Processamento de Linguagem Natural

Vetorização

Yuri Malheiros (yuri@ci.ufpb.br)

Introdução

- Em programação, para trabalhar com dados textuais, usamos normalmente um tipo string
- Entretanto, muitos algoritmos de processamento de linguagem natural trabalham apenas com números (ou vetores)
- O processo de transformação de um texto para uma representação numérica é chamado de vetorização

Tokenização

- O primeiro para passo transformar um texto em uma representação numérica é a tokenização
- Tokenização é o processo de quebrar um texto em pedaços chamados tokens

Tokenização

- Muitas vezes os tokens correspondem as palavras de um texto
  - "Eu comprei pão” (tokens: eu, comprei, pão)
- Mas nem sempre isso é claro
  - “Eu comprei R$ 10,00 de couve-flor” (quais são os tokens?)

Tokenização

- Em idiomas diferentes, a tokenização pode ser mais difícil
- Em inglês:
  - “You aren’t old”
  - “aren’t" pode ser considerado um token, mas pode ser dividido em “are" “not”

Tokenização

- Em idiomas diferentes, a tokenização pode ser mais difícil
- Em alemão, nomes compostos não são separados por espaços:
  - Lebensversicherungsgesellschaftsangestellter
  - (tradução: funcionário de uma empresa de seguro de vida)

Tokenização

- Podemos usar o scikit-learn para separar os tokens de um string:

```
count_vectorizer = CountVectorizer()
word_tokenizer = count_vectorizer.build_tokenizer()

word_tokenizer("eu gostei do filme. o filme é divertido")
```

- Vamos fazer um exemplo no Jupyter Notebook...

Tokenização

- As palavras mais utilizadas nos textos são artigos, preposições e conjunções
- Essas palavras isoladamente tem pouco significado e dominam as contagens de tokens

Stopwords

- Palavras extremamente comuns em um idioma são chamadas de stopwords
- Elas são normalmente retiradas do texto quando estamos trabalhando com processamento de linguagem

Stopwords

- Quais palavras retirar?
  - Não existe uma lista única por idioma
  - Podemos encontrar em bibliotecas ou na Internet algumas listas disponíveis

Stopwords

- A biblioteca NLTK possui uma lista de stopwords em português

```
import nltk

stopwords = nltk.corpus.stopwords.words('portuguese')
stopwords
```

- Algumas das stopwords: 'de', 'a', 'o', 'que', 'e', 'é', 'do', 'da', 'em', 'um', ‘para’, ...

Stopwords

- Usando o scikit-learn para tokenização, temos:

```
count_vectorizer = CountVectorizer(stop_words=stopwords)
analyzer = count_vectorizer.build_analyzer()

analyzer('eu gostei do filme o filme é divertido')
```

- Que vai retornar a lista: [gostei, filme, filme, divertido]

- Vamos fazer um exemplo no Jupyter Notebook...

Stopwords

- Além de excluir palavras que costumam não ter muito significado, com stopwords também simplificamos os nossos dados
  - Diminui o número de palavras
  - Diminui o tamanho dos vetores

Normalização

- Existem outras formas de simplificar e uniformizar textos
- Essas operações são chamadas de normalização
- Uma das mais comuns é tornar todo o texto em minúsculo
  - Isto faz com que palavras que comecem com letras maiúsculas ou minúsculas sejam consideradas como as mesmas palavras

Normalização

- Stemming
  - Retira-se os prefixos e sufixos de uma palavra, mantendo apenas o seu radical
  - Exemplo:
    - aluno, aluna, alunos e alunas → alun
  - As variações de uma mesma palavra são consideradas o mesmo token

Normalização

- Lematização
  - Transforma formas flexionadas de uma palavra num formato base
  - Exemplo:
    - aluno, aluna, alunos e alunas → aluno
    - tenho, tiver, tinha, tem → ter
  - O resultado da lematização é uma palavra existente no dicionário

Normalização

- Lematização e Stemming servem para simplificar um texto diminuindo o seu vocabulário
- Stemming é um processo mais rápido, mas pode ser menos eficaz na simplificação, além de poder gerar palavras que não existem
- A biblioteca NLTK implementa Stemming para língua portuguesa

```
stemmer = nltk.stem.RSLPStemmer()
stemmer.stem("conseguimos")
```

One-hot encoding

- A forma mais simples e comum de transformar uma palavra em um vetor é a técnica one-hot encoding
- Nela, inicialmente definimos um conjunto de palavras que consideraremos (vocabulário)
- No one-hot encoding, cada palavra é representada por um vetor único de tamanho N
  - N é o tamanho do vocabulário

One-hot encoding

- Dado o texto “eu gostei do filme”
- Podemos definir o vocabulário como: [eu, gostei, do, filme]
- Assim, temos os seguintes vetores:
  - `eu      [1, 0, 0, 0]`
  - `gostei  [0, 1, 0, 0]`
  - `do      [0, 0, 1, 0]`
  - `filme   [0, 0, 0, 1]`

Bag of words

- Podemos utilizar essa ideia para representar um documento inteiro como um vetor
- O modelo bag of words (BoW) descreve a ocorrência das palavras em um documento
- Para usá-lo também precisamos definir um vocabulário

Bag of words

- Dados os documentos:
  - eu gostei do filme
  - eu não assisti o filme
- Temos o vocabulário: [eu, gostei, do, filme, não, assisti, o]

- Com isso, vamos criar uma matriz termo-documento onde cada linha representa uma palavra e cada coluna um documento
- Quando a palavra ocorrer em um documento, marcaremos a posição com 1

Bag of words

	Doc 1	Doc 2
eu	1	1
gostei	1	0
do	1	0
filme	1	1
não	0	1
assisti	0	1
o	0	1

Bag of words

- Documentos similares devem ter suas representações no BoW também similares
- Entretanto, o BoW não leva em consideração a ordem das palavras
  - Por isso o nome bag (as palavras ficam misturadas, como se colocadas num saco)

Bag of words

- Dados os documentos:
  - Ambiente bom, mas atendimento ruim e prato principal ruim
  - Prato principal bom e atendimento bom, mas ambiente ruim

- Note que os dois documentos são representados pelo mesmo vetor
  - Mas eles são diferentes!

	Doc 1	Doc 2
bom	1	1
ambiente	1	1
mas	1	1
atendimento	1	1
ruim	1	1
e	1	1
prato	1	1
principal	1	1

Bag of words

- Para contornar o problema anterior, podemos contar quantas vezes cada token aparece

- Os vetores agora são diferentes
- Podemos perceber que o documento 1 é mais negativo que o documento 2

	Doc 1	Doc 2
bom	1	2
ambiente	1	1
mas	1	1
atendimento	1	1
ruim	2	1
e	1	1
prato	1	1
principal	1	1

Bag of words

- Mesmo contando a frequência, ainda temos problemas com textos diferentes, mas que contenham exatamente as mesmas palavras

- Dados os documentos:
  - o jogador acertou a bola
  - A bola acertou o jogador

	Doc 1	Doc 2
o	1	1
jogador	1	1
acertou	1	1
a	1	1
bola	1	1

Bag of words

- Vamos ver um exemplo no Jupyter Notebook...

TF-IDF

- No bag of words, os vetores criados tem valores altos para as dimensões de palavras que se repetem muito
- Retirar stopwords ameniza esse problema, mas podemos ir além

TF-IDF

- Palavras muito frequentes tendem a aparecer em praticamente todos os documentos
- Assim, elas não ajudam a diferenciar os documentos
  - Você não diferencia duas coisas através de suas características iguais

TF-IDF

- Para resolver esse problema, podemos penalizar palavras que aparecem muito em todos os documentos analisados
- Essa abordagem é chamada de Term Frequency-Inverse Document Frequency (TF-IDF)

TF-IDF

- O TF-IDF é calculado através do produto de dois fatores
- O primeiro é a frequência de uma palavra **t** em um documento **d**
  - Ou seja, contamos quantas vezes **t** aparece em **d**
    - $tf_{t,d} = cont(t, d)$
- Este fator representa a importância de um termo em um documento

TF-IDF

- Dado o documento:
  - 1. A cada episódio, essa série me impressiona mais, me faz ficar cada vez mais apaixonado
- Vamos calcular a frequência das palavras “me”, “mais" e “série”
  - $tf_{me,1} = 2$
  - $tf_{mais,1} = 2$
  - $tf_{serie,1} = 1$
- Pela frequência, as palavras “me" e “mais" são mais importantes que “série”

TF-IDF

- No segundo fator do TF-IDF, calculamos a frequência de documentos que contêm uma palavra **t** (df_t)
  - Ou seja, contamos o número de documentos que **t** ocorre
- No conjunto de dados analisado, temos:
  - $df_{me} = 215$
  - $df_{mais} = 273$
  - $df_{serie} = 6$

TF-IDF

- No TF-IDF queremos dar importância para as palavras que ajudam a diferenciar os documentos
  - Ou seja, palavras que não aparecem em muitos documentos
- Por isso, usamos o inverso da frequência de documentos
  - $idf_t = \log \frac{N}{df_t}$
  - Onde **N** é o número de documentos
  - Como em muitos casos temos uma grande quantidade de documentos, então usamos o log para reduzir o intervalo de variação dessa medida

TF-IDF

- No conjunto de dados analisado, temos:
  - $idf_{me} = \log \frac{2787}{215} = 1{,}1127$
  - $idf_{mais} = \log \frac{2787}{273} = 1{,}009$
  - $idf_{serie} = \log \frac{2787}{6} = 2{,}667$
- Percebam que quanto maior o número de documentos contendo uma palavra, menor é o valor do idf_t

TF-IDF

- Por fim, multiplicamos os dois termos para calcular o TF-IDF
  - $tfidf_{me,1} = tf_{me,1} \times idf_{me} = 2 \times 1{,}1127 = 2{,}2254$
  - $tfidf_{mais,1} = tf_{mais,1} \times idf_{mais} = 2 \times 1{,}009 = 2{,}0179$
  - $tfidf_{serie,1} = tf_{serie,1} \times idf_{serie} = 1 \times 2{,}667 = 2{,}667$

- Notem que mesmo a palavra “serie” aparecendo uma única vez no documento, ela obteve o maior valor de TF-IDF, pois ela é mais rara nos documentos do conjunto de dados

TF-IDF

- Vamos ver um exemplo no Jupyter Notebook...

Similaridade

- Representando documentos como vetores, podemos interpretá-los como pontos em um espaço N dimensional
  - Onde N é o tamanho do vocabulário
  - Cada palavra representa uma dimensão

Similaridade

- Partindo da intuição de que documentos com muitas palavras em comum são documentos semelhantes
- Usando bag of words ou TF-IDF, vetores parecidos representam documentos parecidos
  - Os pontos que esses vetores representam estarão próximos no espaço

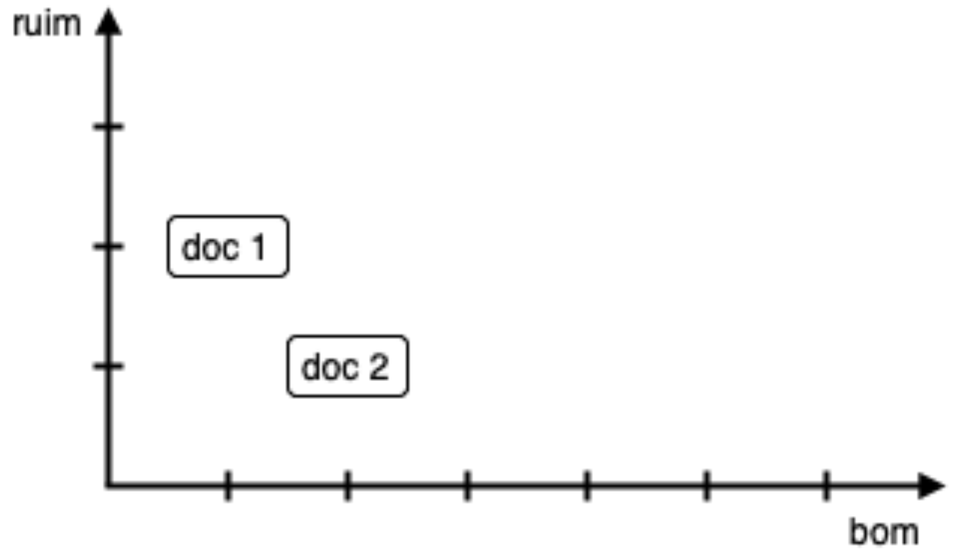
Similaridade

- Dados os documentos:
  - Ambiente bom, mas atendimento ruim e prato principal ruim
  - Prato principal bom e atendimento bom, mas ambiente ruim
- Cada documento pode ser representado como um ponto em um espaço de 8 dimensões
  - Não temos como visualizar 8 dimensões, então vamos utilizar apenas 2

	Doc 1	Doc 2
bom	1	2
ambiente	1	1
mas	1	1
atendimento	1	1
ruim	2	1
e	1	1
prato	1	1
principal	1	1

Similaridade

- No gráfico abaixo:
  - O eixo X é a dimensão da palavra "bom"
  - O eixo Y é a dimensão da palavra "ruim"



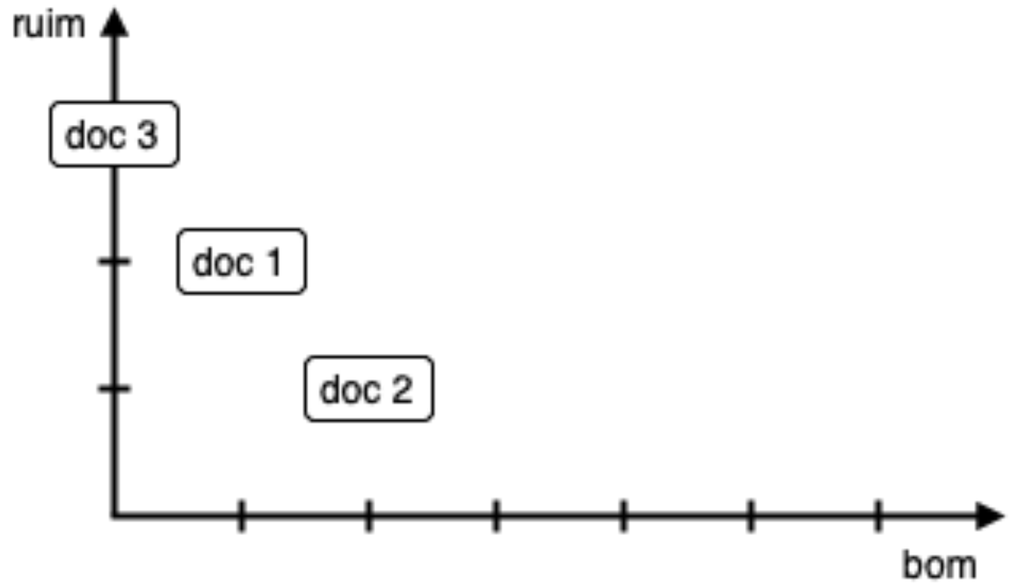
Similaridade

- Adicionando um novo documento:
  - Ambiente bom, mas atendimento ruim e prato principal ruim
  - Prato principal bom e atendimento bom, mas ambiente ruim
  - Prato principal ruim, atendimento ruim e ambiente ruim

	Doc 1	Doc 2	Doc 3
bom	1	2	0
ambiente	1	1	1
mas	1	1	0
atendimento	1	1	1
ruim	2	1	3
e	1	1	1
prato	1	1	1
principal	1	1	1

Similaridade

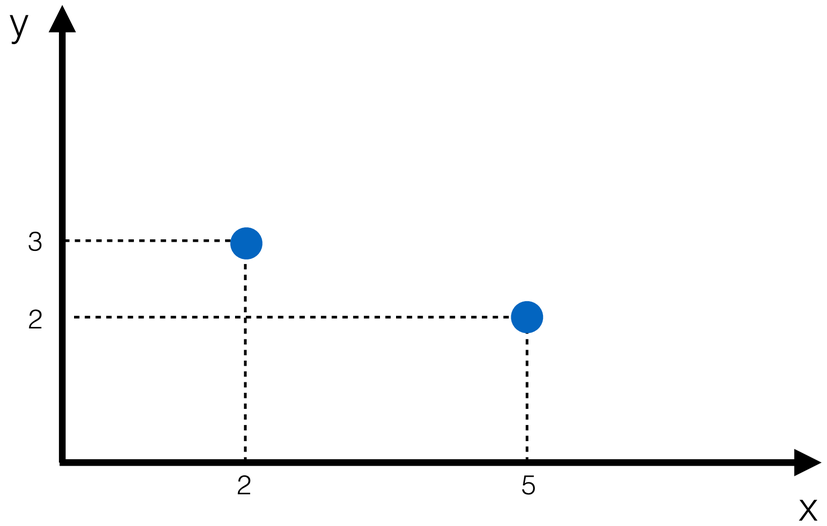
- Temos o gráfico:



- Perceba que o documento 3 está mais próximo do documento 1
  - Ambos são mais negativos que o documento 2

Similaridade

- Conseguimos avaliar visualmente a similaridade entre os três documentos
- Nem sempre isso é possível
- Precisamos de uma medida mais precisa de similaridade entre vetores
- Para isso podemos usar o produto escalar entre vetores:



$$a \cdot b = a_x b_x + a_y b_y$$

$$a \cdot b = 2 \cdot 5 + 3 \cdot 2 = 16$$

Similaridade

- Valores de uma mesma dimensão dos documentos são multiplicados e depois os resultados são somados
- Lembrando que cada palavra representa uma dimensão de vetor
- Então, se dois documentos tiverem muitas palavras iguais, o produto escalar tende a ter um valor alto
- Por outro lado, se dois documentos tiverem muitas palavras diferentes, muitos termos do produto escalar serão iguais a zero, fazendo com que o resultado tenha um valor baixo

Similaridade

- Dados os dois vetores:
  - $v = [1,1,1,1]$
  - $w = [1,1,1,0]$
- Temos:
  - $v \cdot w = (1 \times 1) + (1 \times 1) + (1 \times 1) + (1 \times 0) = 3$

Similaridade

- Dados os dois vetores:
  - $v = [1,1,1,1]$
  - $u = [1,0,0,0]$
- Temos:
  - $v \cdot u = (1 \times 1) + (1 \times 0) + (1 \times 0) + (1 \times 0) = 1$

- Com isso concluímos que **v** se assemelha mais a **w** do que a **u**.

Similaridade

- Dados os dois vetores: $v = [1,1,1,1]$ e $w = [1,1,1,1]$
- Temos: $v \cdot w = (1 \times 1) + (1 \times 1) + (1 \times 1) + (1 \times 1) = 4$

- Dados os dois vetores: $m = [2,2,2,2]$ e $n = [2,2,2,2]$
- Temos: $m \cdot n = (2 \times 2) + (2 \times 2) + (2 \times 2) + (2 \times 2) = 16$

- Os vetores **m** e **n** são mais semelhantes entre eles que **v** e **w**? Isso não faz muito sentido.

Similaridade

- O produto escalar favorece vetores longos
  - Ou seja, vetores que tem valores maiores em suas dimensões
- Documentos longos podem aparecer como mais similares simplesmente por possuírem mais palavras
- O comprimento de um vetor **v** é calculado assim: $$|v| = \sqrt{\sum_{i=1}^{N} v_i^2}$$
- Onde v_i é o valor da dimensão i

Similaridade

- Para calcular a similaridade independentemente do comprimento do vetor, vamos dividir o produto escalar pelo produto do comprimento dos vetores

$$similaridade(v, w) = \frac{v \cdot w}{|v||w|}$$

- Esse valor é o mesmo do cosseno do ângulo entre os vetores

$$similaridade(v, w) = cos(v, w) = \frac{v \cdot w}{|v||w|}$$

Similaridade

- O scikit-learn implementa uma função para calcular a similaridade de cosseno:

```
from sklearn.metrics.pairwise import cosine_similarity

cosine_similarity(vec1, vec2)
```

- Cada parâmetro é uma lista de vetores
- A função vai calcular a similaridade de cada vetor do primeiro parâmetro para cada vetor do segundo parâmetro

Exemplo

- Usando um dataset de reviews, vamos criar um programa que encontra os reviews mais parecidos, dado um review de entrada

Embeddings

- Até agora representamos palavras como vetores esparsos
  - Vetores longos com uma dimensão por palavra
  - Com um vocabulário grande, o vetor costuma ter muitos zeros
- Embeddings são vetores densos
  - Eles têm um número de dimensões bem menor que o número de palavras de um vocabulário
  - Cada dimensão não tem uma interpretação clara

Embeddings

- Na prática, os vetores densos funcionam melhor em aplicações de PLN
- Alguns motivos:
  - Trabalhamos com menos dimensões
  - Palavras similares tendem a ser vetores similares

Embeddings

- O word2vec é um dos métodos mais conhecidos de embedding
  - Ele está disponível para uso livremente
- Os embeddings do word2vec são estáticos
  - Ou seja, temos uma representação fixa para cada palavra de um vocabulário
  - Com isso, podemos reutilizar embeddings criados por terceiros

Embeddings

- A ideia do word2vec é usar as palavras próximas a uma palavra **w** para representar **w**
- Entretanto, ao invés de contar a frequência das palavras, usamos um classificador para prever a probabilidade de uma determinada palavra aparecer próxima a **w**
- O word2vec não usa o resultado da classificação, ele usa os pesos aprendidos pelo classificador

Embeddings

- Uma das revoluções do word2vec é que você pode usar um texto sem qualquer marcação extra para treinar esse classificador
- Para saber se uma palavra costuma aparecer ou não próxima de outra palavra, basta olhar o texto
- Por exemplo, “azul" costuma aparecer perto da palavra “Terra”? Basta analisarmos um conjunto grande de textos para saber se isso é verdade ou não.
- Por outro lado, “amarelo” costuma aparecer perto da palavra “Terra”? Provavelmente não tanto quanto “azul”

Embeddings

- Dado o texto
  - ... limão, uma colher de geléia de damasco, uma xícara ...
- Vamos utilizar uma janela de 2 palavras de contexto
- Nossa palavra alvo é "geléia"
  - ... limão, uma [colher de **geléia** de damasco], uma xícara ...

Embeddings

- ... limão, uma [colher de **geléia** de damasco], uma xícara ...

X	Y
geléia	colher
geléia	de
geléia	damasco

Embeddings

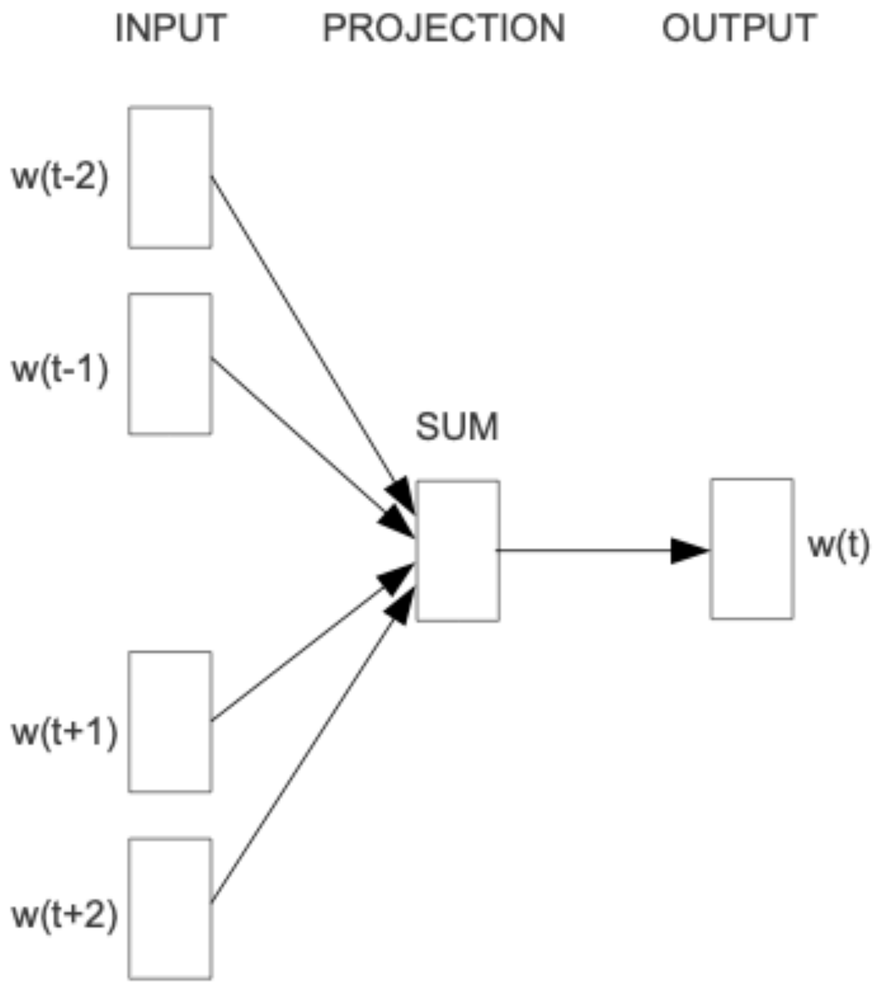
- O objetivo do treinamento do classificador é, dada a entrada **X**, encontrar uma saída o mais similar possível a **Y**

- A similaridade entre dois vetores pode ser calculada usando o produto escalar

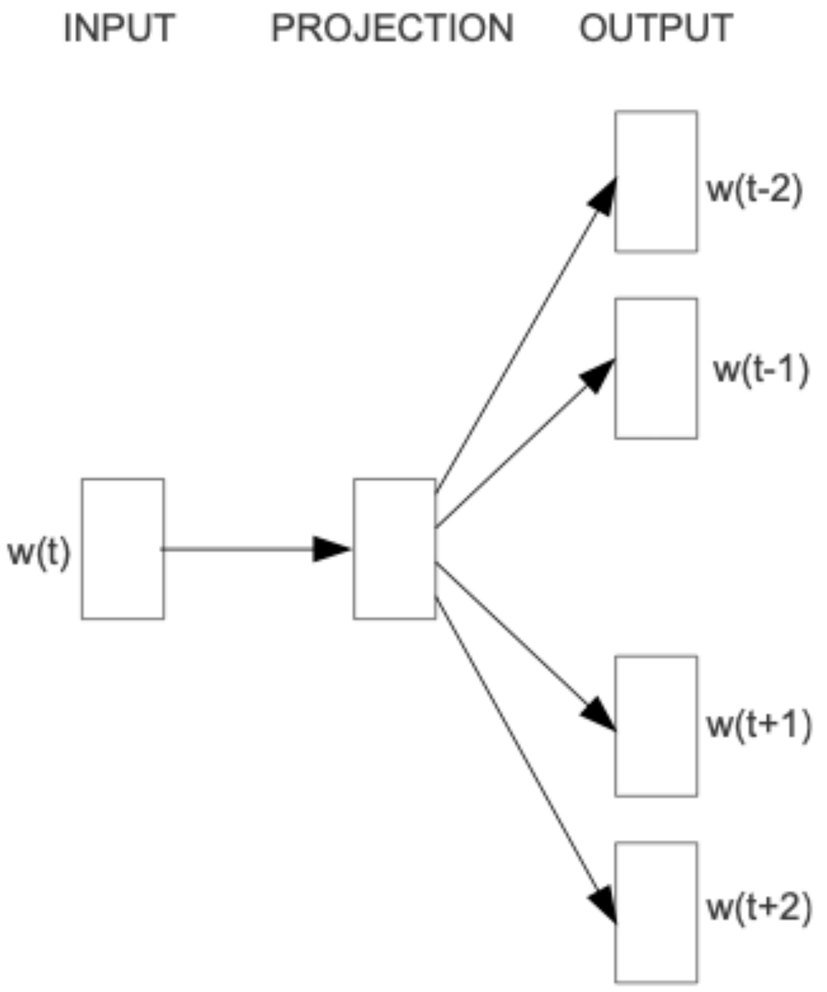
- O processo de treinamento de aprendizagem de máquina é utilizado para ajustar os pesos do classificador para ir em direção ao objetivo

Embeddings

- Classificar as palavras vizinhas dada uma palavra alvo é chamado de modelo skip gram
- Classificar a palavra alvo dadas as palavras vizinhas é chamado de modelo CBOW (continuous bag of words)



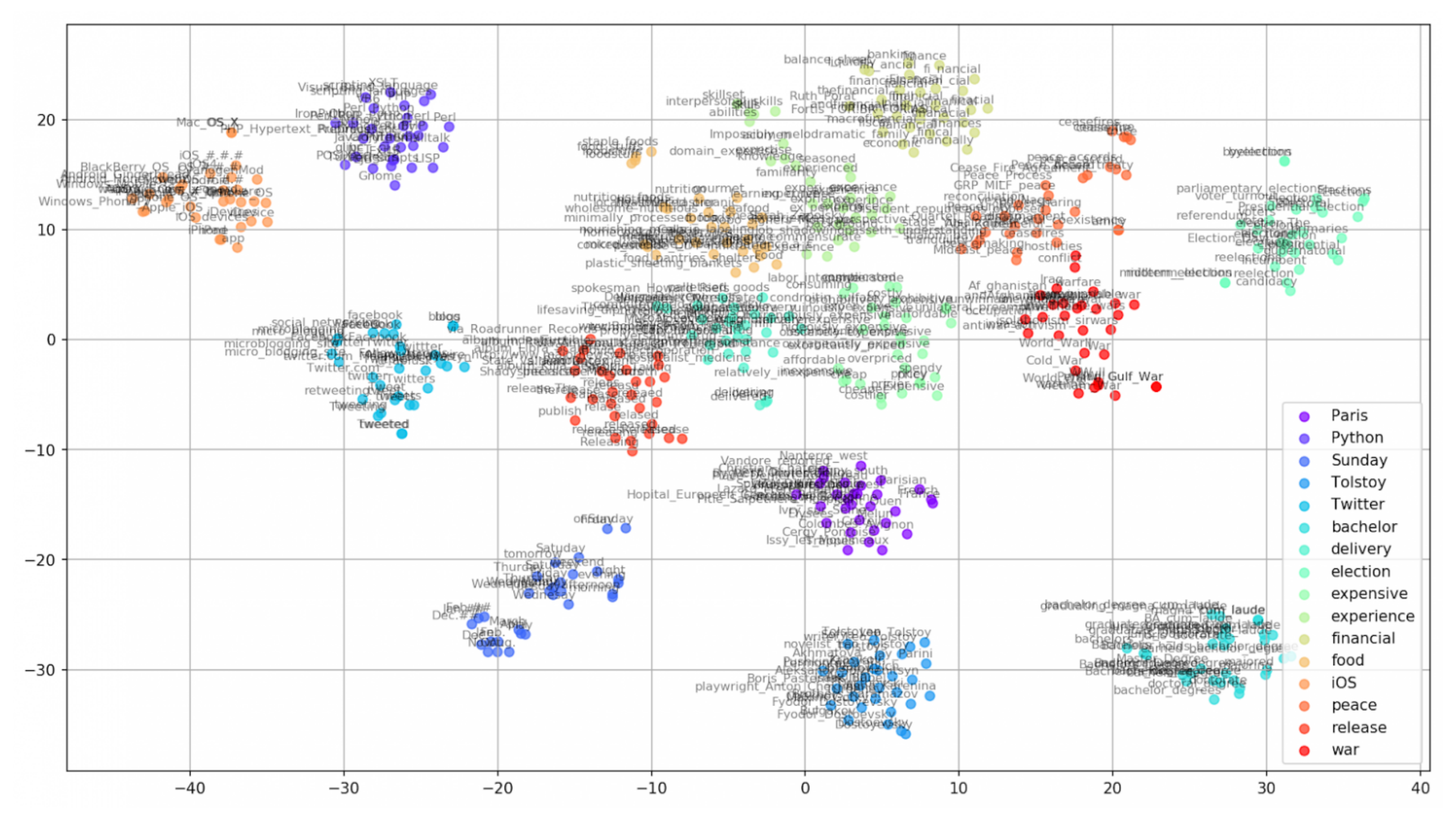
CBOW



Skip-gram

Embeddings

- O site do NILC traz embeddings em português para download:
  - http://www.nilc.icmc.usp.br/embeddings
- Outros embeddings podem ser encontrados livremente na Internet:
  - http://vectors.nlpl.eu/repository/


Paris
Python
Sunday
Tolstoy
Twitter
bachelor
delivery
election
expensive
experience
financial
food
iOS
peace
release
war

Embeddings

- Vamos utilizar embeddings com Python