmo a determinar exatamente qual é a marcação correta.

Chunking

Agora que você entende a marcação de fala, vamos ver a segmentação ou chunking que é **uma maneira de combinar tokens que** desempenham um papel semelhante que funcionam juntos da mesma maneira. O que é semelhante em conceito também à ideia de n-grams. Portanto, o objetivo do chunking é identificar frases em nosso texto. "Um cordeirinho" e "planeta Terra", estes correspondem aos n-gramas que estávamos tentando identificar. **Chunking depende de partes da marcação de fala.**

A maneira como funciona é você especificar uma gramática ou um conjunto de regras que corresponda a parte das tags de fala em uma nova tag e que corresponda ao pedaço que você está identificando. Infelizmente, não há uma maneira objetiva de definir pedaços. Não há nada escrito em pedra e isso vale essencialmente para qualquer coisa relacionada à linguagem. Então você tem que definir seus próprios padrões de tags. Portanto, essas são uma variante simplificada das expressões regulares. Assim, ele usa algumas das mesmas notações, como pontos de interrogação, estrelas e assim por diante, especialmente para os caracteres métricos onde você pode identificar sequências ou combinações de partes de tags de fala. E você delimita as partes das tags de fala entre colchetes e isso deixa claro o que deve ser uma tag POS e o que deve fazer parte da "expressão regular" ou do padrão de tag.



Então vamos ver um exemplo. Aqui vamos definir uma nova tag **NP** que é definida como sendo um determinante opcional. Portanto, zero ou um determinante, seguido por zero ou mais adjetivos e um ou mais substantivos. Então, estamos procurando sequências de tokens que estão modificando e identificando um substantivo específico. Vamos aplicar isso a uma frase simples. Neste caso, "We saw the yelloen dog". "We"

é um pronome, "saw" é um verbo, "the" é um determinante, "yellow" é um adjetivo e "dog" é um nome (substantivo). Assim podemos compor os chunks caracterizados por NP na figura acima.

Você pode ver como o chunking é essencialmente **agrupar as coisas para tornar nossas vidas mais fáceis** quando terminamos de processar nosso texto. As gramáticas podem ficar bastante complexas. Uma coisa que devo observar é que toda a gramática é escrita dentro de uma única string. Sempre recomendado usar uma string bruta, portanto, o r no início da string. E a notação é semelhante à notação de dicionário em Python. Mas é claro que está tudo dentro da string. Vejamos um exemplo:

```
grammar = r"""
        NP: {<DT>?<JJ.*>*<NN.*>+}
        PP: {<IN><NP>}
        VP: {<VB.*><NP|PP|CLAUSE>+$}
        CLAUSE: {<NP><VP>}"""
```

Então você tem a tag que você está criando seguido por dois pontos e, em seguida, o pedaço entre chaves. Esse pedaço está se comportando e é especificado nesta notação de padrão de tag, que é semelhante à notação de expressões regulares como mencionamos. Aqui temos quatro tags, temos NP, PP, VP e cláusulas. E você também pode notar que temos tags que são definidas em função de outras tags. Então você pode realmente ter estruturas recursivas para que, quando você aplicar a mesma gramática à mesma frase, encontre uma estrutura completamente diferente.

Chinking

O próximo conceito que temos que entender é o de chinking. Então, chinking é realmente muito semelhante, semelhante ao conceito de chunking, mas é exatamente o oposto. No chunking, você está combinando as coisas! Em chinking, **você está separando-as e é essencialmente uma maneira mais simples de especificar como agrupar as coisas**. O que você está fazendo é, em vez de definir o que incluir em cada "pedaço", você está definindo e especificando **o que excluir**.

Modelagem de tópicos

Explicit Semantic Analysis

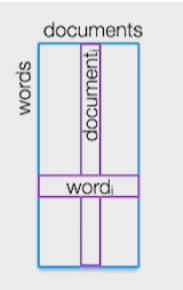
Agora que vimos como fazer alguma representação de texto, partes de marcação de fala e reconhecimento de entidade nomeada. Podemos começar a avançar para tópicos mais interessantes e avançados. Em particular, veremos **a modelagem de tópicos** e diferentes abordagens sobre como identificar o assunto que está sendo discutido em um texto específico ou um corpus específico.

Vamos começar olhando para a **análise semântica explícita**, que apesar de sua simplicidade, é na verdade um algoritmo muito poderoso que nos permite **identificar facilmente tópicos e documentos que cobrem os mesmos assuntos**. Como todos sabemos, um uso comum da PLN é para busca e extração de informações. Assim, pense em coisas como o Google que você está digitando uma consulta e está procurando por documentos que correspondam à consulta. Então, para fazer isso na escala do Google, na ordem de trilhões de documentos, temos que criar alguma **maneira inteligente de processar todos esses dados de maneira eficaz**

e identificar essencialmente quais são os principais tópicos ou os principais assuntos em cada documento ou em cada página da web neste caso.

Já vimos que algumas palavras são mais significativas do que outras. Alguns textos devem ser mais ou menos relevantes. E, em geral, esperamos que, se soubermos **quais palavras são mais importantes para um documento específico**, isso nos dê uma pista forte sobre o que esse documento está falando. Já vimos como podemos identificar e medir as pontuações via TF-IDF, que nos diz essencialmente para cada documento, qual é a palavra mais importante e quão incomum é encontrar essa palavra em qualquer documento. Agora vamos ver como podemos representar essencialmente todo o nosso corpus como **uma matriz de termos de documento**, que resume essencialmente todas essas informações de uma forma compacta e agradável, que podemos usar para outras aplicações.

Em particular, podemos construir essa matriz palavra por documento, onde cada linha corresponde a uma palavra, cada coluna corresponde ao documento e os pesos são os elementos da matriz. Portanto, os pequenos valores dentro de cada uma das células são a pontuação TF-IDF para essa palavra nesse documento. Então, essencialmente, estamos construindo essa matriz de pontuações do TF-IDF, que contém todas as informações, tudo o que sabemos sobre esse corpus e que podemos usar como entrada para nossos algoritmos daqui para frente.



Isso é conhecido como **matriz de termo de documento ou matriz de palavras de documento**. E aqui você pode pensar que cada linha desta matriz está sendo uma representação vetorial do significado dessa palavra. É possível pensar na palavra sendo definida pela força com que ela é representada em cada um dos documentos do nosso corpus. Por outro lado, você também pode pensar nos documentos como sendo definidos pelas palavras que contribuem para ele. Portanto, é uma espécie de relação complementar entre os dois e uma maneira complementar de olhar para o problema.

Para **análise semântica explícita**, normalmente, o que fazemos é construir essa grande matriz, essa grande matriz de documento de termo, onde temos um número muito grande de documentos, normalmente algo como a Wikipedia em inglês, por exemplo, que tem milhões de páginas, portanto milhões de documentos. E dada esta matriz, podemos procurar ou representar qualquer documento simplesmente tomando a soma ou a média de todas as palavras que esse documento contém. E estamos usando para representar cada palavra, claro, as linhas dessa matriz.

Então, essencialmente, estamos tratando as linhas da matriz de termos de documentos como se fossem uma incorporação de palavras (word embedings). Anteriormente mostramos ser possível representar palavras por vetores que de alguma forma contêm alguma informação sobre o significado da palavra, ou pelo menos alguma informação que nos permita distinguir uma palavra da outra. Você se lembra da codificação one-hot e o pacote de palavras (bag of words), que é uma maneira de combinar representações de palavras em representações vetoriais ao somar todos os vetores de palavras para as palavras nesse documento?

Portanto, esta é essencialmente a mesma abordagem. Agora estamos apenas redefinindo o que nossos vetores realmente são. Você pode medir a similaridade de palavras ou documentos simplesmente tomando a similaridade do cosseno, vendo quão próximos estão os dois vetores. Assim, o que isso está fazendo é medir o cosseno do ângulo entre dois vetores. A ideia é que se dois vetores são paralelos, o **cosseno será um**. Se eles são completamente perpendiculares, ou se eles estão apontando para posições opostas, em direções perpendiculares, então a similaridade do **cosseno será zero**. E isso nos diz como eles estão relacionados um com o outro.

O problema com a análise semântica explícita e outros tipos de aplicativos que estão diretamente relacionados a ela, é que, essencialmente, requer uma grande base de conhecimento neste caso, toda a Wikipedia em inglês, o que significa que você tem representações dimensionais muito altas.

Document Clustering

Se nos for dada uma representação vetorial das palavras nos documentos podemos medir suas semelhanças, ou seja, quão relacionadas são, usando a similaridade de cosseno, que mede o cosseno do ângulo entre as vetores. Uma aplicação interessante que podemos fazer é calcular a similaridade do cosseno de cada documento com qualquer outro documento. Isso nos dará uma matriz quadrada, que é simétrica, significando que a distância entre o vetor u e o vetor v, ou melhor, a semelhança entre o vetor u e o vetor v é a mesma que a semelhança entre o vetor v e vetor u.

$$\text{sim}\left(\vec{u}, \vec{v}\right) = \frac{\vec{u} \cdot \vec{v}}{||\vec{u}|| \, ||\vec{v}||}$$

Portanto, temos uma grande matriz quadrada simétrica que informa como cada documento está relacionado a todos os outros documentos. Essa matriz pode ser muito útil, e há uma enorme variedade de técnicas que podem usar essa matriz **para identificar clusters nos dados subjacentes**. Neste caso, para identificar clusters em nossos documentos. Mais adiante, veremos como esses agrupamentos correspondem essencialmente a tópicos, às ideias principais ou aos principais assuntos que os documentos discutem.

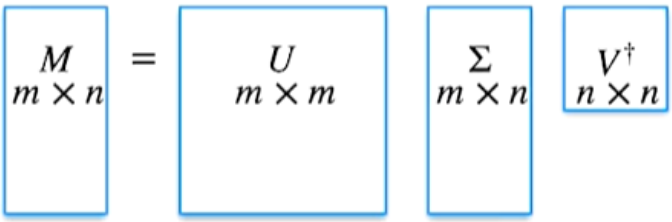
Uma abordagem mais simples que veremos é impor algum tipo de similaridade mínima e limite mínimo de corte e, essencialmente, considerar que qualquer par de documentos que tenha uma similaridade maior que esse limite pertence ao mesmo cluster. Então, **se dois documentos são muito semelhantes entre si, se suas representações vetoriais têm uma alta similaridade de cosseno, isso significa que eles provavelmente estão discutindo coisas semelhantes**. Esse é o tipo de intuição por trás desse conteúdo. O que nos traz naturalmente a ideia de agrupamento hierárquico de documentos.

Desta forma, você começa atribuindo cada documento ao seu próprio cluster, para que você tenha tantos clusters quanto documentos. Você calcula a semelhança, então quão semelhantes todos esses clusters são entre si. Você encontra os dois clusters mais semelhantes e os mescla. Então agora, você tem um cluster a menos e repete o processo. Conforme você repete o processo, você acompanha a similaridade em cada ponto, a similaridade vai fazendo com que você mescle os clusters. Quando você termina o algoritmo, acaba com apenas um único cluster que tem todos os documentos, mas nesse processo, você basicamente construiu essa árvore de relacionamentos entre documentos e a relação de cada documento com seus vizinhos. Então você pode, mais tarde, impor essencialmente um corte, em algum lugar e isso vai te ajudar a identificar os clusters para que você possa explorar mais facilmente os efeitos de ter diferentes tipos de documentos. Este é tipicamente um algoritmo usado em muitas outras aplicações, não apenas em PLN, mas também em aprendizado de máquina, classicamente também em álgebra linear para encontrar estrutura em matrizes e assim por diante. Portanto, existem muitas implementações de biblioteca de alta qualidade que você pode usar imediatamente.

Latent Semantic Analysis

O próximo algoritmo que vamos considerar é conhecido como análise semântica latente, e é chamado latente porque está tentando **encontrar uma representação interna dos dados**, e esse é um tópico relativamente comum em muitas, muitas abordagens. Para encontrar um latente interno, e aqui latente significa **espaço desconhecido**, no qual você **pode representar os dados que são mais significativos do que os que você já possui**.

A maneira como ele faz isso é começando com uma matriz de termos de documento e agora realizando **uma decomposição de valor singular** nessa grande matriz. Uma decomposição de valor singular, ou SVD, é uma técnica clássica da álgebra linear que essencialmente **decompõe uma matriz**, normalmente uma matriz retangular, portanto, mais linhas do que colunas, o que obviamente é o caso mais comum para matriz de termos de documento, porque você normalmente tem muito mais palavras do que documentos. Vamos transformar esta matriz M por N, onde M é o número de palavras, N é o número de documentos, em três outras matrizes, a matriz U, que é uma matriz quadrada, M por M, então palavra por palavra, a matriz sigma, que é uma matriz diagonal que terá N por N, então documento por documento, e a matriz de transposição V, que também é N por N, então documento por documento novamente.



Você pode pensar nessas três matrizes como representações das palavras e dos documentos nesse espaço latente interno, e a matriz sigma do meio que **define qual é a importância relativa de cada uma das dimensões**. A forma como o algoritmo é construído, **os valores na diagonal da matriz sigma são ordenados do mais alto para o mais baixo, então do mais importante, da contribuição mais forte para mais baixa**. O que fazemos, normalmente, é começar com a decomposição completa e, em seguida, truncamos as matrizes.

Então está dizendo que em vez de ter todas as dimensões, em vez de ter todos os vetores, todas as colunas, ou todas as linhas dessas matrizes, **vamos truncar essas representações da matriz em no máximo k colunas ou k linhas, certo?** Portanto, k colunas para a matriz U de palavras e k linhas para a transposição V; que corresponde aos documentos. Então, o que isso vai fazer é que a matriz U será uma espécie de matriz de transformação que mapeia palavras para o espaço do tópico, o espaço interno latente. E a matriz de transposição V, que mapeia o documento para esse mesmo espaço. Assim, você vê como você está essencialmente encontrando uma representação interna dos dados na qual você pode mapear as palavras e mapear os documentos para ela e, idealmente, essa representação será mais significativa, ou menos barulhenta que a original.

Se você refizer o produto dessas matrizes que você encontrou decompondo a matriz original, o que você encontrará é uma versão aproximada da matriz original. Por que nos importaríamos particularmente com uma versão aproximada? A ideia é que ao remover e eliminar os termos de ordem superior, basicamente significando as linhas e colunas que correspondem às dimensões latentes com os menores valores dos elementos sigma, você está essencialmente removendo muito do ruído na matriz, então você está limpando a estrutura de sua matriz de termos de documento. Claro, se k é igual a N, então você recupera a matriz original exatamente como era.

Para qualquer valor de k em vez de N, você acaba com uma aproximação da matriz, que será menos ruidosa. Então, como vimos, a análise semântica explícita se define cada palavra com base nos documentos para os quais ela está contribuindo. Aqui, por outro lado, **a análise semântica latente está tentando encontrar um espaço latente, uma representação latente, essencialmente uma representação intrínseca de cada palavra e documento**.

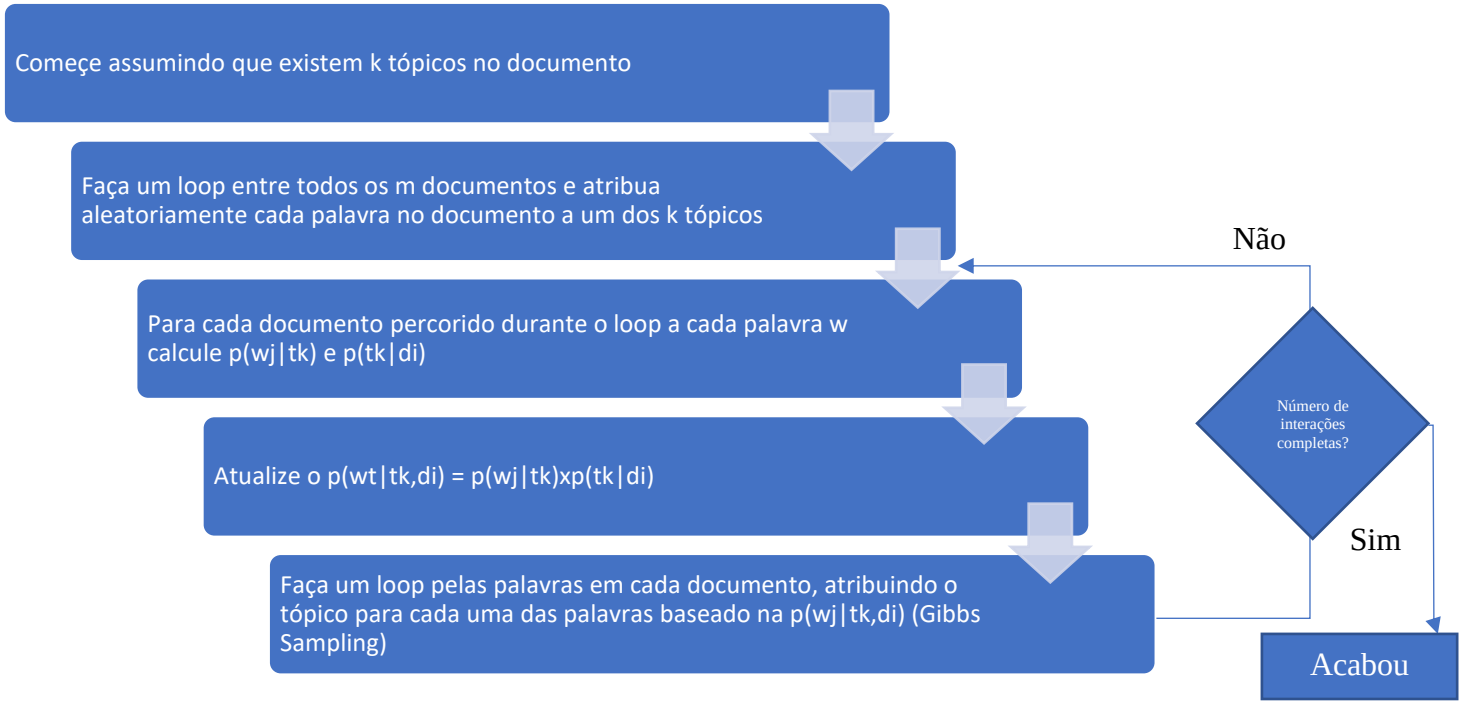
Essas soluções estão adotando abordagens diferentes para encontrar e escolher o mesmo resultado, que é identificar tópicos, ou pelo menos documentos relacionados uns aos outros. Estamos aproximando a matriz de termos de documentos por essa decomposição de valor único de k dimensional com cada uma das dimensões dessas dimensões correspondendo a uma dimensão latente, ou um tópico de discussão. A matriz U_k representa a distribuição de cada tópico entre as palavras, então você está mapeando palavras para tópicos, enquanto a matriz V_k analisa como cada documento é distribuído entre os tópicos.

Um conceito importante a se ter em mente, é que uma relação simbiótica entre palavras e documentos no sentido de que as palavras que fazem parte de um documento dependem de qual é o tópico desse documento. Mas se você conhece as palavras, também pode definir os tópicos, e se conhece os tópicos, pode adivinhar melhor quais são as palavras. Depois de termos essa decomposição, podemos mapear quaisquer novos documentos ou consultas que nos interessam nesse espaço singular fazendo essa transformação algébrica simples e isso nos dará essencialmente um vetor coluna que representa a consulta dentro do espaço latente na mesma representação que o espaço latente, então podemos simplesmente fazer a similaridade de cosseno usual para identificar outros documentos relacionados a este.

LDA - Latent Dirichlet Allocation

Nesta seção, veremos "Latent Dirichlet Allocation", que talvez seja **um dos algoritmos mais robustos para detecção de tópicos no corpus de documentos**. O "LDA", em oposição aos outros algoritmos que vimos até agora, é **um modelo generativo ou estatístico** que está essencialmente aproveitando essa relação simbiótica entre palavras, documentos e tópicos. Portanto, está essencialmente definindo que cada **documento é apenas uma mistura ou uma distribuição de probabilidade, sobre um pequeno número de tópicos**. E que cada palavra que faz parte do documento só está ali porque se refere a alguns dos tópicos aos quais o documento se refere.

Isso foi originalmente introduzido em PLN em 2003, mas originou-se da biologia e da psicologia. Então está vindo de uma área completamente diferente. Mas no final, com base em dois pressupostos fundamentais: documentos são misturas de tópicos e tópicos são misturas de palavras. Desta forma, se você sabe quais são as palavras, você pode descobrir quais são os tópicos. E se você sabe quais são os tópicos, você pode descobrir quais são as palavras. Você sempre tem essa relação. Portanto, a maneira como o algoritmo funciona é de forma iterativa, onde você começa adivinhando as distribuições de palavras em tópicos, de tópicos em documentos e, em seguida, redefinindo e refinando lentamente as relações com as distribuições de probabilidade.



Neste pequeno fluxograma temos um resumo do algoritmo. Você começa atribuindo os tópicos "K" aleatoriamente. Então, atribui-se alguma distribuição de probabilidade para cada documento de pertencer a um dos tópicos "K". Então você passa por todos os documentos e está atribuindo, ou verificando com que frequência cada palavra aparece para cada tópico. Essa ação está dando a você a probabilidade de essa palavra pertencer a esse tópico ou estar relacionada a esse tópico. Esse é o processo de configuração, onde você está inicializando todas as estruturas de dados que você precisa. Em seguida você simplesmente passa por cada documento, percorrendo cada uma das palavras, e computando novamente, a probabilidade condicional da palavra pertencer a esse tópico. E do tópico pertencer a esse documento.

Assim, $p(w_j|t_k)$ é essencialmente a probabilidade de todos os documentos atribuídos ao tópico t_k para uma dada palavra w_j. Então, basta contar quantos documentos que pertencem a esse tópico têm essa palavra. Enquanto o $p(t_k|d_i)$ é simplesmente a proporção de todas as palavras no documento d_i que são atribuídas ao tópico t_k, certo? Você está verificando como as palavras estão sendo influenciadas pelo tópico e como o tópico está sendo influenciado pelos documentos que são definidos pelas palavras. O próximo passo é você atualizar a probabilidade de que a palavra pertença a esse tópico naquele documento, tomando o produto dessas duas probabilidades condicionais que você acabou de calcular.

Você continua repetindo o processo, seja por **um número finito de etapas** ou até convergir, para que suas probabilidades não mudem mais. E o que isso faz essencialmente é, mais uma vez, identificar um conjunto de tópicos aos quais você pode atribuir as palavras e os documentos. Assim, você terá uma representação semelhante ao que você viu com outros algoritmos, mas, porque está fazendo isso de uma maneira diferente, os tópicos que encontraremos provavelmente serão diferentes. Além disso, um ponto que você deve ter notado em todos esses algoritmos até agora é que você precisa especificar quantos tópicos deseja. Na verdade, não há necessariamente uma boa regra que defina o que é um bom valor de "K". Essencialmente, o que você precisa fazer é tentar vários casos e ver como seus tópicos são bons. Existem algumas regras práticas que você pode usar em alguns casos. Mas em muitas situações, você está fazendo algum tipo de tentativa e erro, ou pelo menos avaliando alguma métrica sobre os tópicos encontrados em uma ampla gama de casos e, em seguida, escolhendo o melhor.

Non-Negative Matrix Factorization

O algoritmo final que veremos é a **fatoração matricial não negativa**. Este é outro algoritmo de fatoração e está meio relacionado com a análise semântica latente que já vimos. Mas agora, em vez de realizar uma decomposição de valor singular que essencialmente divide a matriz em duas partes, você pode **tentar iterativamente um caso diferente porque já tem a composição completa**. Aqui você tem que realmente refazer o processo todas as vezes. Mais uma vez, você deve especificar o valor de 'K', 'M' a priori desde o início. E então tentará encontrar uma representação de sua matriz original como o produto de duas matrizes. Se sua matriz original M era m linhas por n colunas, então m palavras por n documento, você está tentando encontrar duas outras matrizes W e H onde W será m por k. Portanto, palavras por tópicos e H são tópicos por documentos.

Novamente, é a mesma ideia que você está passando de palavras para tópicos e depois de tópicos para documentos. Então, há essa relação que discutimos nas últimas seções. Vamos refinar iterativamente nossa estimativa do que são os elementos de W e H, de modo que eles reconstruam M da melhor maneira possível. Claro que, como vimos anteriormente, dependendo do valor de K sua reconstrução será cada vez melhor. Portanto, quanto maior o valor de K, melhor é a reconstrução, mas, idealmente, você está tentando encontrar algum valor intermediário de K que resulte em **tópicos significativos**.

As colunas de W são os vetores de base para cada um dos tópicos. As colunas de H, são a contribuição de cada tópico para cada documento. Então, simplesmente olhando as colunas de W ordenadas por valores, você pode ver imediatamente quais são as palavras que mais contribuem para cada um dos tópicos. E essencialmente o que vamos fazer, vamos tentar iterativamente encontrar os valores de W e H que minimizem a diferença. Aqui estamos olhando apenas para a diferença ao quadrado, elemento por elemento da matriz original M e a matriz reconstruída.

Felizmente para nós, o Scikit-learn é o pacote padrão para aprendizado de máquina que possui uma implementação muito robusta, e podemos usá-lo imediatamente. Portanto, não precisamos implementar o algoritmo em si, mas é sempre importante entender exatamente como ele funciona.

Agora terminamos nossa revisão sobre algoritmos de modelagem de tópicos vamos passar a analisar conceitos de análise de sentimentos.

Análise de Sentimentos

Quantificando palavras e sentimentos

O próximo passo em nossa jornada de PLN é a Análise de Sentimentos. Aqui, vamos ver como podemos **quantificar quão positivas ou negativas palavras específicas são**, e usar isso para entender **se um determinado texto está dando a você uma crítica positiva ou negativa**. Se é uma mensagem positiva ou uma mensagem negativa. O primeiro passo é entender como podemos quantificar palavras e entender exatamente quais palavras são positivas e quais palavras são negativas. Todos nós usamos palavras para descrever nossos sentimentos e como estamos nos sentindo sobre coisas específicas e quais são nossos pensamentos. E algumas palavras têm conotações e significados óbvios. Por exemplo, amor, sim, amizade, bom, tendem a ser considerados positivos e transmitem sentimentos positivos, enquanto ódio, não, animosidade, mal e assim por diante, tendem a ser considerados negativos ou associados a sentimentos negativos.

Outras palavras como azul, talvez, indiferença, branco e assim por diante não têm fortes conotações positivas ou negativas, exceto talvez em contextos muito específicos. Portanto, uma maneira simples de quantificar se um texto está falando positivamente ou negativamente é simplesmente observar a proporção entre as palavras positivas e as palavras negativas. E, o que vamos fazer na abordagem simples mais básica é contar quantas palavras positivas temos. e quantas palavras negativas. Vamos definir o sentimento como sendo simplesmente a diferença entre o positivo e o negativo, normalizado pelo número total de palavras de sentimento. Isso naturalmente nos dará um índice que varia entre mais um, todo positivo e menos um negativo, totalmente negativo. Na maioria das vezes, haverá valores intermediários que nos dizem o quão predominante é um sentimento em relação ao outro. Essa é a abordagem mais simples possível.

Você pode estar se perguntando, como encontramos as listas de palavras ou léxicos no jargão técnico, que estão associadas a valências. Em particular, listas de palavras positivas e listas de palavras negativas. Existem vários locais em que você pode facilmente baixar e encontrar online esses léxicos: Opinion Lexicon, Opinion Finder, o Valence Aware Dictionary e o sEntiment Reasoner e o VADER.

Há também o produto comercial, que é o Linguistic Inquiry and Word Count, pronuncia-se LIWC. Todos eles são selecionados manualmente, como no caso do VADER e do LIWC, ou determinados e extraídos de pesquisas online ou outras ferramentas, como no caso do Opinion Lexicon e Opinion Finder. Eles vão te dar uma lista de palavras e te dizer se elas são positivas ou negativas. No caso mais simples possível, as pontuações serão apenas mais um ou menos um. Em versões mais sofisticadas, haverá uma pontuação numérica com diversos valores possíveis.

Em particular, existem alguns detalhes no léxico do VADER e na análise de sentimentos do VADER. O VADER tem mais de 7.000 tokens únicos, palavras únicas. Foi feito à mão por 10 anotadores diferentes, então 10 pessoas diferentes decidiram para cada palavra, quão positiva e quão negativa ela era. O léxico também inclui smileys porque foi desenvolvido para uso com mídias sociais e ferramentas online.

	word	mean	std	scores
5499	rebelling	-1.1	1.51327	[-3, -1, -2, -1, -2, -1, -1, -3, 2, 1]
2187	disliked	-1.7	0.64031	[-2, -3, -2, -1, -1, -1, -2, -2, -1, -2]
5289	pressurizers	-0.7	0.90000	[-1, 0, -2, -1, -2, -1, 0, 0, 1, -1]
5369	promiscuities	-0.8	1.32665	[-1, 0, -2, -2, -1, 2, -3, -1, 0, 0]
5738	rigorous	-1.1	1.51327	[-2, -3, 1, 0, 1, 0, -1, -1, -3, -3]
2131	discard	-1.0	0.44721	[-1, -1, -1, -1, 0, -1, -1, -1, -2, -1]
1968	despair	-1.3	1.90000	[2, -1, -3, -1, -3, 1, -3, 1, -3, -3]
765	antagonized	-1.4	0.66332	[-2, -1, -2, -2, -1, 0, -2, -1, -1, -2]
1175	boycott	-1.3	0.45826	[-2, -1, -1, -1, -1, -2, -1, -1, -2, -1]
2085	dignify	1.8	0.74833	[1, 2, 2, 1, 3, 3, 1, 2, 2, 1]

Acima, mostramos apenas uma amostra aleatória desse conjunto de dados.Você tem para cada palavra, a pontuação média, o desvio padrão e a lista de pontuações que cada um dos 10 anotadores deu. Assim, o valor médio que temos é simplesmente a média dessas 10 pontuações. O desvio padrão é apenas o desvio padrão das pontuações padrão.

O NLTK tem suporte para VADER internamente, como parte de seu SentimentIntensityAnalyzer. O SentimentIntensityAnalyzer tem uma função chamada pontuação de polaridade que recebe strings como entradas. A maioria dos métodos e funções que vimos do NLTK esperamos tokens. Aqui, na verdade, está

esperando strings, strings brutas e outros tokens. E a pontuação de polaridade está realmente processando a string diretamente e analisando-a usando o léxico direto para fornecer uma pontuação ao longo de quatro dimensões: Negativo, neutro, positivo e compound (ou composto - uma combinação dos outros 3). Eles vão te dar uma pontuação em todas essas três dimensões. E isso é normalizado para que a soma total de positivo, negativo e neutro seja um. E então o composto, que é uma métrica interna que está dando uma combinação de todos os três que resume o quão positivo ou negativo todo o texto é. Portanto, não é apenas uma soma ou não apenas uma média em todos os três, na verdade está fazendo algo um pouco mais sofisticado. O composto variará entre mais um para um positivo e menos um para uma classificação negativa.

Negações e Modificadores

Como podemos usar léxicos para quantificar se um pedaço de texto é positivo ou negativo? Temos que levar nossa análise um passo adiante e considerar a possibilidade **de negações** e **modificadores**. Todos sabemos que o significado de uma frase não é simplesmente a soma dos significados de cada palavra e que **algumas palavras são capazes de modificar outras**. Portanto, não podemos considerar apenas palavras individuais. Por exemplo, **não bonita**, é muito diferente de bonita. Quando ouvimos o "não", palavra que vem antes da palavra bonita está na verdade modificando o significado do n-gram, essencialmente do bigrama neste caso particular.

Sendo assim, uma maneira óbvia de avançar seria apenas pegar n-gramas diretamente e, em seguida, medir o sentimento de cada n-grama. Como vimos, quando analisamos explicitamente os n-grams, isso pode resultar em um número muito grande de novos tokens e novos n-grams a serem considerados, o que pode ser problemático e pode levar a limitações de memória, desempenho, mutações e assim por diante.

Logo, uma abordagem intermediária de algo mais prático, é simplesmente considerar o efeito e quantificar, modificar palavras como não muito, pouco, variar e assim por diante. Portanto, também podemos identificá-las automaticamente, fazendo ações de marcação de fala, ou podemos apenas ter uma lista de modificadores e incluí-los em nossa análise também. Na verdade, é possível com um simples ajuste, identificar automaticamente quando você tem esses n-grams modificadores e, essencialmente, calcular uma pontuação para cada um dos n-grams que temos, que são detectados à medida que você avança de forma dinâmica pelo texto.

Para dar um exemplo, aqui ao lado temos algumas palavras modificadoras possíveis. Então não (not), obviamente tem uma pontuação de menos um. que simplesmente inverte o significado da palavra que o segue. Muito (very), é um intensificador, então está aumentando a pontuação, então aqui estamos dando a pontuação do modificador de 1,5. Um pouco (somewhat), é um pouco menos intenso, por isso temos um valor menor de 1,2. Bonito (pretty), no sentido de muito bom (pretty good), ou muito ruim (pretty bad), por exemplo, seria um intensificador de digamos 1,5. Claro, esses números são apenas para fins ilustrativos. Você deve adaptá-los para sua aplicação específica. Aqui o truque é, nós só temos que verificar enquanto estamos varrendo nossos textos quais são os nossos tokens. Se o token que temos é um token modificador e está sendo seguido por uma de nossas palavras com valência, então, precisamos a ajustar sua pontuação, e é claro que devemos ter cuidado pois a mesma palavra pode ser um modificador ou uma palavra de valência por si só.

Essencialmente, estamos mantendo dois dicionários ou listas de palavras separadas, uma para modificadores e outra para valência. Assim, os dicionários estão mapeando os modificadores e a valência para as respectivas pontuações. Para cada palavra, verificamos se é um modificador, e depois verificamos a valência dando

prioridade aos modificadores. Isso nos ajudará a identificar os n-gramas. Se for um modificador, a próxima palavra pode ser, uma palavra de valência, então estamos anexando-a e adicionando-a ao n-gram atual. E no final, estamos apenas computando a pontuação para esse n-gram específico, como se fosse um token próprio com sua própria pontuação. Apesar da simplicidade, essa abordagem é bastante robusta. Ele pode lidar com sequências bastante longas de modificadores, duplas negativas e assim por diante, e ainda fornecer um resultado bastante razoável no final.

Abordagem baseada em Corpus

Até agora, todas as abordagens que consideramos se basearam **em listas de palavras selecionadas à mão com pontuações associadas**. Obviamente, essa não é uma abordagem que pode ser dimensionada muito bem para muitos aplicativos. É difícil encontrar, por exemplo, falantes de línguas minoritárias que estejam dispostos ou sejam capazes de selecionar essas palavras de sentimentos, e que sejam capazes de entender o suficiente da nuance da língua para fornecer pontuações adequadas para essas palavras.

Assim, o advento da internet e grandes conjuntos de dados online, propiciou o surgimento de um conjunto diferente de técnicas que dependem desses grandes conjuntos de dados online e abordagens de aprendizado de máquina para gerar esses léxicos automaticamente. Sem dúvida você já encontrou, em sites de compras online, análises de produtos que normalmente estão associadas a eles. Algum tipo de pontuação, muitas vezes de uma a cinco estrelas, às vezes apenas uma pontuação boa ou ruim. O objetivo da abordagem é aproveitar esses grandes dados corporativos de análises de produtos, mas também outros tipos de dados com algum tipo de pontuação associada a eles e treinar modelos de aprendizado de máquina visando prever qual será a pontuação basear-se apenas nas palavras da revisão.

Portanto, essas são técnicas de aprendizado de máquina supervisionadas um tanto sofisticadas que, quando você as treina para tentar prever esse núcleo da revisão com base apenas nas palavras que fazem parte da revisão, como consequência indireta, você está essencialmente descobrindo quanto cada palavra está contribuindo para a pontuação final. Então, quanto das palavras, ou quanto cada palavra é positiva ou quanto é negativa. É claro que você também pode identificar modificadores usando partes da marcação de fala, o que o ajudará a melhorar ainda mais seus algoritmos de aprendizado de máquina. E esse novo conjunto de técnicas tem um conjunto totalmente diferente de prós e contras que você deve ter em mente.

Como vimos, os léxicos com curadoria manual são tipicamente subjetivos, certo? Você está dependendo da opinião de seus anotadores, eles são potencialmente incompletos. Talvez você não consiga encontrar fatores suficientes para fazer um bom trabalho. Talvez você não consiga encontrar os anotadores em um idioma específico e assim por diante. Os léxicos que você pode gerar automaticamente a partir de grandes empresas usando algum tipo de abordagem de aprendizado de máquina consegue abranger uma gama mais ampla de idiomas e assuntos.

Você não depende de realmente identificar anotadores e, normalmente, até pagar anotadores para fazer esse trabalho, é algo que você pode fazer quase sozinho, simplesmente usando processos automatizados. É claro que a curadoria manual tem vantagem quando você está falando de conjuntos de dados menores, onde a entrada subjetiva de especialistas no assunto é mais importante, enquanto a geração automática geralmente depende de **conjuntos de dados muito grandes**, que têm seus próprios vieses devido à maneira como foram construídos.

Um exemplo óbvio é que a maioria desses conjuntos de dados usados para treinar **algoritmos de análise de sentimentos** são originários de análises de produtos, certo? Então eles são tipicamente mais comerciais e são

mais propensos a discutir objetos do que a discutir situações ou pessoas. Portanto, dependendo do objetivo da sua aplicação, eles podem não ser os léxicos úteis. No final, seus resultados serão **tão bons quanto seu léxico**. Seus resultados são sempre tão bons quanto seus dados e sua compreensão de seus vieses e limitações da abordagem que você está usando. Portanto, certifique-se de entender todos os preconceitos e todas as limitações das possibilidades que você tem em seu kit de ferramentas.

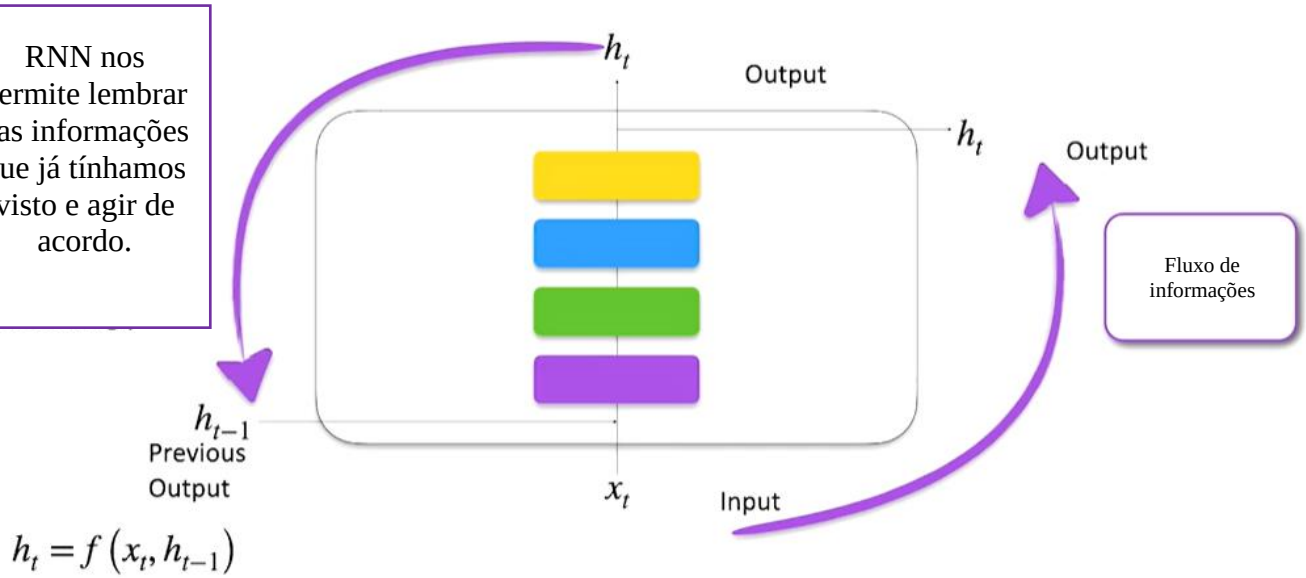
E aqui terminamos a parte teórica da nossa revisão!! Vamos passar agora para a nossa aposta estratégica!

APOSTA ESTRATÉGICA

A ideia desta seção é apresentar os pontos do conteúdo que mais possuem chances de serem cobrados em prova, considerando o histórico de questões da banca em provas de nível semelhante à nossa, bem como as inovações no conteúdo, na legislação e nos entendimentos doutrinários e jurisprudenciais[1].

Redes Neurais Recorrentes

RNN nos permite lembrar das informações que já tínhamos visto e agir de acordo.



Na aposta estratégica, veremos modelos de sequência ou modelos de sequência a sequência (sequence-to-sequence), como também são conhecidos, usando diferentes tipos de redes neurais recorrentes (RNN). As RNN fecharam o ciclo de fluxo de informações, alimentando a saída passada junto com a entrada atual. Agora a saída que estamos representando como h_t será uma função não apenas da entrada atual $f(x_t)$, mas também da saída mais recente $h_{(t-1)}$.

Assim como a saída atual depende diretamente da saída anterior. Portanto, a h de T depende da h de t-1, a h de t-1 depende de h de t-2 ... , o que significa que cada saída dependerá implicitamente de todas as saídas

[1] Vale deixar claro que nem sempre será possível realizar uma aposta estratégica para um determinado assunto, considerando que às vezes não é viável identificar os pontos mais prováveis de serem cobrados a partir de critérios objetivos ou minimamente razoáveis.

anteriores. Uma das consequências diretas disso é que a sequência na qual você treina a rede e, literalmente, a sequência na qual você forneceu uma entrada à rede, terá que os exemplos afetarão diretamente as saídas que você obtém. Você está alimentando a sequência de entradas e vai gerar uma sequência de saída. Portanto, elas também são referidas como **modelos de sequência a sequência**, a rede neural recorrente mais simples foi introduzida há vários anos. E, tudo o que ela faz é, em vez de simplesmente alimentar a entrada, então o $f(x_t)$ concatena a entrada atual com a saída imediatamente anterior e alimenta ambas na função de ativação.

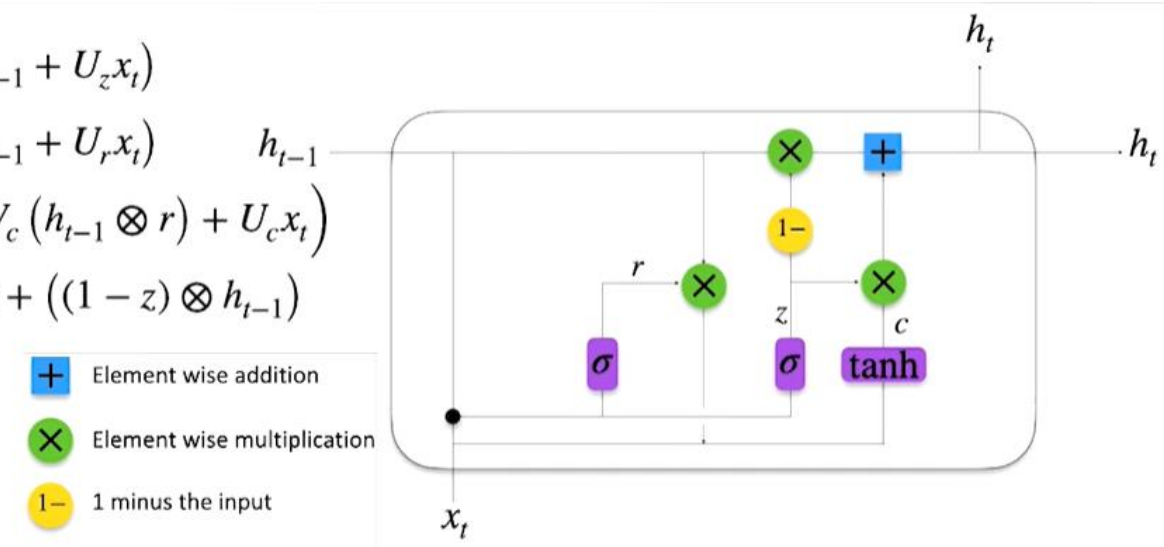
Gated Recurrent Unit

As redes neurais recorrentes são uma abordagem relativamente simples para tentar resolver esse tipo de problema de modelagem de sequências. Você tem algum tipo de abordagem que mapeia uma sequência de entrada para uma sequência de saída. Uma opção mais sofisticada é chamada de Gated Recurrent Unit. Foi introduzido em 2014 por Cho pesquisador da NYU. Isso foi utilizado para resolver um dos problemas que assola modelos de aprendizado: o problema do gradiente de fuga.

Pense assim: se você tiver muitos parâmetros para otimizar e aprender em seu sistema, às vezes você pode chegar a um estado em que partes de sua rede atingirão um estado um pouco estável. Portanto, gradiente de fuga significa um lugar plano. Assim, seu modelo irá convergir para uma solução menos que ótima. Quanto mais parâmetros você tiver, quanto mais complexo for o seu modelo, mais provável será que isso aconteça. Assim, esta arquitetura específica foi desenvolvida para ajudar a evitar este problema minimizando o conjunto de parâmetros, minimizando essencialmente a complexidade interna da arquitetura.

Embora ainda tenha um desempenho bom o suficiente e tenha um desempenho melhor do que apenas a abordagem tradicional para redes neurais recorrentes. Em particular, o GRU é usado por ser um pouco mais simples e ter conseguido um desempenho quase tão bom quanto a LSTMs, mas pode realmente obter melhor desempenho no caso de conjuntos de dados menores, quando você tem menos dados para treinar seu sistema.

$$z = \sigma\left(W_z h_{t-1} + U_z x_t\right)$$
$$r = \sigma\left(W_r h_{t-1} + U_r x_t\right)$$
$$c = \tanh\left(W_c \left(h_{t-1} \otimes r\right) + U_c x_t\right)$$
$$h_t = (z \otimes c) + \left((1 - z) \otimes h_{t-1}\right)$$



Vamos ver como é o GRU e olhar dentro da unidade recorrente do gateway e ver como ela funciona com mais detalhes. Se você olhar com um pouco de cuidado, poderá ver como a arquitetura original da rede neural

recorrente ainda está lá. Você tem a tangente hiperbólica gerando a saída, com base na saída anterior e na entrada atual. Mas agora, você tem uma série de outras portas ou portas lógicas. Isso está ajudando a controlar exatamente como essa saída é gerada, quanto da saída anterior deve contribuir para a saída final e quanto você deve se lembrar dela.

A primeira porta é conhecida como **porta de atualização (update)** e essencialmente determina quanto do estado anterior deve ser mantido. Ela está usando uma função de ativação sigmóide simples e dá a você um número entre zero e um que você multiplicará a saída gerada a partir dessa entrada atual modulando-a. Portanto, se o valor for zero, você o removerá completamente. Se for um, você o está mantendo completamente.

A segunda porta é a **de redefinição (reset)**, que está desempenhando um papel semelhante. Mas agora está tentando determinar quanto da saída anterior deve ser removida, ao calcular a saída atual. Por fim, temos uma **porta de memória (memory)** atual, que está aqui está essencialmente pegando os resultados da porta de saída que acabamos de ver e combinando-o com a entrada atual e alimentando isso na tangente hiperbólica. Na rede neural recorrente regular que gerará o candidato de saída, digamos para esta iteração específica, a **porta final (**a porta de saída) calculará a saída final combinando todas as partes. Ele está combinando cada parte da saída anterior. Portanto, h_{t-1} e uma parte da saída atual gerada pela tangente hiperbólica nesta iteração e combinando-as.

O que está calculando, é a média dos dois com o fator determinado pelas portas originais que vimos pela função de ativação sigmóide original. Um componente está sendo multiplicado diretamente por esse sigmóide. O outro componente está sendo multiplicado por um menos o Sigmóide. Então, o Sigmoid está realmente agindo como o botão que controla quanto obtemos da saída anterior versus quanto obtemos da atual.

Para usar a GRU em sua arquitetura de rede, a única alteração que você precisa fazer na forma como está definindo a rede é simplesmente substituir uma célula GRU, colocando-a no lugar de uma célula RNN. Elas são projetadas para funcionar exatamente da mesma maneira e para serem completamente intercambiáveis. Essa é uma das vantagens de usar um pacote como Keras. Nele você transforma cada um desses componentes em pequenos blocos de construção, como Legos que você pode combinar e montar e usar para experimentar as diferentes arquiteturas e ver qual funciona melhor para o seu caso específico.

Word2vec Embeddings

Vamos explorar com mais detalhes algumas aplicações mais avançadas e práticas do processamento de linguagem natural. Começando com os **embeddings do Word2Vec,** que foram projetados para tirar proveito do que é conhecido como a **hipótese de distribuição em linguística**. A ideia fundamental é que palavras com significados semelhantes tendem a ocorrer em contextos semelhantes. Então, temos aqui alguns exemplos em que se eu lhe der duas frases com duas palavras faltando no início, duas no final e casa ou carro, na posição do meio, você pode chegar a frases que fazem sentido e que são significativos.

_____ _____ carro _____ _____

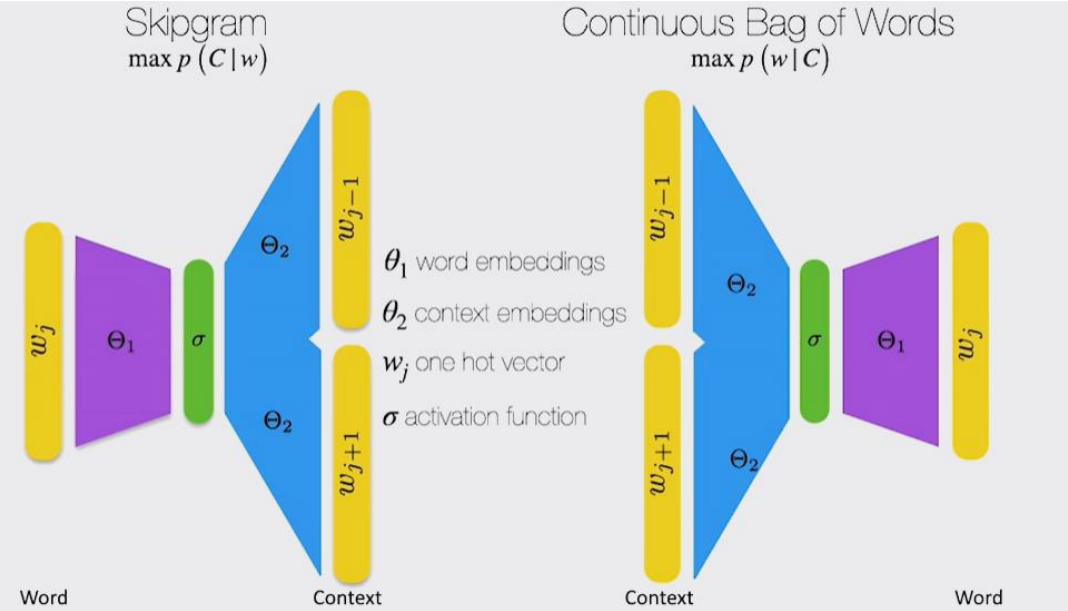
_____ _____ casa _____ _____

Por outro lado, se eu lhe der uma frase de cinco palavras com uma palavra faltando no meio, você também pode facilmente encontrar uma palavra que se encaixe ali e que seja significativa e gramaticalmente correta. No primeiro exemplo, o que temos é a palavra específica que você está tentando encontrar um contexto que maximize a probabilidade, que é o mais provável. Enquanto, no segundo caso, você recebe o contexto e pergunta qual é a palavra ou conjunto de palavras mais provável. Você pode pensar nisso como duas abordagens complementares, onde em um caso, você recebe a palavra e tenta adivinhar o contexto em que ela aparece, enquanto no outro caso, você recebe o contexto e você pedem para adivinhar qual é a palavra que está faltando.

O vermelho _____ é lindo

O azul _____ é velho

O Word2Vec operacionaliza essas definições com duas versões de uma rede de ajuste simples, que estamos mostrando aqui.



No lado esquerdo, temos o Skipgram, que é quando você recebe a palavra e tenta adivinhar o contexto em que a palavra aparece. E aqui o contexto é definido por **uma janela de palavras** antes e depois da palavra central, a palavra que você está usando para definir qual é o contexto. No exemplo simples, estamos apenas mostrando uma palavra antes e uma palavra depois. Então, o que você recebe é a palavra W_J e você deve adivinhar o que pode ser a palavra J mais um e a palavra J menos um. Portanto, a palavra à direita e a palavra à esquerda. Isso é feito com um ajuste simples para nossa rede com apenas uma única camada oculta com a função de ativação linear.

Apenas dois conjuntos de matrizes de peso, que estou chamando aqui de θ_1 e θ_2. Assim, θ_1 acabará se tornando a incorporação de palavras para as palavras de entrada, a θ_2 se tornará a incorporação de palavras para as palavras de contexto. Tanto as palavras de entrada quanto as palavras de contexto são representadas como vetores codificados one-hot, exatamente como vimos no início da nossa revisão. E como a função de ativação para a camada oculta é apenas a função de ativação linear, o que essa arquitetura está fazendo é simplesmente

pegar o produto escalar da incorporação de palavras vezes a incorporação de contexto. E está tentando encontrar o conjunto de pesos que maximiza esse produto escalar, de modo que o produto escalar seja máximo quando estiver correspondendo à palavra de contexto correta.

Conceitualmente, o que estamos fazendo é dado uma palavra, tente adivinhar várias palavras. Na prática, para cada palavra de entrada, temos uma distribuição de todas as palavras possíveis e estamos escolhendo o Xmax como sendo as palavras de contexto.

No lado direito, a situação é a mesma, mas invertida. Temos o que é conhecido como Bolsa Contínua de Palavras (CBOW), em oposição à versão anterior que é conhecida como Skipgram. No Saco de Palavras Contínuo, você recebe as palavras de contexto e é solicitado a adivinhar qual é a palavra que está faltando. Portanto, a sugestão é perfeitamente simétrica. Agora, θ_1 está no lado da entrada e θ_2 está no lado de saída. A função de ativação ainda existe e você ainda está essencialmente tentando maximizar o produto escalar desses dois vetores. Os vetores de incorporação que você obtém de um lado ou de outro, de uma abordagem ou de outra, serão ligeiramente diferentes e serão ligeiramente otimizados para objetivos diferentes.

Esta é a versão muito genérica, muito simples e simples do Word2Vec. Na prática, existem diversas variações que tentam melhorar a eficiência do algoritmo. Como vimos, estamos usando vetores codificados one-hot, o que significa que temos um vetor muito grande com muitos zeros e apenas um elemento que é um. E na camada final, estamos usando um Softmax para selecionar qual é a palavra mais provável nesse longo vetor de quase zeros.

Uma das principais características do tipo de embeddings Word2Vec é que os vetores representam informações significativas, contedo informações sobre o significado da palavra. Isso pode ser visto facilmente quando projetamos esses vetores em duas dimensões, e descobrimos que palavras estão relacionadas umas às outras, que são usadas em contextos semelhantes, tendem a se agrupar.

GloVe

Agora eles têm uma compreensão mais profunda de onde fazer as incorporações. Vamos olhar com mais atenção para o GloVe, que são **vetores globais**, que foram desenvolvidos como **uma alternativa ao word2vec por Stanford**, pelo grupo Stanford NLP em 2014, então no ano seguinte ao word2vec.

Um dos objetivos do GloVe era demonstrar que você não precisa usar explicitamente uma arquitetura de rede neural para desenvolver e também deixar bem claro e explícito que esses modelos estão usando e dependem muito de coocorrências. E ele faz isso essencialmente **computando os vetores**, a incorporação de vetores entre duas palavras é vista como o produto escalar das duas palavras, de modo que o resultado do produto escalar seja proporcional ao número de vezes que essas duas palavras ocorrem.

GloVe é essencialmente um modelo log-bilinear com um objetivo ponderado. A principal intuição subjacente ao modelo é a simples observação de que as razões das probabilidades de coocorrência palavra-palavra têm o potencial para codificar alguma forma de significado.

Imprima o capítulo Aposta Estratégica separadamente e dedique um tempo para absolver tudo o que está destacado nessas duas páginas. Caso tenha alguma dúvida, volte ao Roteiro de Revisão e Pontos do Assunto que Merecem Destaque. Se ainda assim restar alguma dúvida, não hesite em me perguntar no fórum.

QUESTÕES ESTRATÉGICAS

Nesta seção, apresentamos e comentamos uma amostra de questões objetivas selecionadas estrategicamente: são questões com nível de dificuldade semelhante ao que você deve esperar para a sua prova e que, em conjunto, abordam os principais pontos do assunto.

A ideia, aqui, não é que você fixe o conteúdo por meio de uma bateria extensa de questões, mas que você faça uma boa revisão global do assunto a partir de, relativamente, poucas questões.

1.

Uma organização está implementando um sistema de busca de informações interno, e a equipe de desenvolvimento resolveu avaliar diferentes modelos de linguagem vetoriais que ajudariam a conectar melhor documentos e consultas em departamentos que usam terminologias distintas em áreas de negócio que se sobrepõem. Um dos analistas ressaltou que seria interessante guardar os vetores de todo o vocabulário do modelo em um cache, de forma a aumentar a eficiência de acesso e reduzir certos custos de implantação.

Das alternativas abaixo, aquela que lista apenas os modelos compatíveis com essa estratégia de caching é:

A TF-IDF, BERT;

B Word2Vec, BERT, GPT-2; ,

C GloVe, GPT-2;

D Word2Vec, GloVe;

E GPT-2, BERT.

Comentário: Vejamos as definições de cada um dos termos apresentados nas alternativas da questão:

TF-IDF (*Term Frequency – Inverse Document Frequency*) → é uma ferramenta utilizada para análise de texto e com isso saber a frequência e o peso das palavras dispostas em documentos dentro de um *corpus*. O objetivo é medir a importância de uma palavra dentre de um documento considerando a sua presença no próprio documento analisado e sua ausência nos demais documentos.

BERT (*Bidirectional Encoder Representations from Transformers*) → é uma ferramenta criada pelo Google para permitir as máquinas entenderem, ou melhor, compreenderem a linguagem humana que foi digitada pelos usuários nos buscadores. Isso muda a maneira como nós, humanos, fazemos a busca, tornando essa tarefa mais amigável de ser feita.

Word2Vec → é uma ferramenta de aprendizado não supervisionado para criação de representação vetorial, a qual está presente em textos que utilizem linguagem natural. Essa ferramenta trabalha em pares.

GPT-2 (*Generative Pre-Training Transformer 2*) → é uma ferramenta de inteligência artificial utilizada, primordialmente, para geração de textos.

GloVe (*Global Vectors for Word Representation*) → é uma ferramenta de aprendizado não supervisionado que é utilizado para representações vetoriais de palavras. Sua missão é, após a análise de um corpus, unir palavras semelhantes.

Desta forma, podemos marcar nossa resposta na alternativa D

Gabarito: alternativa D.

2.

Durante a elaboração de um sistema de busca de informações biomédicas, foi construído um modelo de linguagem vetorial não contextual para estimar relações de similaridade semântica necessárias para comparação entre queries e documentos. Entretanto, verificou-se nos testes iniciais que o desempenho do modelo ficou insatisfatório, devido a muitos termos técnicos presentes nos documentos testados, que não haviam sido incorporados ao modelo.

Para aliviar esse problema, uma tarefa de processamento do texto e seu estágio correspondente no processamento de linguagem natural que poderiam ser aplicados na construção do modelo são, respectivamente:

A Word embedding; Análise léxic;

B Lematização; Análise sintática;

C Decomposição morfológica; Análise léxica;

D Word embedding; Análise semântica;

E Decomposição morfológica; Análise sintática.

Comentário: Primeiramente precisamos conhecer os estágios de PLN: Análise Léxica, Sintática e Semântica.

A Análise Léxica é responsável por manipular o léxico, que é composto por palavras que armazenam os seus significados e categorias lexicais. Essa etapa é responsável por fazer **a verificação ortográfica, podendo ser classificada como: substantivo, verbo, advérbio, pronome, numeral, preposição, conjunção, interjeição, artigo e adjetivo.**

Na análise sintática testa se os sintagmas foram postos na sequência correta, ou seja, se por exemplo dois substantivos podem se seguir ou não.

A análise semântica precisa entrar no mérito de cada combinação de palavras para entender o contexto e sentido das frases.

Com isso, podemos saber que a Decomposição Morfológica faz parte da Análise Léxica, já que a morfologia é justamente a identificação de substantivos, adjetivos, entre outros. Assim, podemos marcar nossa resposta na alternativa C.

Gabarito: alternativa C.

3.

A atividade de classificação de documentos envolve um grande número de tarefas de processamento de linguagem natural, o que pode levar a dúvidas quanto a sua aplicação.

A alternativa que contém apenas tarefas que sejam exemplos de classificação de documentos é:

A análise de sentimento, tokenização;

B POS-tagging, reconhecimento de entidades nomeadas;

C filtragem de SPAM, análise de sentimento;

D análise sintática, POS-tagging;

E filtragem de stopwords, reconhecimento de linguagem.

Comentário: Essa questão apresenta um conjunto de conceitos que podem ser divididos em 3 grandes grupos: tarefas de pré-processamento, estágios de PLN e exemplos de classificação de documentos.

Como tarefas de pré-processamento temos: Tokenização, PoS-Tagging e Filtragem de Stopwords, Reconhecimento de entidades nomeadas.

Como estágios de PLN temos a análise sintática

Como exemplos de classificação de texto podemos listar a análise de sentimentos e a filtragem de Spam.

Assim, a nossa resposta encontra-se na alternativa C.

Gabarito: alternativa C.

4.

Considere os documentos A e B a seguir.

A = “Há pessoas que choram por saber que as rosas têm espinho”

B = “Há outras que sorriem por saber que os espinhos têm rosas”

A submatriz da matriz de TF-IDF desses dois documentos correspondente aos termos “Rosas”, “Choram” e “Sorriem”, nessa ordem, é:

(A) $\begin{bmatrix} 0 & 0 & \frac{1}{11} \\ 0 & \frac{\log 2}{11} & 0 \end{bmatrix}$;

(B) $\begin{bmatrix} \frac{1}{11} & \frac{1}{11} & 0 \\ \frac{1}{11} & 0 & \frac{1}{11} \end{bmatrix}$;

(C) $\begin{bmatrix} 0 & \frac{\log 2}{11} & 0 \\ 0 & 0 & \frac{\log 2}{11} \end{bmatrix}$;

(D) $\begin{bmatrix} 0 & 0 & \frac{\log 2}{11} \\ 0 & \frac{1}{11} & 0 \end{bmatrix}$;

(E) $\begin{bmatrix} \frac{1}{11} & \frac{\log 2}{11} & 0 \\ \frac{1}{11} & 0 & \frac{\log 2}{11} \end{bmatrix}$.

Comentário: Ao observar os documentos acima, vemos que a palavra "**rosas"**aparece nos dois documentos, portanto é uma palavra considerada comum nos documentos acima. Como estamos diante de uma palavra incomum, sua relevância na submatriz é 0.

A palavra "**choram"** aparece apenas no documento A, portanto é considerada uma palavra relevante, "inédita", logo, importante. No documento A, ela tem uma relevância e o seu valor na submatriz será logarítmico, ***log2/11***. Já no documento B essa palavra não aparece, portanto para esse documento, ela não tem relevância e seu valor é 0.

A palavra "**sorriem"** aparece apenas no documento B, portanto é considerada uma palavra relevante, "inédita", logo, importante. No documento B, ela tem uma relevância e o seu valor na submatriz será logarítmico, ***log2/11***. Já no documento A essa palavra não aparece, portanto para esse documento, ela não tem relevância e seu valor é 0.

Apenas com essas três palavras, a submatriz é a seguinte:

$$\begin{bmatrix} 0 & {log2}/{11} & 0 \\ 0 & 0 & {log2}/{11} \end{bmatrix}$$

Essa submatriz é encontrada, apenas, na letra C, sendo esse o nosso gabarito.

Gabarito: alternativa C.

5. Ano: 2023 Banca: TRC Assunto: Processamento de Linguagem Natural

Sobre o processamento de linguagem natural (PLN) assinale a alternativa correta.

a) O processamento é sempre feito sobre uma linguagem formal, como o português.

b) A questão da ambiguidade semântica não deve ser levada em consideração visto que é possível encontrar o significado das frases através do contexto.

c) O PLN pode ser dividido em três grandes ações o processamento, a compreensão do texto e a geração de resposta.

d) A sumarização de texto abstrativa é mais fácil de ser aplicada sobre um texto do que a extrativa.

e) Inteligência artificial é um subconjunto do processamento de linguagem natural.

Comentário: Vamos comentar cada uma das alternativas:

a) (Errada). Português é um exemplo de linguagem natural.

b) (Errada). A ambiguidade semântica deve ser levada em consideração.

c) (Certo). Essas são as 3 grandes ações do PLN: Processamento, compreensão e geração de resposta.

d) (Errada). A sumarização abstrativa é mais complexa do que a extrativa.

e) (Errada). O processamento de linguagem natural é um subconjunto da inteligência artificial.

Gabarito: alternativa C.

6. Ano: 2023 Banca: TRC Assunto: Processamento de Linguagem Natural

Sobre os conceitos de semântica vetorial assinale a alternativa correta.

a) Um embedding é uma representação geométrica com valor real de algo que geralmente é contínuo.

b) O uso de embeddings é importante por algumas técnicas só entendem valor numéricos, por exemplo, as redes neurais.

c) Stemming é um processo em que o texto de entrada é dividido em unidades menores.

d) Lematização é um processo para identificar radicais de palavras.

e) Tokenização é a forma original de uma palavra que você costuma encontrar como palavra-chave em um dicionário

Comentário: Vamos comentar cada uma das alternativas a seguir:

a) (Errada) Um embedding é uma **representação vetorial** com valor real de algo que geralmente é discreto.

b) (Certo) Exatamente, alguns algortimos, em especial os de redes neurais só trabalham com valores numéricos.

c) (Errado) Stemming é um processo para identificar radicais de palavras.

d) (Errado) Um lema é a forma original de uma palavra que você costuma encontrar como palavra-chave em um dicionário. É também a forma básica da palavra antes da inflexão. A lematização é usada para reduzir o conjunto de palavras.

e) (Errado) Tokenização é um processo em que o texto de entrada é dividido em unidades menores. Existem dois tipos de tokenização: (1) A tokenização de palavras que divide uma frase em tokens (equivalente a palavras e pontuação) (2) A tokenização de frases que divide um trecho de texto incluindo possivelmente mais de uma frase em frases individuais.

Logo, nossa resposta encontra-se na alternativa B.

Gabarito: alternativa B.

---

7. Ano: 2023 Banca: TRC Assunto: Processamento de Linguagem Natural

Uma das formas de reduzir a dimensionalidade em problemas de processamento de linguagem natural é:

a) Usando métodos lineares que procuram por combinações mais relevantes de recursos ou componentes, como o PCA.

b) Usando KNN para procurar os vizinhos mais próximos e agrupá-los em conjuntos semelhantes.

c) Usando dados não estruturados para facilitar a transferência deles entre os algoritmos

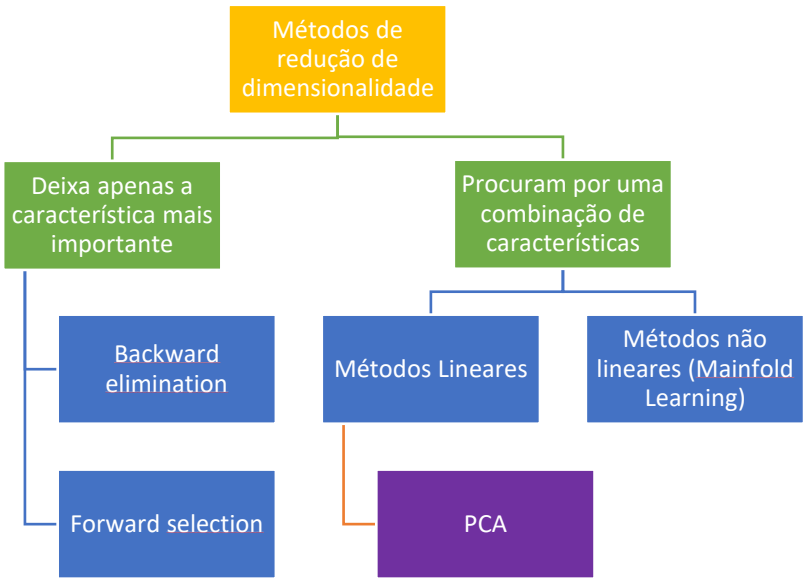
d) Usando dados abertos pois o cidadão pode contribuir com a redução de dimensionalidade dos dados garantindo sua qualidade.

e) Usando N-grams para representar uma sequência contígua de uma ou mais ocorrências de unidades linguísticas

Comentário: A redução da dimensionalidade pode ser feita de duas maneiras diferentes:

- Mantendo apenas as variáveis mais relevantes do conjunto de dados original (esta técnica é chamada de seleção de recursos)
- Ao encontrar um conjunto menor de novas variáveis, cada uma sendo uma combinação das variáveis de entrada, contendo basicamente as mesmas informações que as variáveis de entrada (esta técnica é chamada de redução de dimensionalidade)

Agora, os métodos ou formas usadas para redução de dimensionalidade podem se utilizar de métodos lineares ou não lineares. Além disso, o resultado obtido pode conter um ou vários recursos dos dados. Essa classificação das formas de redução de dimensionalidade pode ser vista na figura a seguir:



Assim, temos nosso gabarito na alternativa A.

Gabarito: alternativa A.

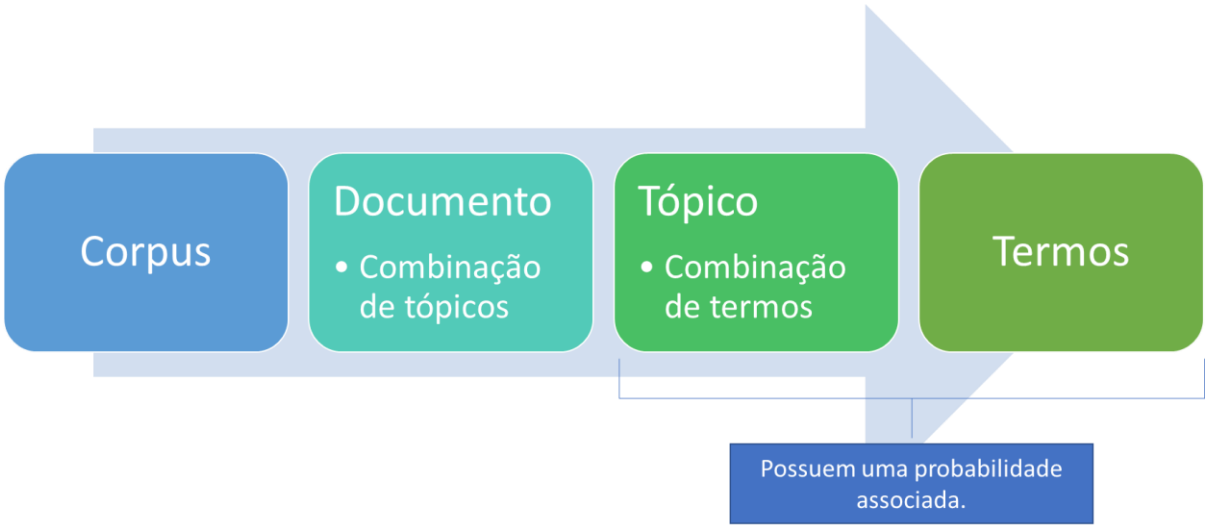
---

8. Ano: 2023 Banca: TRC Assunto: Processamento de Linguagem Natural

Qual o termo abaixo pode ser definido como uma coleção de documentos associados a um contexto específico:

a) Tokens
b) Tópicos
c) Corpora
d) Corpus
e) Biblioteca

Comentário:



Corpus é uma **coleção de documentos** de texto sobre os quais aplicaríamos rotinas de mineração de texto ou de processamento de linguagem natural para derivar inferências. Um conjunto de corpus é deonomidando **Corpora**. Logo, temos a nossa resposta na alternativa D.

Gabarito: alternativa D.

---

9. Ano: 2023 Banca: TRC Assunto: Processamento de Linguagem Natural

Sobre a classificação de texto, assinale a alternativa correta.

a) A classificação de texto é bem diferente da classificação utilizada na mineração de dados, inclusive em relação ao processo.

b) Um dos tipos de classificação de texto é a classificação binária que vai definir um conjunto de três ou mais classes.

c) No processo de classificação, podemos dividir nossa base em 2 ou 3 partes. Quando dividimos em 3 grupos temos partes específicas para treinamento, teste e validação.

d) Não é necessário transforma o texto em um vetor, pois os algoritmos mais modernos já são capazes de trabalhar com dados textuais diretamente.

e) A avaliação do modelo deve ser feita depois da publicação do mesmo para a comunidade de usuários.

Comentário: Vamos comentar cada uma das alternativas acima:

a) (Errada) A classificação de texto segue a mesmas lógica da tarefa de classificação. O modelo deve ser treinado com dados rotulados e, em seguida, pode ser usado para prever algo sobre novos textos.

b) (Errada) A classificação binária vai dividir o conjunto de dados em duas classes.

c) (Certa) Na prática, na maioria dos casos dividimos nossos dados em treinamento e teste, mas, em algumas condições usamos um terceiro subconjunto dos dados chamado de validação. Vamos aproveitar a questão para rever a definição dos 3 conceitos:

- **Conjunto de treinamento**: Conjunto de exemplos utilizados para a aprendizagem, ou seja, para se adequar aos parâmetros do classificador.
- **Conjunto de validação**: um conjunto de exemplos usados para ajustar os parâmetros de um classificador, por exemplo, para escolher o número de unidades ocultas em uma rede neural.
- **Conjunto de testes**: um conjunto de exemplos usados apenas para avaliar o desempenho de um classificador totalmente especificado.

d) (Errada). Errado, algoritmos mais modernos, como BERT, fazem um embedding sobre os dados textuais para transformá-los em valores ou vetores numéricos.

e) (Errada) A avaliação do modelo deve ser feita antes de se pôr o modelo em produção.

Gabarito: alternativa C.

---

10. Ano: 2023 Banca: TRC Assunto: Processamento de Linguagem Natural

- Sobre análise de sentimento, avalie a afirmativa incorreta.

a) O nível de análise por dividido em nível de mensagem, nível de sentença ou nível de entidade/aspecto.

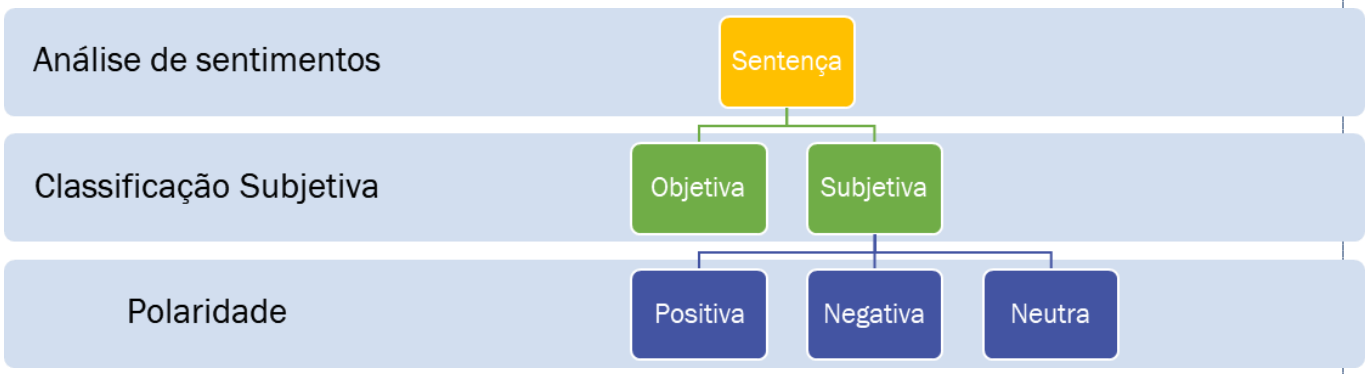
b) O objetivo da análise de sentimento é definir ferramentas automáticas capazes de extrair informações subjetivas de textos em linguagem natural, como opiniões e sentimentos

c) A análise de sentimento deve ser capaz de criar um conhecimento estruturado e acionável para ser usado por um sistema de apoio à decisão ou um tomador de decisão.

d) Podemos classificar uma opinião em regular ou comparativa.

e) A classificação de polaridade pode definir um texto como positivo, negativo ou neutro, sendo, portanto, uma classificação objetiva.

Comentário: A questão pede para analisarmos as alternativas e descobrimos a incorreta. O erro encontra-se na alternativa E. A classificação de polaridade é uma classificação subjetiva, conforme visto na figura abaixo.



As demais alternativas fornecem um resumo do assunto.

Gabarito: alternativa E.

11. Ano: 2023 Banca: TRC Assunto: Processamento de Linguagem Natural

PLN é uma área que se sobrepõe a outras. Surgiu em campos como inteligência artificial, linguística, linguagens formais e compiladores. Com o avanço das tecnologias de computação e o aumento da disponibilidade de dados, a forma como a linguagem natural está sendo processada mudou. Anteriormente, um sistema tradicional _____era usado para cálculos. Hoje, cálculos em linguagem natural são feitos usando _____ e técnicas de aprendizado profundo (deep learning).

Assinale a alternativa que preenche corretamente as lacunas acima:

a) Baseado em regras – inteligência artificial

b) Baseado em correlação – uma abordagem geométrica

c) Baseado em instinto – uma abordagem analítica baseada em dados

d) Baseado em regras – uma abordagem clerical

e) Baseado em correlação – uma abordagem matricial

Comentário: Essa questão tenta deixar explícito a evolução dos modelos baseados em regras, que eram excessivamente complexos de especificar e não conseguiam cobrir todos os casos, para os modelos baseados em inteligência artificial, por exemplo as redes neurais, que conseguem capturar nuances dos dados e executar tarefas como uma precisam maior. Os modelos de deep learning são uma evolução dos modelos que permitem capturar ainda mais detalhes das variações dos dados.

Assim, a nossa resposta encontra-se na alternativa A.

Gabarito: alternativa A.

---

12. Ano: 2023 Banca: TRC Assunto: Processamento de Linguagem Natural

A redução de dimensionalidade apresenta diversos benefícios. Dentre as alternativas a seguir, qual destas não apresenta um benefício da redução de dimensionalidade:

a) O espaço necessário para armazenar os dados é reduzido conforme o número de dimensões diminui.

b) Um número menor de dimensões leva a menos tempo de computação/treinamento.

c) Melhora a execução de alguns algoritmos.

d) Trata problemas de multicolinearidade, removendo recursos redundantes.

e) Ajuda a encontrar valores ausentes.

Comentário: Aqui estão alguns dos benefícios da aplicação de redução de dimensionalidade a um conjunto de dados:

- O espaço necessário para armazenar os dados é reduzido conforme o número de dimensões diminui
- Um número menor de dimensões leva a menos tempo de computação/treinamento

- Alguns algoritmos não funcionam bem quando temos grandes dimensões. Portanto, a redução dessas dimensões precisa acontecer para que o algoritmo seja útil.
- Ela cuida da multicolinearidade removendo recursos redundantes. Por exemplo, você tem duas variáveis - 'tempo gasto na esteira em minutos' e 'calorias queimadas'. Essas variáveis estão altamente correlacionadas, pois quanto mais tempo você gasta correndo em uma esteira, mais calorias você queima. Portanto, não há nenhuma justificativa para armazenar ambos, pois apenas um deles faz o que você precisa
- Ajuda na visualização de dados. É muito difícil visualizar dados em dimensões mais altas, portanto, reduzir nosso espaço para 2D ou 3D pode nos permitir traçar e observar padrões mais claramente

Perceba que encontrar valores ausentes é um benefício da exploração de dados e não da redução de dimensionalidade. Excluir uma coluna com vários valores ausentes pode ser considerado uma redução, mas simplesmente encontrar essa coluna não pode. Assim, temos a nossa resposta na alternativa E.

Gabarito: alternativa E.

---

13. Ano: 2023 Banca: TRC Assunto: Processamento de Linguagem Natural

A classificação de texto é usada em várias aplicações e vários domínios, assinale a opção que não é considerada uma aplicação em que a classificação de textos pode ser usada:

a) Na identificação de idioma, como identificar o idioma de novos tweets ou postagens.

b) Na tradução de texto de português para inglês.

c) Na atribuição de autoria, ou identificação de autores desconhecidos de textos de um grupo de autores.

d) Para a triagem de postagens em um suporte online para serviços de saúde mental.

e) Para segregar notícias falsas de notícias reais.

Comentário: Na lista acima, apenas a tradução de texto não é considerada uma tarefa de classificação de texto. Logo, temos a resposta na alternativa B.

Gabarito: alternativa B.

---

14. Ano: 2023 Banca: TRC Assunto: Processamento de Linguagem Natural (HARD)

Sobre classificação de texto, assinale a alternativa incorreta:

a) Naive Bayes é um classificador probabilístico que usa o teorema de Bayes para classificar textos com base na evidência vista em dados de treinamento.

b) O desequilíbrio entre elementos das classes é um dos motivos mais comuns para um classificador não ter um bom desempenho.

c) Semântica vetorial é o mapeamento entre o espaço vetorial proveniente da representação distribucional e o espaço vetorial proveniente da representação distribuída.

d) No contexto de PLN, a conversão de texto bruto em uma forma numérica adequada é chamada de representação de texto.

e) A representação de unidades de texto (caracteres, fonemas, palavras, frases, sentenças, parágrafos e documentos) como vetores de números é conhecida como modelo de espaço vetorial.

Comentário: Essa questão apresenta conceitos um pouco mais avançados, tenta levar você a se aprofundar em alguns tópicos do assunto ... de cara, posso dizer que o erro da questão está na alternativa C.

Lembre-se que, **embedding** é um termo usado quando a partir de um conjunto de palavras em um corpus, fazemos um mapeamento entre o espaço vetorial proveniente da representação distribucional e o espaço vetorial proveniente da representação distribuída.

Já a semântica vetorial refere-se ao conjunto de métodos de PLN que objetivam aprender as representações de palavras com base em propriedades de distribuição de palavras em um grande corpus.

Vamos agora comentar um pouco sobre as demais alternativas:

Naive Bayes é um **classificador probabilístico** que usa o teorema de Bayes para classificar textos com base na evidência vista em dados de treinamento. Ele **estima a probabilidade condicional de cada recurso de um determinado texto para cada classe** com base na ocorrência desse recurso naquela classe e multiplica as probabilidades de todos os recursos de um determinado texto para calcular a probabilidade final de classificação para cada classe. Por fim, escolhe a classe com probabilidade máxima.

A questão do desequilíbrio entre dados pode ser exemplificada da seguinte forma. Imagine que vamos classificar textos em relevantes ou irrelevantes para uma pesquisa acadêmica. Se existirem poucos exemplos de artigos relevantes (~ 20%) em comparação com os artigos não relevantes (~ 80%) no conjunto de dados. Esse desequilíbrio de classe faz com que o processo de aprendizagem seja desviado para a categoria de artigos não relevantes, pois há poucos exemplos de artigos “relevantes”.

As demais alternativas estão corretas e reforçam a ideia de representação textual e modelo de espaço vetorial vista ao longo da aula.

Gabarito: alternativa C.

---

15. Ano: 2023 Banca: TRC Assunto: Processamento de Linguagem Natural

Qual das ações abaixo não é considerada uma das etapas para construção de um sistema de classificação de textos.

a) Coletar ou criar um conjunto de dados rotulado adequado para a tarefa.

b) Dividir o conjunto de dados em dois (treinamento e teste) ou três partes: treinamento, validação (ou seja, desenvolvimento) e conjuntos de teste e, em seguida, decidir a(s) métrica(s) de avaliação.

c) Transformar o texto bruto em vetores de recursos.

d) Treinar um classificador usando os vetores de recursos e os rótulos correspondentes do conjunto de teste.

e) Usar a(s) métrica(s) de avaliação, comparar o desempenho do modelo no conjunto de teste.

Comentário: A questão apresentar uma casca de banana, mas faz parte do tipo de questão que pode aparecer na sua prova. A alternativa incorreta está na letra D. Na realidade, você deve treinar um classificador usando os vetores de recursos e os rótulos correspondentes do conjunto de **treinamento**.

Gabarito: alternativa D.

QUESTIONÁRIO DE REVISÃO E APERFEIÇOAMENTO

A ideia do questionário é elevar o nível da sua compreensão no assunto e, ao mesmo tempo, proporcionar uma outra forma de revisão de pontos importantes do conteúdo, a partir de perguntas que exigem respostas subjetivas.

São questões um pouco mais desafiadoras, porque a redação de seu enunciado não ajuda na sua resolução, como ocorre nas clássicas questões objetivas.

O objetivo é que você realize uma autoexplicação mental de alguns pontos do conteúdo, para consolidar melhor o que aprendeu ;)

Além disso, as questões objetivas, em regra, abordam pontos isolados de um dado assunto. Assim, ao resolver várias questões objetivas, o candidato acaba memorizando pontos isolados do conteúdo, mas muitas vezes acaba não entendendo como esses pontos se conectam.

Assim, no questionário, buscaremos trazer também situações que ajudem você a conectar melhor os diversos pontos do conteúdo, na medida do possível.

É importante frisar que não estamos adentrando em um nível de profundidade maior que o exigido na sua prova, mas apenas permitindo que você compreenda melhor o assunto de modo a facilitar a resolução de questões objetivas típicas de concursos, ok?

Nosso compromisso é proporcionar a você uma revisão de alto nível!

Vamos ao nosso questionário:

Perguntas

1) **Por que o pré-processamento de texto é feito em PNL?**
2) **Quais são algumas das vantagens de usar o Saco de Palavras (Bag of Word) para extrair recursos?**
3) **Quais são as diferenças entre TF-IDF e TF**
4) **Quais são os diferentes tipos de pré-processamento de texto?**
5) **O que é um Vetor One-Hot? Como eles podem ser usados no processamento de linguagem natural?O que são Sistemas de Recomendação?**
6) **Qual é a diferença entre Lemmatização e Stemming?**
7) **Qual é o uso da etiqueta PoS (parte da fala)?**

8) Quais são algumas ambiguidades enfrentadas no PNL?

Perguntas com respostas

1) Por que o pré-processamento de texto é feito em PNL?

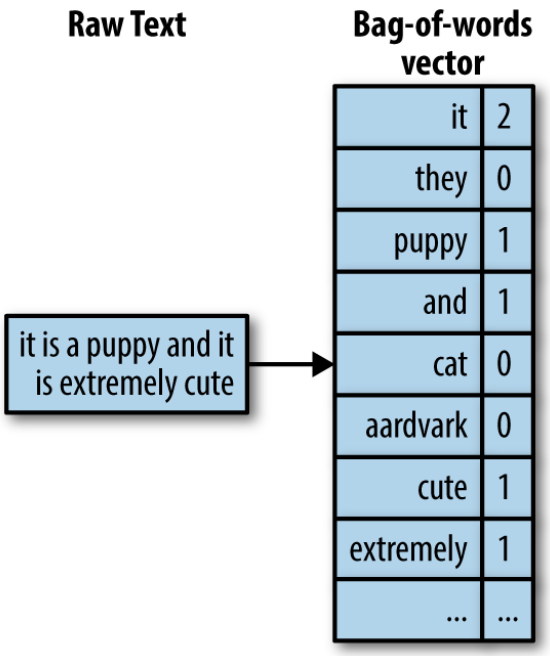
O pré-processamento de texto é feito para transformar um texto em uma forma mais digestível para que os algoritmos de aprendizagem de máquina possam ter melhor desempenho. Constata-se que em tarefas como análise de sentimentos, realizar alguns pré-processamentos, como remover stop-words, ajuda a melhorar a precisão do modelo de aprendizado de máquina. Alguns pré-processamento de texto comum feitos são:

- Remoção de tags HTML,
- Remoção de stop-words,
- Remoção de números,
- Lemmatização.
- Stemming

2) Quais são algumas das vantagens de usar o Saco de Palavras para extrair recursos?

Bag of Words identifica a ocorrência de palavras em um documento. Identifica o vocabulário e a presença de palavras conhecidas. Por isso, é muito simples e flexível.

É intuitivo que documentos que consistem em conteúdo semelhante sejam semelhantes de outras maneiras, como em relação ao significado. Assim, o processo BoW criará um grupo simples e rápido de recursos que podem ser usados. O modelo BoW pode ser o mais simples e complicado possível. A principal diferença é como o vocabulário das palavras é mantido, e como as diferentes palavras são pontuadas.



3) Quais são as diferenças entre TF-IDF e TF

Primeramente vamos olhar para a definição dos termos:

TF-IDF: Term Frequency-Inverse Document Frequency

TF: Term Frequency

Diferença:

- TF-IDF é uma *estatística numérica* que pretende refletir a *importância* de uma palavra para o documento em uma coleção do corpus.
- TF é uma *contagem* do número de vezes que uma palavra ocorre em um documento.
- O TF-IDF é dado por:

$$TFIDF = IDF(t) * TF(t,d)$$

Onde:

$$IDF(t) = log_e \frac{Número\ total\ de\ documentos\ em\ um\ corpos}{Número\ de\ documentos\ em\ que\ o\ termo\ \boldsymbol{t}\ aparece}$$

- TF é dada pela contagem de uma palavra no documento pelo número de palavras em d:

$$TF(t,d) = \frac{Número\ de\ ocorrências\ do\ termo\ \boldsymbol{t}\ no\ documento\ \boldsymbol{d}}{Total\ de\ termos\ no\ documento\ \boldsymbol{d}}$$

4) Quais são os diferentes tipos de pré-processamento de texto?

As etapas do pré-processamento de texto podem ser divididas 3 em grandes tipos:

Tokenização: É um processo onde um grupo de textos é dividido em peças menores, ou tokens. Os parágrafos são tokenizados em frases, e as frases são tokenizadas em palavras.

Normalização: Normalização do banco de dados é onde a estrutura do banco de dados é convertida em uma série de formas normais. O que ele consegue é a organização dos dados para parecer semelhante em todos os registros e campos. Da mesma forma, no campo da PNL, a normalização pode ser o processo de conversão de todas as palavras em sua minúscula. Isso faz com que todas as frases e tokens pareçam iguais e não complica o algoritmo de aprendizagem de máquina.

Remoção de ruído: É um processo de limpeza do texto. Fazer coisas como remover caracteres que não são necessários, como espaços brancos, números, caracteres especiais, etc.

5) O que é um Vetor One-Hot? Como eles podem ser usados no processamento de linguagem natural?O que são Sistemas de Recomendação?

Um one-hot é um grupo de bits que só tem um bit 1 e todos os outros bits são 0. No Processamento da Linguagem Natural, o vetor one-hot pode ser usado para representar uma frase na forma de uma matriz de tamanho N x N, onde N é o número de palavras individuais no corpus.

Por exemplo, a frase "Pedro escolheu um pedaço de pimenta em conserva" pode ser transformada em uma matriz de onde cada palavra é representada pelas 7 colunas. Portanto, a saída da frase é: 1 x 7[0000001, 0000010, 0000100, 0001000, 0010000, 0100000, 1000000]

Uma representação compreensível de um vetor de one-hot é mostrada pelo diagrama abaixo:

	1	2	3	4	5	6	7	
I	1	0	0	0	0	0	0	0
ate	0	1	0	0	0	0	0	0
an	0	0	1	0	0	0	0	0
apple	0	0	0	1	0	0	0	0
and	0	0	0	0	1	0	0	0
played	0	0	0	0	0	1	0	0
the	0	0	0	0	0	0	1	0
piano	0	0	0	0	0	0	0	1

6) Qual é a diferença entre Lemmatização e Stemming?

Stemming apenas remove os últimos caracteres de uma palavra, muitas vezes levando a significados e ortografias incorretas.

Por exemplo: eating -> eat, Caring -> Car.

A lemmatização considera o contexto e converte a palavra em sua forma básica significativa, que é chamada de Lemma.

Por exemplo: Stripes -> Strip (verb) -or- Stripe (noun), better -> good

7) Qual é o uso da etiqueta PoS (parte da fala)?

A marcação PoS é usada para classificar cada palavra em sua parte de fala. Partes da fala podem ser usadas para encontrar padrões gramaticais ou léxicos sem especificar a palavra usada. Em inglês, especialmente, a mesma palavra pode ser diferentes partes da fala, por isso, a marcação PoS pode ser útil para diferenciar entre eles.

8) Quais são algumas ambiguidades enfrentadas no PNL?

Algumas ambiguidades enfrentadas são:

Ambiguidade Lexical: Palavras que têm mais de um significado.

Ambiguidade sintática: Gramáticas usadas em frases são ambíguas, e mais de uma árvore analisada é correta para uma frase dada gramática.

Ambiguidade Semântica: Mais de uma interpretação semântica para uma sentença.

Ambiguidade Pragmática: Surge quando a declaração não é específica, e o contexto não fornece as informações necessárias para esclarecer a declaração.

Assim terminamos a nossa revisão sobre processamento de linguagem natural. Espero revê-lo ainda mais animando com o conteúdo na próxima aula.

Forte abraço e bons estudos.

"Hoje, o 'Eu não sei', se tornou o 'Eu ainda não sei'"

(Bill Gates)

Thiago Cavalcanti

Face: www.facebook.com/profthiagocavalcanti
Insta: www.instagram.com/prof.thiago.cavalcanti
YouTube: youtube.com/profthiagocavalcanti

ESSA LEI TODO MUNDO CONHECE:
PIRATARIA É CRIME.

Mas é sempre bom revisar o porquê e como você pode ser prejudicado com essa prática.

1. 

Professor investe seu tempo para elaborar os cursos e o site os coloca à venda.

2. 

Pirata divulga ilicitamente (grupos de rateio), utilizando-se do anonimato, nomes falsos ou laranjas (geralmente o pirata se anuncia como formador de "grupos solidários" de rateio que não visam lucro).

3. 

Pirata cria alunos fake praticando falsidade ideológica, comprando cursos do site em nome de pessoas aleatórias (usando nome, CPF, endereço e telefone de terceiros sem autorização).

4. 

Pirata compra, muitas vezes, clonando cartões de crédito (por vezes o sistema anti-fraude não consegue identificar o golpe a tempo).

5. 

Pirata fere os Termos de Uso, adultera as aulas e retira a identificação dos arquivos PDF (justamente porque a atividade é ilegal e ele não quer que seus fakes sejam identificados).

6. 

Pirata revende as aulas protegidas por direitos autorais, praticando concorrência desleal e em flagrante desrespeito à Lei de Direitos Autorais (Lei 9.610/98).

7. 

Concurseiro(a) desinformado participa de rateio, achando que nada disso está acontecendo e esperando se tornar servidor público para exigir o cumprimento das leis.

8. 

O professor que elaborou o curso não ganha nada, o site não recebe nada, e a pessoa que praticou todos os ilícitos anteriores (pirata) fica com o lucro.

Deixando de lado esse mar de sujeira, aproveitamos para agradecer a todos que adquirem os cursos honestamente e permitem que o site continue existindo.