Universidade Federal Fluminense
Instituto de Matemática e Estatística
Curso de Bacharelado em Matemática

Representação Vetorial de Textos: Doc2Vec

Autor: Gabriel Gonzalo Ledesma Valenotti

Orientador: Prof. Jéssica Quintanilha Kubrusly

Niterói – RJ

Julho / 2022

Ficha catalográfica automática - SDC/BIME
Gerada com informações fornecidas pelo autor

V152r Valenotti, Gabriel Gonzalo Ledesma
Representação Vetorial de Textos: Doc2Vec / Gabriel Gonzalo Ledesma Valenotti ; Jéssica Quintanilha Kubrusly, orientadora. Niterói, 2022.
49 f. : il.

Trabalho de Conclusão de Curso (Graduação em Matemática)- Universidade Federal Fluminense, Instituto de Matemática e Estatística, Niterói, 2022.

1. Mineração de texto. 2. Aprendizado de máquinas. 3. Produção intelectual. I. Kubrusly, Jéssica Quintanilha, orientadora. II. Universidade Federal Fluminense. Instituto de Matemática e Estatística. III. Título.

CDD -

Bibliotecário responsável: Debora do Nascimento - CRB7/6368

Gabriel Gonzalo Ledesma Valenotti

Representação Vetorial de Textos:
Doc2vec

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação de Bacharelado em Matemática. da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Bacharel em Matemática.

Aprovada em 28 de Julho de 2022.

BANCA EXAMINADORA

Prof. Dra. Jessica Quintanilha Kubrusly - Orientador
Universidade Federal Fluminese - UFF

Prof. Karina Yuriko Yaginuma
Universidade Federal Fluminese - UFF

Prof. Marcos de Oliveira Lage Ferreira
Universidade Federal Fluminese - UFF

Niterói – RJ
2022

Resumo

O objetivo principal deste trabalho é apresentar o algoritmo não supervisionado para representação vetorial de documentos, *Doc2vec*, o qual consiste em associar cada texto a um vetor numérico considerando seu contexto no corpus. *Doc2Vec* é uma extensão do algoritmo *Word2Vec*, o qual também será apresentado neste trabalho. Através de *Doc2vec* veremos a importância de considerar a relação semântica entre as palavras com seu contexto. Finalmente, compararemos o desempenho de classificadores usando representações vetoriais *Doc2Vec* com uma representação numérica que não considera a relação entre as palavras com seu contexto, *Bag of Word* (BOW). Veremos que com a representação de *Doc2Vec* todos os classificadores apresentaram um melhor desempenho computacional.

Palavras chaves: Doc2Vec, Wor2Vec, BOW, Word Embedding.

Abstract

The main objective of this work is to present the unsupervised algorithm for vector representation of documents, *Doc2Vec*, which consists of associating each text with a numerical vector considering its context in the corpus. *Doc2Vec* is an extension of the Word2Vec algorithm, which will also be presented in this work. Through *Doc2Vec* we will see the importance of considering the semantic relationship between words and their context. By *Doc2Vec* we will see the importance of considering the semantic relationship between words and their context. Finally, we will compare the performance of classifiers using *Doc2Vec* vs a numerical representation that does not consider the relationship between words and their context, *Bag of Word* (BOW). We will see that with the representation of *Doc2Vec* all classifiers presented better computational performance.

Key-words: Doc2Vec, Wor2Vec, BOW, Word Embedding.

Dedicado a mi madre.

Agradecimentos

A minha família, em primeiro lugar, pelo apoio desde sempre. Sem eles eu não estaria estudando até hoje.

A meus amigos e colegas, que, mesmo estando longe, sempre me acompanharam e apoiaram durante meus estudos e nos momentos quando mais os precisava.

A minha orientadora, Prof. Jessica Kubrusly, pela dedicação e, principalmente, pela paciência.

À Universidad Federal Fluminense, por ser a instituição onde estou me formando como matemático. Meu eterno gratidão para a UFF.

Lista de Figuras

Lista de Tabelas

Sumário

[0]Criado por: NDE bacharelado em matemática da UFF. Versão: 2020_12_20. Disponível em: http://ime-uff.org/

Capítulo 1

Introdução

Na área de Processamento de Linguagem Natural (PLN), a classificação de texto é uma das atividades com maior demanda atualmente, pois é capaz de resolver uma ampla gama de problemas bastante versáteis. Alguns modelos de classificação de texto são usados em vários setores, para atender a diversos propósitos, dentre os quais podemos destacar:

- Análise de sentimentos.
- Monitoramento de mídias sociais.
- Comentários do cliente.
- Pesquisa de mercado.
- Geração Automática de Texto.

Um grande problema com a modelagem de texto é que os algoritmos de *machine learning* não podem trabalhar diretamente com texto bruto; o texto deve ser convertido em números ou, especificamente, em vetores de números. Além disso, estes algoritmos preferem entradas e saídas bem definidas e de dimensão fixa. Por esse motivo, para poder aplicar algoritmos de *machine learning* em conjuntos de textos (documentos, frases, palavras, etc) é necessário o conhecimento de métodos e técnicas para representar de forma numérica qualquer conjunto de texto.

O objetivo principal deste trabalho é comparar dois métodos de representação vetorial de texto: *Doc2Vec* e *Bag of Word* (BOW). *Doc2Vec* é um método representação vetorial de texto baseado fortemente na relação que existe entre cada palavra com seu

contexto. Por outro lado, *Bag of Word* (BOW) também é um método representação vetorial de texto mas que não considera esta relação. Para poder comparar estes métodos vamos escolher 6 classificadores tradicionais, cujos dados de entrada serão cada representação vetorial, e comparaemos seus desempenhos. Veremos que com a representação de *Doc2Vec* todos os classificadores apresentaram um melhor desempenho computacional.

Este trabalho está organizado da seguinte forma: no capítulo 2 serão descritos os mecanismos de funcionamento dos métodos *Bag of word* e *Doc2vec*. No caso de *Doc2vec* serão descritas suas duas arquiteturas. Também será explicada uma breve motivação a este método de representação. Na última seção serão descritos os 6 classificadores e a implemetação computacional de cada um deles. No capítulo 3 serão apresentados e comparados, por meio de tabelas, os desempenhos obtidos por cada classificador usando os dois métodos de representação de texto.

Capítulo 2

Metodologia

Neste trabalho monográfico serão apresentados dois métodos de representação vetorial de texto, o método *Bag of word* (BOW) e o método *Doc2vec*. No último capítulo, serão avaliados vários classificadores para comparar seus desempenhos utilizando como dados de entrada cada uma destas representações vetoriais.

O banco de dados utilizado para este trabalho coleta um conjunto de dados de um e-commerce de roupa feminina com as avaliações escritas pelos clientes. Para cada opinião existe um rótulo que classifica se a opinião é considerada positiva ou negativa. A categoria **Sim** será representada pelo número 1 e a categoria **Não** será representada pelo número 0.

O objetivo dos classificadores será prever com a maior exatidão possível se uma determinada opinião pode ser considerada como positiva ou negativa.

Para este banco de dados estamos considerando 8344 opiniões pré-rotuladas (com 1 ou 0) e seguiremos a seguinte metodologia:

- Descrição do banco de dados.
- Efetuar o pré-processamento nos dados de texto.
- Construir o vocabulário para cada modelo (*doc2vec* e *Bag of Word*).
- Construir as respectivas representações vetoriais (de dimensão 100).
- Selecionar 6 classificadores.
- Calcular e comparar as acurácias obtidas pelos algoritmos de classificação usando a representações *Doc2Vec* e *Bag Of Word*.

Na seção de anexos são apresentadas as tabelas com as *stop-word* e as palavras selecionadas para construir a representação por *Bag of Word*.

2.1 Representação de texto Bag of Word (BOW)

Bag of word (BOW) é um método de representação vetorial de dados de texto. Esta representação resulta bastante útil para tarefas de classificação de textos por meio de algoritmos de *machine learning*. Também tem sido aplicado (com sucesso) a muitos problemas de modelagem de idiomas. Uma vantagem importante de *bag of word* é que seu funcionamento é simples de entender e de implementar.

Este método não considera a ordem das palavras no documento, ele apenas se preocupa pela ocorrência de palavras conhecidas, não pela posição onde elas estão. Basicamente, o método precisa de dois objetos:

- Um vocabulário de palavras conhecidas.
- Uma medida da presença de palavras conhecidas.

O método assume a seguinte ideia: documentos são semelhantes se tiverem conteúdos semelhantes. A partir do conteúdo podemos inferir algo sobre o significado do documento.

A maior complexidade do método está na decisão de como definir o vocabulário de palavras conhecidas (ou **tokens**) e na forma de marcar a presença de palavras conhecidas. Para ter uma ideia mais clara do funcionamento de BOW vamos analisar o seguinte exemplo.

Exemplo: Vamos considerar as seguintes frases:

It was the best of times,

it was the worst of times,

it was the age of wisdom,

Agora vamos definir o vocabulário fazendo uma lista de todas as palavras extraídas das frases. Aqui será considerado apenas palavras únicas, ignorando maiúsculas e acentuação:

$$V = [\textit{it, was, the, best, of, times, worst, age, wisdom}]$$

Este é um vocabulário de 9 palavras de um **corpus** contendo 18 palavras (total de palavras das 3 frases).

Agora vamos marcar as palavras em cada documento. Como o vocabulário tem 9 palavras, podemos usar uma representação de documento de comprimento fixo de 9, com uma posição no vetor para marcar cada palavra.

O método mais simples para marcar a presença de palavras é através de valores booleanos, 0 para ausente, 1 para presente. Outra maneita útil de marcar é por meio da frequência de cada palavra no texto.

Usando a ordem arbitrária de palavras listadas acima em nosso vocabulário V, a representação do documento seria a seguinte:

$$\textit{It was the best of times} = [1, 1, 1, 1, 1, 1, 0, 0, 0]$$

$$\textit{it was the worst of times} = [1, 1, 1, 0, 1, 1, 1, 0, 0]$$

$$\textit{it was the age of wisdom} = [1, 1, 1, 0, 1, 0, 0, 1, 1]$$

Também podemos apresentar os resultados na seguinte tabela:

Texto	it	was	the	best	of	times	worst	age	wisdom
It was the best of times	1	1	1	1	1	1	0	0	0
it was the worst of times	1	1	1	0	1	1	1	0	0
it was the age of wisdom	1	1	1	0	1	0	0	1	1

Tabela 2.1: Representação por *Bag of Word* (BOW)

Esses vetores de 0 e 1 também são chamados de vetores **one-hot encoded**. Os vetores one-hot encoded são vetores com todas as entredas nulas menos em uma entrada onde recebe o valor de 1 para indicar a posição de um membro escolhido em um conjunto.

2.1.1 Problemas com Bag of Word

À medida que o tamanho do vocabulário aumenta, o mesmo acontece com a representação vetorial dos documentos. No exemplo anterior, o comprimento do vetor do documento é igual ao número de palavras conhecidas.

Assim, para um corpus muito grande, como milhares de textos, o comprimento do vetor pode ser milhares ou milhões de posições. Além disso, cada documento pode conter muito poucas das palavras conhecidas no vocabulário. Isso resulta em um vetor com muitos 0's, chamado de **vetor esparso** (**sparse vector** em inglês) ou representação esparsa (**sparse representation**).

Estes vetores esparsos requerem mais recursos de memória e computação, o que pode tornar o processo de classificação muito desafiador para algoritmos tradicionais. Por isso, temos a necessidade de diminuir o tamanho do vocabulário quando usamos o modelo *Bag of word*.

Existem técnicas simples de limpeza de texto que podem ser usadas como um primeiro passo, tais como:

- Utilizar só letras minúsculas
- Eliminar acentuação
- Eliminar palavras frequentes que não contêm muita informação, chamadas de **stop words**, como *a*, *de*, *o*, etc.
- Corregir palavras mal escritas.
- Reduzir palavras à sua raíz (por exemplo, *jogando* e *jogar*).

Uma abordagem mais sofisticada é criar um vocabulário de palavras agrupadas. Isto permite que *Bag of word* capture um pouco mais de significado (semântica) do documento.

Nesta abordagem, cada palavra ou token é chamado de **gram**. Criar um vocabulário de pares de palavras é, por sua vez, chamado de modelo **bi-gram**. Apenas são considerados os bi-grams que aparecem no corpus, não todos os bi-grams possíveis de formar.

Em geral, um N-gram é uma sequência de N palavras que definem um token de tamanho N. Assim, um 2-gram (ou bi-gram) é uma sequência de duas palavras de palavras, um 3-gram (ou tri-grama) é uma sequência de três palavras de palavras. Por exemplo, os bi-grams do texto *It was the best of times* são: (*it, was*), (*was, the*), (*the, best*), (*best, of*), (*of, times*).

Uma grande limitação com modelo *Bag of word* é a falta de relação semântica entre palavras. Descartar a ordem das palavras implica ignorar o contexto e, por sua vez, o significado das palavras no documento (semântica). A ordem nas palavras podem oferecer muito ao modelo, por exemplo, entender a diferença entre *amigo velho* e *velho amigo*. Também os sinônimos, por exemplo, *bicicleta velha* e *bicicleta usada*, e muitos outros casos.

Um método alternativo para representar vetorialmente um texto, considerando o contexto e a semântica, é o algoritmo **Doc2vec** que apresentaremos no seguinte capítulo.

2.2 Representação de texto Doc2Vec

Doc2vec é um algoritmo não supervisionado utilizado para gerar vetores que representam textos (por exemplo, frases, parágrafos ou documentos). Apresentado em 2014 por Mikolov [8]. Este algoritmo é uma adaptação de outro algoritmo para representação vetorial de palavras chamado **Word2vec** [7]. A característica principal destes algoritmos é (além da representação vetorial) a construção de vetores próximos para documentos (textos ou palavras) semelhantes.

As arquiteturas dos modelos *Doc2Vec* e *Word2Vec* são redes neurais, onde as entradas são os vetores *one-hot* que representam palavras de um vocabulário do modelo e as saídas são distribuições de probabilidades.

Para a estapa de pré-processamento, similar ao caso de BOW, com *Doc2vec* também vamos converter os textos do conjunto de treino em textos com letras minúsculas, definir os *tokens* e remover as *stop words*.

Antes de descrever o funcionamento da representação de texto de *Doc2Vec*, precisamos entender inicialmente o mecanismo de representação vetorial de palavras de *Word2Vec*.

2.3 Representação de palavras Word2Vec

Word2vec é um método de **Word embeddings**. *Word embeddings* significa que as palavras são representadas por vetores reais, para que possam ser tratadas como qualquer outro vetor matemático. Ou seja, é uma transformação de um texto para um espaço vetorial com as operações canônicas de soma e subtração. Para *Word2Vec*, um vetor de palavra é uma representação vetorial numérica que varia segundo o significado contextual de uma determinada palavra.

Através dos *Word embeddings* podemos processar matematicamente as palavras de um texto. Além disso, se duas palavras têm um significado semelhante, suas representações vetoriais também serão próximos, ou seja, com os *Word embeddings* temos uma forma de medir a semelhança entre palavras. Ao longo deste trabalho, veremos que é possível relacionar não apenas palavras, mas também frases e textos inteiros. Isso é muito útil em várias tarefas e sistemas de PNL, como recuperação de informações, análise de sentimento ou sistemas de recomendação.

Word2Vec foi proposto em 2013 também por T. Mikolov. Ele propôs dois modelos diferentes para *Word embeddings*, o modelo **Continuous Bag of Words (CBOW)** e o modelo **Skip-gram**. Na verdade, estes dois modelos têm o mesmo propósito, mas isso será discutido nas seguintes seções.

2.3.1 CBOW e skip-gram

A suposição básica para ambos os modelos é a hipótese distribucional: palavras que ocorrem nos mesmos contextos tendem a ter significados semelhantes. Ou seja, uma palavra é caracterizada pelas palavras vizinhas (o seu contexto). Para ambos os modelos é necessário entender a noção de contexto de uma palavra ou, também chamado de, **janela de palavras** de uma palavra referenciada.

Por exemplo, vamos considerar uma sequência de palavras:

$$w_0,\ w_1, w_2, ..., w_{N_1}, w_N$$

Assim, para a j-ésima palavra w_j, o contexo de w_j é representado da seguinte maneira:

$$\underbrace{w_0,\ w_1,\ w_2, ...,\ w_{j-1}}_{\text{contexto à esquerda}},\quad \underbrace{w_j}_{\text{palavra central}}\quad ,\ \underbrace{w_{j+1}, ...,\ w_{N-1},\ w_N}_{\text{contexto à direita}}.$$

Também pode ser representado da seguinte forma:

$$\underbrace{w_{j-m},\ w_{j-m+1}, ...,\ w_{j-1}}_{\text{contexto à esquerda}},\quad \underbrace{w_j}_{\text{palavra central}}\quad ,\ \underbrace{w_{j+1}, ...,\ w_{j+m-1},\ w_{j+m}}_{\text{contexto à direita}},$$

sendo m o **tamanho da janela** de palavras.

Em um corpus V, para cada palavra podem existir vários contextos c associados. Representaremos por $C(w)$ ao conjunto dos contextos da palavra w.

Como já foi mencionado, *Word2Vec* possui duas abordagens:

- Prever uma palavras dado um contexto: Este modelo tenta prever a i-ésima palavra, w_i, em uma frase usando uma janela de palavras de tamanho t. Ou seja, dada uma janela de palavras $w_{i-t}, w_{i-t+1}, ..., w_{i-1}, w_{i+1}, ..., w_{i+t-1}, w_{i+t}$ o algoritmo tentará prever a palavra-alvo w_i. Este modelo também é conhecido pelo nome **Continuous Bag of Word (CBOW)**.

- Prever um contexto dada uma palavra: Este modelo tenta prever o contexto $w_{i-t}, w_{i-t+1}, ..., w_{i-1}$ em torno da palavra w_i, dada a i-ésima palavra no texto, denotado por wi. Este modelo é conhecido como o modelo **skip-gram**. A seguinte imagem ilustra o diagrama de cada modelo para uma janela de palavras de tamanho 2.

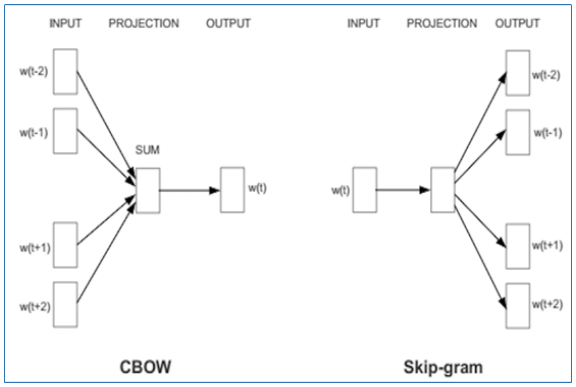

Figura 2.1: CBOW e Skip-gram. (Fonte: Tomas Mokolov et al [4])

Modelo Skip-gram

A arquitetura do modelo *skip-gram* é baseada em uma rede neural com uma única camada oculta (sem função de activação) para calcular um vetor de probabilidades (que logo descreveremos) dos vetores de saídas, com o objetivo estimar os pesos da camada oculta. Esses pesos serão os vetores de palavras (*word embedding*) que estamos procuramos.

Especificamente, a arquitetura do Skip-gram é uma rede neural artificial composta de:

- Uma camada de entrada: para o treinamento será considerado, como dado de entrada, uma palavra w no corpus V e sua representação vetorial codificada pelo vetor *one-hot*, e_i. Para cada par de dados de treinamento, a palavra de entrada da rede é a palavra central w do contexto. Não tem função de ativação nesta camada.

- Uma camada oculta: obtida pela multiplicação de uma matriz W com $|V|$ filas e N colunas aplicada a cada palavra de entrada $e_i \in \mathbf{R}^N$ para obter os valores na camada escondida h_i. Aqui a i-ésima fila de W representa o i-ésimo *embedding*, que é um vetor de tamanho N. Também não tem função de ativação nesta camada.

- Uma camada de saída: obtida pela aplicação da função *softmax* da multiplicação de uma matriz W', com N filas e $|V|$ colunas, com os valores da camada oculta h_q. Para a matriz W' a i-ésima coluna representa o *embedding* da i-ésima palavra de contexto. Ou seja, dado um *embedding* de uma palavra central h_i, a multiplicação com esta matriz retorna um vetor z, para o qual aplicaremos a função de ativação softmax em z. Assim, é gerado um vetor $\hat{y}$ de tamanho $|V|$, onde o k-ésimo elemento

$\hat{y}_k$ representa a probabilidade de que a k-ésima palavra esteja no contexto de w. A arquitetura pode ser observada na seguiunte imagem:

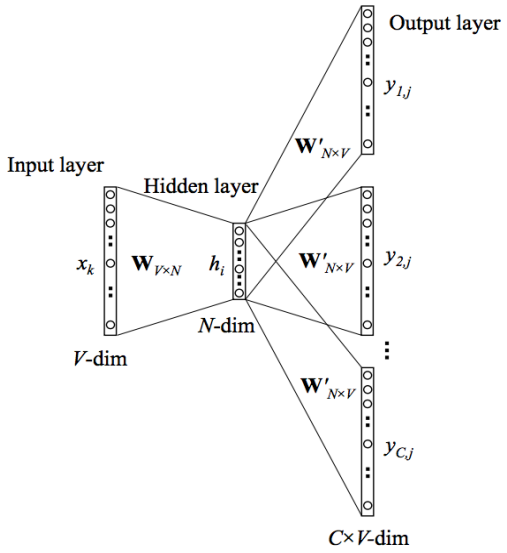

Figura 2.2: ANN Skip-Gram. (Fonte: Aggarwal [2])

Treinando o modelo Skip-gram

Agora precisamos gerar as amostrar para realizar o treinamento da rede. Para isso, seguiremos os seguintes passos:

- Passar por cada palavra de texto para fixar uma palavra central.
- Selecionar as palavras vizinhas, dentro de um determinado contexto (a janela de palavras).
- Gerar os pares de dados com cada amostra sendo composta pela palavra central e as palavras do contexto.
- Usar a representação *one-hot*, como aparece no seguinte exemplo:.

Consideremos novamente a frase **it was the best of times**. Então, para um contexto de tamanho $m = 2$, os pares de treino são:

$$(it, (was, the)) = (e_1, (1_2, e_3)),$$

$$(was, (it, the, best)) = (e_2, (e_1, e_3, e_4)),$$

$$(the, (it, was, best, \ of)) = (e_3, (e_1, e_2, e_4, e_5)),$$

$$(best, (was, the, of, times)) = (e_4, (e_2, e_3, e_5, e_6)),$$

$$(of, (the, best, times)) = (e_5, (e_3, e_4, e_6)),$$

$$(times, (best, of)) = (e_6, (e_4, e_5))$$

para o vocabulário $V = [it, \ was, \ the, \ best, \ of, \ times]$ onde cada palavra tem sua representação one-hot encoded e_j.

É importante mencionar que cada palavra de amostra no corpus V será representada inicialmente pelo vetor *one-hot* com o tamanho do corpus, ou seja, cada palavra de amostra w_i, para $i = 1, 2, ..., |V|$, será representada pelo vetor

$$\underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}}_{|V| \times 1} =: e_j$$

onde o valor é 1 na i-ésima posição do corpus V. Assim os pares de treino são da forma $(e_i, (e_{j_1}, e_{j_2}, ..., e_{j_m})$ com e_{j_k} no contexto de e_i.

Formalmente, dado uma palavra w representada em termos dos *one-hot encoded* $e_j = [x_{j1}, ..., x_{j|V|}]$, com $x_{ji} \in \{0, 1\}$, a entrada da camada oculta pode ser calculada pela fórmula

$$h_q = \sum_{j=1}^{|V|} W_{jq} x_j \ q \in \{1, ..., N\}.$$

Logo, a probabilidade de que a i-ésima palavra do contexto assume a j-ésima palavra do vocabulário é calculada pela função softmax

$$\hat{y}_{ij} = p(y_{ij} = 1|w) = \frac{exp(\sum_{q=1}^{N} h_q W'_{qj})}{\sum_{k=1}^{|V|} exp(\sum_{q=1}^{N} h_q W'_{qk})}.$$

A função de perda é o negativo do logaritmo da função de verossimilhança de $y_{ij} \in \{0, 1\}$ de uma instância de treinamento. Ou seja,

$$L = -\sum_{i=1}^{m}\sum_{j=1}^{|V|} y_{ij} \ln(\hat{y}_{ij}).$$

Assim, pelo algoritmo de *backpropagation*, podemos estimar os pesos pelas seguintes fórmulas:

$$w_{ij} \leftarrow w_{ij} - \alpha \frac{\partial L}{\partial w_{ij}}$$

$$w'_{ij} \leftarrow w'_{ij} - \alpha \frac{\partial L}{\partial w'_{ij}}$$

onde α é a razão de aprendizado (um parâmetro que devemos fixar).

Caso o leitor precise de uma rápida descrição do algoritmo *backpropagation*, pode consultar o apêndice deste trabalho monográfico para uma breve introdução ao método, na página 31.

Modelo Continuous bag of word (CBOW)

O modelo *Continuous bag of word* (CBOW) segue um processor análogo, mas inverso, a *Skip-gram*, ou seja, dada uma janela de palavras, o modelo prevê uma palavra central w a partir desse contexto. Na figura 2.3 podemos ver um exemplo para a frase *The cat sat on*. Aqui temos as palavras de contexto $w_1 = The$, $w_2 = cat$, $w_3 = sat$ e a palavra central $w = on$.

Similar a *Skip-gram*, CBOW também treina uma rede neural com uma camada de entrada, uma camada de saída com função de ativação softmax e uma camada escondida sem função de ativação. A única diferença em CBOW é que agora será calculada a soma das camadas escondidas. Podemos observar a semelhança da arquitectura de CBOW com *Skip-gram* na figura 2.4:

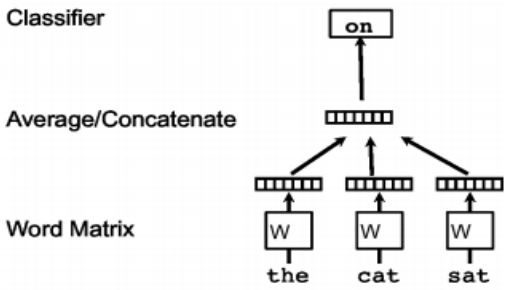

Figura 2.3: Continuous bag of word (Fonte: Thomas Mikolov et al [4])

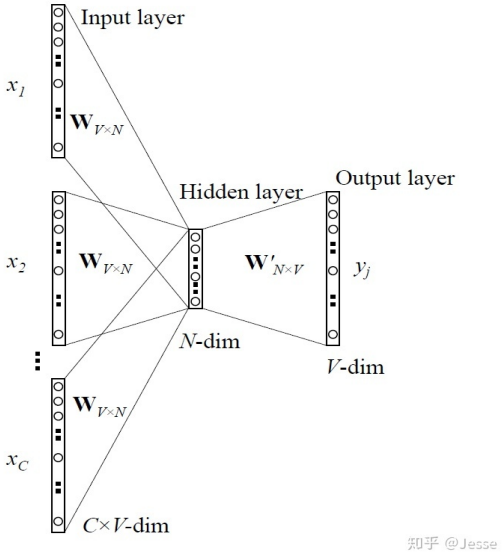

Figura 2.4: Continuous bag of word (Fonte: Aggarwal [2])

Como já foi mencionado, para este modelo temos que as saídas da camada escondida, h_q, podem ser calculadas pela seguinte fórmula:

$$h_q = (x_1 + ... + x_m)^T W_q = \sum_{i=1}^{m} \sum_{j=1}^{|V|} W_{jq} x_{ij}$$

Assim, a probabilidade de que a palavra central seja a j-ésima palavra do corpus pode ser calculada pela função softmax:

$$\hat{y}_j = p(y_j = 1|w_1...w_m) = Softmax((x_1 + ... + x_m)^T W W'_j) =$$

$$= \frac{exp(\sum_{q=1}^{N} h_q W'_{qj})}{\sum_{k=1}^{|V|} exp(\sum_{q=1}^{N} h_q W'_{qk})}.$$

Para este caso, a função de perda L é definida por

$$L = -\ln(p(y_j = 1|w_1...w_m)) = -\ln(\hat{y}_j)$$

Similar a *Skip-gram*, também podemos estimar os pesos W_{ij} e W'_{ij} pelas fórmulas do algoritmo *backpropagatio*:

$$w_{ij} \leftarrow w_{ij} - \alpha \frac{\partial L}{\partial w_{ij}}$$

$$w'_{ij} \leftarrow w'_{ij} - \alpha \frac{\partial L}{\partial w'_{ij}}$$

onde α é a razão de aprendizado (um parâmetro que devemos fixar).

O conjunto de treino de CBOW é parecido ao cojunto de treino de *Skip-gram* mas invertendo a ordem dos pares de treino. Seguindo o exemplo de *skip-gram*, dada a frase **it was the best of times**, para um contexto de tamanho $m = 2$, os pares de treino são da forma:

$$((was,the), it) = ((e_2, e_3), e_1),$$

$$((it,the,best), was) = ((e_1, e_3, e_4), e_2),$$

$$((it,was,best,\ of), the) = ((e_2, e_3, e_4, e_5), e_3),$$

$$((was,the,of,\ times), best) = ((e_2, e_3, e_5, e_6), e_4),$$

$$((the,\ best,times), of) = ((e_3, e_4, e_6), e_5),$$

$$((best,of), times) = ((e_4, e_5), e_6)$$

para o vocabulário $V = [it,\ was,\ the,\ best,\ of,\ times]$ com cada palavra representada pelo vetor *one-hot encoded* correspondente.

2.4 Por quê word2vec?

Agora que já foi descrito o funcionamento de *word2vec* e a suas arquiteturas, vamos explicar a motivação destas arquiteturas de redes neurais para representar vetorialmente palavras de um texto, ou seja, vamos discutir uma motivação ao método Word2vec.

Consideremos um texto com seu respectivo corpus de palavras V. Consideremos também uma sequência de palavras de contexto $w_1, ..., w_m$ de uma determinada palavra w. Vamos denotar as representações em vetores one-hot das palavras $w_1, ..., w_m$ e w por $x_1, ..., x_m \in \mathbb{R}^{|V|}$ e y, respectivamente. Sabemos que com esta representação não estamos considerando nenhuma relação entre a palavra w com seu contexto $w_1, ..., w_m$. Então, para poder construir um conjunto de vetores que sim consiga extrair a relação da palavra w com seu contexto, precisamos transformar esses vetores one-hot em outros vetores $h_1, ..., h_m$ que serão chamados de embeddings.

Uma tentativa de transformar estes vetores one-hot em vetores embeddings é através de matrizes $W \in M_{|V|\times N}(R)$ e $W' \in M_{N\times|V|}(R)$, onde $|V|$ é o tamanho do vocabulário e N é um inteiro fixo, as quais serão aplicadas aos vetores de one-hot x_i e y, respectivamente. Ou seja, definimos $h_i = x_i^T W \in \mathbb{R}^N$ e $W'y$ de tal modo que, para esses vetores h_i, a 'chance' de conter a palavra $W'y$ seja 'muito alta' (pois é a frequência de ter essa palavra entre $w_1, ..., w_m$ é alta). A melhor maneira de interpretar esta 'chance' é através de uma função de probabilidade p.

O problema agora é estabelecer um método para construir essas matrizes W, W' e a função de probabilidade p com tais condições, ou seja, que a probabilidade $p(w_{out} = w|w_1...w_m) = p(W'y|h_1...h_m)$ seja alta. Uma possível solução é selecionar uma função de probabilidade p (a softmax da soma dos h_i, por exemplo) e estimar os coeficientes das matrizes W w W' de tal maneira que a probabilidade $p(w_{out} = w|w_1...w_m) = p(W'y|h_1...h_m)$ seja máxima. Para isso podemos usar um algoritmo de otimização, por exemplo, o método do gradiente descendente (o mesmo método usado em backpropagation). Podemos ilustrar a ideia no seguinte diagrama:

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \rightarrow \begin{bmatrix} W \\ W \\ \vdots \\ W \end{bmatrix} \rightarrow \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_N \end{bmatrix} \rightarrow \sum_q h_q \rightarrow \Big[\ W'\Big]_{N\times|V|} \rightarrow \text{Softmax} \rightarrow\ \hat{y}$$

Podemos observar como esse diagrama relembra a arquitetura CBOW de *Word2vec*. Com um argumento análogo, também podemos motivar a arquitetura de *Skip-gram*.

Agora, uma vez entendido o algoritmo de *Word2Vec*, podemos estender esta ideia para implementar o algoritmo de representação vetorial de texto *Doc2Vec*.

2.5 Doc2Vec

Doc2vec utiliza arquitecturas similares às arquitecturas de Word2vec, a diferença agora é a adição de um campo novo referente ao texto que queremos representar numericamente. Este campo é chamado de **Paragraph ID**. Para este campo, será necessário associar uma *tag* exclusiva para cada documento.

Para *Word2Vec* vimos que as duas arquiteturas utilizadas são *Skip-gram* e *Continuous Bag Of Word*. Para *Doc2vec*, os algoritmos correspondentes são **distributed memory version of Paragraph Vector (PV-DM)** e **distributed bag of words of Paragraph Vector (PV-DBOW)**.

PV-DBOW é o modelo de *doc2vec* análogo ao modelo *Skip-gram* de *word2vec*. Os vetores de parágrafos são obtidos pelo treinamento de uma rede neural com o propósito de construir uma distribuição de probabilidade de palavras em um parágrafo dada uma palavra aleatoriamente selecionada.

Vamos descrever o funcionamento da arquitetura PV-DM que a qual, segundo Mikolov [5], oferece um melhor desempenho. Para isso, vamos considerar um corpus D com C documentos $D = \{d_1, d_2, ..., d_C\}$ e N tokens únicos extraídos do corpus D. Os N tokens formarão nosso dicionário. Para cada $d_i \in D$ vamos construir o vetor *one-hot*, formando uma matriz vetorial de contagem M de tamanho $C \times N$, onde cada fila da matriz M contém a frequência de *tokens* no documento d_i. Agora, com esta representação, podemos treinar o modelo por meio da arquitetura CBOW de *Word2vec*. Aqui, em vez de usar apenas palavras para prever a palavra central, também adicionar um documento no campo Paragraph ID. Depois de treinar a rede CBOW, multiplicamos a matriz de pesso W, associada ao documento d_i, com a matriz M de d_i. Este produto será o *embeddig* do documento d_i. Podemos ter uma ideia do mecanismo de PV-DM através da Figura 2.5.

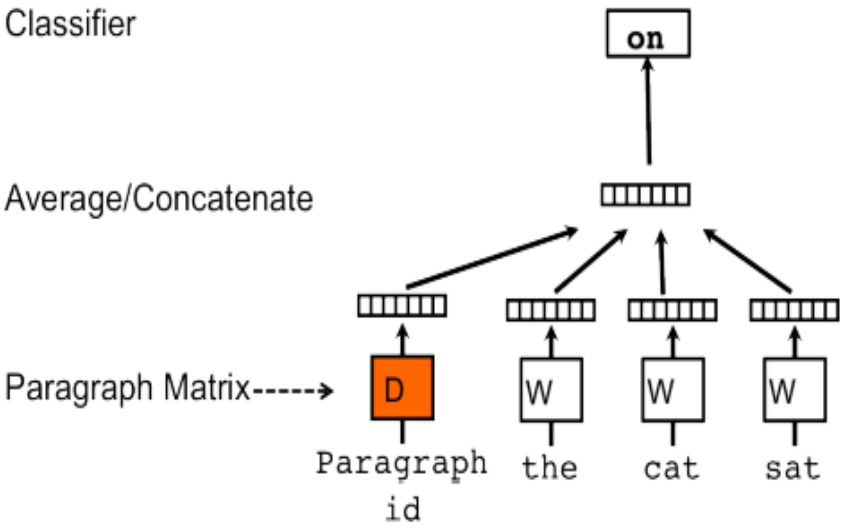

Figura 2.5: PV-DM. (Fonte: Thomas Mikolov et al [5])

Capítulo 3

Resultados

Neste capítulo vamos destacar algumas informações extraídas pelos métodos *Doc2vec* e *Bag of Word* referentes ao banco de dados mencionado no início do capítulo 2. Este banco de dados será descrito com mais detalhe na seguinte seção. Também vamos descrever a implementação computacional dos classificadores selecionados e seus respectivos parâmetros. Finalmente, serão comparados os desempenhos de cada classificador utilizando como dados de entrada as duas representações vetoriais.

3.1 Descrição do banco de dados

Como foi mencionado no início do capítulo 2, vamos utilizar um banco de dados de um e-commerce de roupa feminina com 8344 registros de avaliações escritas pelos clientes. Para cada opinião existe um rótulo que classifica se a opinião é considerada positiva ou negativa. A categoria **Sim** será representada pelo número 1 e a categoria **Não** será representada pelo número 0.

Para este banco de dados, utilizaremos uma proporção de 70 % dos 8344 dados para treinar os classificadores. Ou seja, 5840 registros serão utilizados como dados de treinamento para cada classificador e 2504 registros para dados de teste. Com relação à proporção de rótulos 0 e 1, o banco de dados dispõe de um número balanceado de registros, ou seja, a metade dos registros (4172 textos) estão classificados com o rótulo 0 (categoria **Não**) e a outra metade dos registros estão classificados com o rótulo 1 (categoria **Sim**).

Como pré-processamento, vamos eliminar as palavras mais frequentes que não contém informação importante dos textos, ou seja, vamos remover as stop words. A lista de

stop word podem ser consultadas na tabela do anexo.

Para a implementação computacional utilizaremos o programa Python e importaremos varias das suas bibliotecas. Com o pacote *Pandas* manipularemos os *data frames*, com o submódulo *preprocessing* do pacote *gensim* será realizado o trabalho de pré-processamento dos textos e com o pacote *nltk* será construido o vocabulário (tokens).

Agora vamos destacar algumas informações extraídas pelo método *Doc2vec*:

3.1.1 Representação Doc2vec do banco de dados

Importando o submódulo *doc2vec* do pacote *gensim* podemos implementar a representação vetorial dos textos. Aqui estamos interessados em construir vetores de dimensão 100 com a arquitetura PV-DM.

Com este submódulo podemos extrair conjuntos de textos semelhantes (contextos parecidos), por exemplo, os textos 169 e 4586 (ambos com rótulo 0):

$$\text{texto}[169] = \textit{Scratchy The sweater is cute, but scratchy.}$$

$$\text{texto}[4586] = \textit{Scratchy Cute but too thick and rough.}$$

Também permite listar as palavras semelhantes de uma determinada palavra, por exemplo, temos as palavras *cute* e *dress* com as seguintes palavras semelhantes:

cute	adorable	dress	sweater
	beautiful		shirt
	great		jumpsuit
	love		top
	lovely		romper
	excited		blouse
	nice		pants

Tabela 3.1: Palavras semelhantes a *cut* e *dress* extraídas por *Doc2vec*

3.1.2 Representação Bag of Word do banco de dados

Para a representação vetorial de tamanho 100 pelo método de *Bag of Word*, vamos considerar um vocabulário com as 100 palavras mais informativas do corpus de documen-

tos. A metodologia utilizada para extrair estas palavras pode ser consultada no artigo de Jessica Kubrusly, Ana Luiza Neves e Thamires Louzada Marques [5]. A continuação, apresentamos a tabela com o vocabulário das 100 palavras selecionadas:

skirt	tall	wear	cheap	disappoint
fabric	feel	fit	material	tight
bust	purchase	shape	fall	front
jeans	medium	wide	button	huge
light	look	nice	short	size
sleeve	sweater	true	quality	sale
super	thin	comfy	perfect	comfortable
cute	store	summer	wash	tank
cut	beautiful	body	length	lovely
online	perfectly	regular	retailer	soft
neck	shoulder	bottom	boxy	flatter
gorgeous	pattern	arm	casual	lb
blue	buy	hope	loose	model
photo	recommend	xs	piece	design
day	picture	receive	stretch	unflattering
price	black	chest	bite	hip
line	tee	detail	bad	expect
pair	pant	reviewer	waist	sheer
low	hang	hit	jacket	review
color	lot	love	petite	return

Tabela 3.2: Vocabulário com as 100 palavras mais informativas do corpus

Medida de contagem

O método de contagem de presença de palavras será dada pela frequência de cada palavra do vocabulário em cada texto. Uma vez realizadas as representações vetoriais de tamanho 100, vamos selecionar os 6 classificadores para avaliar seus desempenhos utilizando cada representação.

3.2 Classificadores selecionados

Na seguinte seção vamos avaliar e comparar os desempenho dos seguintes classificadores selecionados:

- **Regressão Logística**
- **Árvore de decisão**
- **Random Forest**
- **K Nearest Neighbour** (KNN)
- **Artificial Neural Network** (ANN)
- **Support Vector Machine** (SVM)

Para o leitor interessado em estudar cada um destes algoritmos de classificação, é formtemente recomendado a consulta do livro de Kevin Murphy [6].

3.3 Desempenho dos classificadores usando BOW e Doc2Vec

Para avaliar a diferença entre os métodos *Bag Of Word* e *Doc2Vec*, foi avaliado o desempenho de cada algoritmo de classificação selecionado utilizando cada tipo de representação (BOW e *Doc2Vec*). Os resultados são apresentados por meio das matrizes de confusão e das acurácias, acompanhadas dos tempos de processamento nas seguintes tabelas.

3.3.1 Tabelas de comparação dos resultados

Resultados com Regressão Logística

A Tabela 3.3 apresenta os desempenhos do algoritmo de classificação *Regressão Logística* para cada representação vetorial. Para implementar este algoritmo, foi importado o submódulo *LinearRegression* do pacote *sklearn*, com o parâmetro *solver = 'lbfgs'* para indicar o método de Broyden-Fletcher-Goldfarb-Shanno como método de otimização.

Doc2Vec	1	0
1	1007	245
0	209	1043
Acurácia	0.8187	
Tempo de proc.	0.809 s	

BOW	1	0
1	937	291
0	940	272
Acurácia	0.4955	
Tempo de proc.	0.607 s	

Tabela 3.3: Regressão Logística

Resultados com Árvore de decisão

A Tabela 3.4 apresenta os desempenhos do algoritmo de classificação *Árvore de decisão* para cada representação vetorial. Para implementar este algoritmo, foi importado o submódulo *DecitionTreeClassifier* do pacote *sklearn* com o parâmetro *criterion = 'gini'* para usar a informação de Gini. Por default, o número mínimo de ramos é 2 e o número mínimo de folhas é 1.

Doc2Vec	1	0
1	764	488
0	506	746
Acurácia	0.5774	
Tempo de proc.	1.39 s	

BOW	1	0
1	863	365
0	788	424
Acurácia	0.5274	
Tempo de proc.	0.734 s	

Tabela 3.4: Árvore de decisão

Resultados com Random Forest

A Tabela 3.5 apresenta os desempenhos do algoritmo de classificação *Random Forest* para cada representação vetorial. Para implementar este algoritmo, foi importado o submódulo *RandomForestClassifier* do pacote *sklearn* com o parâmetro *n-estimators* = 100, para usar 100 árvores de decisão.

Doc2Vec	1	0
1	1031	221
0	427	825
Acurácia	0.7412	
Tempo de proc.	2.078 s	

BOW	1	0
1	1030	198
0	1025	187
Acurácia	0.4988	
Tempo de proc.	0.322 s	

Tabela 3.5: Random forest

Resultados com K Nearest Neighbour

A Tabela 3.6 apresenta os desempenhos do algoritmo de classificação *K Nearest Neighbour* (KNN) para cada representação vetorial. Para implementar este algoritmo, foi importado o *KNeighborsClassifier* do pacote *sklearn* com o parâmetro *n-neighbors* = 7, para indicar que devem ser escolhido os 7 vizinhos mais pertos. Por default, a distância é definida pela norma L2.

Doc2Vec	1	0
1	729	523
0	249	1003
Acurácia	0.6917	
Tempo de proc.	1.020 s	

BOW	1	0
1	924	304
0	933	279
Acurácia	0.493	
Tempo de proc.	0.823 s	

Tabela 3.6: KNN

Resultados com Artificial Neural Network

A Tabela 3.7 apresenta os desempenhos do algoritmo de classificação *Artificial Neural Network* (ANN) para cada representação vetorial. Para implementar este algoritmo, foi importado o *MLPClassifier* do pacote *sklearn* com o parâmetro *max-iter* = 10000 para realizar 10000 iterações e o parâmetro *hidden-layer-sizes = (50,50)* para indicar que a rede deve conter duas camadas escondidas de tamanho 50. Por default, a função de ativação nas camadas escondidas é a função *ReLU*.

Doc2Vec	1	0
1	926	326
0	292	960
Acurácia	0.7532	
Tempo de proc.	23.134 s	

BOW	1	0
1	895	333
0	873	339
Acurácia	0.5057	
Tempo de proc.	12.011 s	

Tabela 3.7: ANN

Resultados com Support Vector Machine

A Tabela 3.8 apresenta os desempenhos do algoritmo de classificação **Support Vector Machine** (SVM) para cada representação vetorial. Para implementar este algoritmo, foi importado o submódulo *SVC* do pacote *sklearn*. Indicamos o parâmetro da margem de erro $C = 2$. Por default, o kernel é a função **Radial Basis** (RBF).

Doc2Vec	1	0
1	952	300
0	195	1057
Acurácia	0.8023	
Tempo de proc.	4.68 s	

BOW	1	0
1	991	237
0	976	236
Acurácia	0.5028	
Tempo de proc.	4.002 s	

Tabela 3.8: SVM

Podemos observar diretamente das tabelas que os classificadores escolhidos registraram melhores desempenhos de acurácia quando utilizaram como entrada de dados a representação vetorial de texto de *Doc2Vec*. Sendo os algoritmos *Support Vector Machine* (SVM) e a *Regressão Logística* os classificadores com melhores desempenhos, superando o 80% de acurácia (80,23% e 81,87%, respectivamente). Com *Random Forest* e a Rede Neural Artificial de duas camadas também foram registadas acurácias superiores ao 50% da acurácia registrada pelo método *Bag of Word*. O algoritmo *Árvore de Decisão* foi o único classificador com desempenho similar ao desempenho registrado por *Bag of word* (57,74% usando *Doc2Vec* e 52,54% usando *Bag of Word*).

Por outro lado, com relação ao tempo de processamento (treinamento e predição) de cada classificador utilizando o método de *Doc2vec*, podemos observar que, entre os classificadores com maior acurácia, o algoritmo de *Regressão Logística* teve o menor registro, com 0.809 segundos, inferior ao tempo de processamento da Rede Neural Artificial de duas camadas (com 23.134 segundos) e do registro do classificador *Support Vector Machine* (com 4.68 segundos).

Com estas estatísticas de desempenho de cada classificador, utilizando os métodos de representação vetorial *Bag of Word* e *Doc2vec*, para problemas de classificação de texto, onde cada palavra é caracterizada pelo contexto, a concatenação do algoritmo de Regressão Logística com a representação Doc2vec apresenta duas grandes vantagens com relação aos outros classificadores selecionados: acurácia alta e baixo tempo de processamento. Além disso, com o algoritmo de *Regressão Logística* também poderíamos obter alguma interpretabilidade nos resultados da classificação, o que nos permitiria realizar análises estatísticos muito mais aprofundados.

Capítulo 4

Conclusão

Comparando os resultados dos classificadores utilizando o método de *Doc2Vec* (*Word2Vec*) com os resultados obtidos pelo método *Bag of Word*, podemos concluir que os algoritmos de representação vetorial que consideram fortemente as relações (semânticas, morfológicas, etc) entre as palavras com seu contexto podem aumentar significativamente, em muito dos casos, o desempenho do classificador para problemas de classificação de textos. Estas relações podem conter informações relevantes sobre cada texto e algoritmos como *Doc2Vec* consiguem representar estas informações em forma de vetores numéricos. Este tipo de representações vetoriais apresentaram evidências de facilitar o processamento da classificação e, ao mesmo tempo, de otimizar a acurácia do classificador.

Para futuros projetos, serão implementados métodos estatísticos mais sofísticados e gráficos mais apropriados para estudar e entender geometricamente a representação vetorial de *Doc2Vec* (*Word2Vec*) e também para ajustar melhor os parâmetros dos classificadores de tal modo que a acurácia seja máxima. Com este objetivo de otimizar a acurácia, também serão exploradas algumas variantes do algoritmo de *Word2Vec* e *Doc2Vec*, por exemplo, utilizando funções de ativação na camada oculta tanto na arquitetura *CBOW* quanto na arquitetura *Skip-gram*.

Capítulo 5

Tópicos para trabalhos futuros

O método de *Word2Vec* é uma das abordagens mais utilizadas para representação vetorial de texto, mas também é importante mencionar que existem outros métodos alternativos que podem melhorar o desempenho obtido pelo classificador usando *Doc2Vec*. Na seguinte seção dedicamos um espaço para comentar alguns artigos e tópicos de interesse que poderiam servir como ponto de partida para futuras pesquisas sobre métodos de classificação de texto mais sofisticados.

5.1 Artigos e tópicos

Em 2018, Aditya Siddhant and Zachary C. Lipton [1] apresentam um artigo onde destacam o forte desempenho dos algoritmos de Active learning (AL) baseado em redes neurais profundas para problemas de classificação de texto. São mencionados os primeiros artigos onde são utilizados estes algoritmos, mostrando resultados promissores. Neste artigo é estabelecida uma taxonomia para classificar as estratégias de consultas (query's trategies) relevantes para o processo de AL para o problema de classificação de texto. Além disso, são citados artigos com avanços recentes em classificação de texto baseados em redes neurais e da representação vetorial das palavras (word embeddings) considerando as relações semânticas entre elas. Por exemplo, os métodos Wor2vec [4], GloVe [2] e fastText.

Em 2014, Yoon Kim [10] apresenta um artigo que é a referência principal para problemas de classificação de texto baseado em redes neurais de convolução. Este método consiste em pré-treinar as palavras usando Word2Vec e inicializar uma matriz bidimensional para a camada de entrada de tamanho $n \times d$, onde n denota o número de palavras

em uma frase, e d representa a dimensão do vetor de palavra. Logo, é aplicado um max pooling para a o vetor de palavra de entrada através de vários núcleos de convolução. A função de perda utilizada para treinar este modelo é a entropia cruzada.

Também em 2014, Jeffrey Pennington, Richard Socher e Christopher Manning apresentam o famoso método de representação de texto, **GloVe** [4]. Neste artigo é apresentado um algoritmo não supervisionado para representar palavras em forma vetorial. Segue o mesmo paradigma de Word2Vec com a diferença de que otimiza ainda mais a relação entre as palavras estabelecida pela semântica entre elas. Vários métodos de classificação de texto são implementados com a representação GloVe, mostrando um melhor desempenho com relação a Word2Vec.

Em 2017, os pesquisadores Ye Zhang, Matthew Lease e Byron C. Wallace [9] apresentam métodos de AL baseado em CNN para a classificação de texto. Estes métodos seguem a ideia do algoritmo de Yoon Kim [10], mas associando uma função objetivo para cada tipo de corpus a ser classificado (documentos ou frases). Todos estes métodos apresentaram um alto desempenho com relação à acurácia.

Em 2020, Zhang X., Haitao Wang, Jie He e Shufen Liu [11] apresentam um método de classificação de texto integrando os N-gram com Redes Neurais de Convolução (CNN) para problemas de classificação de texto curto. Este método também está baseado no artigo de Yoon Kim [10]. Aqui os N-gramas são usados para filtrar a matriz de texto inicial e em seguida continuar com a CNN. Para vários datasets, os modelos 4-gram-ACNN apresentaram melhoras com relação à precisão da classificação.

Capítulo 6

Apêndice

6.1 Backpropagation

A continuação vamos dar uma breve descrição do algoritmo de *Backpropagation* de uma rede neural artificial. Vamos considerar uma rede neural artificial, com k valores de entrada $x_1, ..., x_k$, k pesos $w_1, ...w_k$ e um valor escalar de saída y. Aqui f é a função de ativação da camada oculta, como podemos observar na figura 6.1

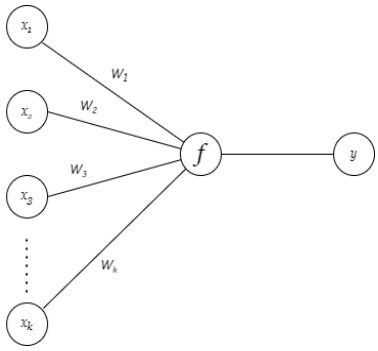

Figura 6.1: Rede Neural cim uma camada oculta e um nó de saída

Esta rede neural artificial representa a seguinte operação

$$y = f(u),$$

onde u é o valor escalar da camada oculta definido pelo produto interno dos vetores $w = [w_1, ..., w_k]^T$ e $x = [x_1, ..., x_k]$, ou seja,

$$u = \sum_{i=1}^{k} w_i x_i = w^T x.$$

Para cada função de ativação f podemos modelar um comportamento diferente da rede neural. Entre as funções de ativação mais usadas temos a **função logística** (ou **sigmoide**)

$$f(u) = \frac{1}{1+e^{-u}};$$

também temos a função tangente hiperbólica

$$f(u) = tanh(u) = \frac{1-e^{-u}}{1+e^{-u}};$$

a função linear

$$f(u) = mu,$$

entre outras.

Também são usadas funções de ativação não diferenciáveis, por exemplo, a função **RELU**, definida por $f(u) = max(0, u)$ e a função indicadora (ou **Step function** em inglês) definida por $f(u) = I_{[0,+\infty)}(u)$.

6.1.1 Treinamento por backpropagation

Vamos considerar o seguinte problema: Seja um conjunto de pares de amostras $D = \{(x_1^{(i)}, ..., x_k^{(i)}, y^{(i)}) \in \mathbf{R}^{k+1} : i = 1, ..., n\}$ tais que essas amostras obedecem um determinado modelo teórico

$$y^{(i)} = f\left(\sum_{j=1}^{k} w_j x_j^{(i)}\right) + \epsilon_i = f(w^T x^{(i)}) + \epsilon_i,$$

para $i = 1, 2, ..., n$.

Queremos estimar os pesos $w_1, ..., w_k$ de tal maneira que o modelo ajustado

$$\hat{y}^{(i)} = f\left(\sum_{j=1}^{k} \hat{w}_j x_j^{(i)}\right)$$

minimiza o valor $L(y, \hat{y}) = L([y^{(1)}, ..., y^{(n)}], [\hat{y}^{(1)}, ..., \hat{y}^{(n)}])$ para alguma **função objetivo** L selecionada.

Observamos que este problema pode ser representado graficamente pela rede neural artificial ilustrada na figura 6.1. Um método para estimar estes pesos w_i é conhecido pelo algoritmo de **backpropagation**.

A ideia deste algoritmo é a seguinte: dado um número de iterações r e valores iniciais para os pesos $\hat{w}_1, ..., \hat{w}_k$, vamos atribuir o valor

$$\hat{w}_i - \alpha \frac{\partial L}{\partial \hat{w}_i}(y, \hat{y})$$

no valor atual de $\hat{w}_i$ ou, equivalentemente, para o vetor $\hat{w} = [\hat{w}_1, ..., \hat{w}_k]$, atribuir o valor da operação

$$\hat{w} - \alpha \nabla_{\hat{w}} L(y, \hat{y})$$

onde α é a razão de aprendizado e $\nabla_{\hat{w}} L(y, \hat{y})$ é o vetor gradiente da função L. Este processo de atribuição (ou atualização) dos pesos deve ser realizado r vezes.

Formalmente, em pseudocódigo, temos o seguinte algoritmo:

Algorithm 1 Backpropagation

Input: iteracoes, α, $D = \{(x^{(i)}, y^{(i)}) \in \mathbf{R}^{k+1} : i = 1, ..., n\}$

Output: vetor de pesos $\hat{w}$

Inicializar vetor de pesos $\hat{w}$

for i = 1, ..., iteracoes **do**

$\hat{y} \leftarrow [f(\hat{w}^T x^{(1)}), ..., f(\hat{w}^T x^{(n)})]$

$\hat{w} \leftarrow \hat{w} - \alpha \nabla_{\hat{w}} L(y, \hat{y})$

end for

Como exemplo, vamos descrever este processo utilizando a função de ativação sigmoide

$$f(u) = \frac{1}{1 + e^{-u}}$$

e afunção objetivo $L(\hat{y}, y) = \frac{1}{2}||\hat{y} - y||^2$.

Por contas diretas, observamos que

$$f'(u) = f(u)f(-u) = f(u)(1 - f(u)).$$

Logo, usando regra da cadeia, temos

$$\frac{\partial L}{\partial \hat{w}_i} = \nabla_{\hat{y}} L.[f'(w^T x^{(1)}), ..., f'(w^T x^{(n)}))] \frac{\partial u}{\partial \hat{w}_i} =$$

$$= (\hat{y} - y).(\hat{y}(1 - \hat{y}))x_i.$$

Então

$$\nabla_{\hat{w}} L(y, \hat{y}) = (\hat{y} - y).\hat{y}(1 - \hat{y}).x$$

Assim, para um número fixo de iterações, r, um valor $\alpha > 0$ e o vetor de pesos inicializado $\hat{w}$, iteramos r vezes o processo de atualização dos pesos por meio das seguintes atribuições:

$$\hat{y} \leftarrow [f(\hat{w}^T x^{(1)}), ..., f(\hat{w}^T x^{(n)})]$$

$$w \leftarrow w - \alpha(\hat{y} - y).\hat{y}(1 - \hat{y}).x$$

Backpropagation com multiplas saídas

No caso de uma rede neural com vários nós de saídas $y = [y_1, ..., y_l]$, k entradas $x = [x_1, ..., x_k]$ e uma camada oculta com m nós, teremos duas matrizes de pesos $[w_{ij}]$ e $[w'_{ij}]$ como mostra na imagem 6.2

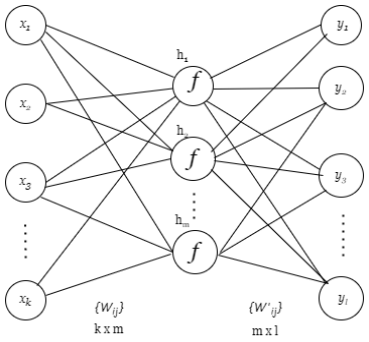

Figura 6.2: Rede Neural com uma camada oculta e vários nós de saída

Para atualizar os pesos w_{ij} e w'_{ij} podemos fixar um nó de saída, por exemplo y_3, um nó da camada oculta h_j e treinar o vetor de pesos $w_j = [w_{ij}, ..., w_{kj}]$, similar ao caso com um nó de saída. Na imagem 6.3 podemos observar que temos o mesmo caso inicial

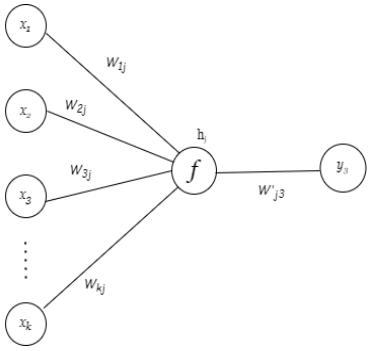

Figura 6.3: Rede Neural com uma camada oculta e vários nós de saída

Assim, podemos aplicar o algoritmo de backpropagation para fixando outra nó de saída y_k e qualquer outro nó da camada aculta h_j para atualizar todos os pesos w_{ij} e w'_{ij}.

Para mais detalhes e teoria mais aprofundada sobre redes neurais, o leitor pode consular o livro [3] dos autores Ian J. Goodfellow, Yoshua Bengio e Aaron Courville.

Capítulo 7

Anexo: Tabelas de Stop Words

put	wherein	though	re	within	whereafter
above	hundred	mine	upon	meanwhile	become
nor	latter	been	of	over	almost
take	along	when	call	ours	after
co	bottom	from	hers	due	more
other	rather	nobody	on	whereas	somehow
six	whom	interest	least	noone	hereby
among	up	because	yourselves	around	his
to	using	eight	see	nine	only
every	fifteen	you	whole	find	perhaps
fify	without	eg	part	off	thereafter
although	per	while	others	doesn	could
empty	anyhow	all	below	twenty	alone
then	regarding	everyone	yet	there	behind
thus	moreover	say	should	yourself	some
may	else	why	beforehand	cry	few
with	please	whereupon	one	herself	onto
latterly	de	nowhere	eleven	very	through
them	made	amongst	be	their	afterwards
still	how	fire	etc	is	namely

Tabela 7.1: Stop Words

someone	being	beside	against	make	full
couldnt	forty	former	toward	hence	anyway
bill	any	has	into	would	last
that	even	fill	once	mostly	via
sixty	ltd	amount	thin	whence	will
she	formerly	my	during	and	cannot
km	ten	doing	something	who	four
until	as	sometimes	sincere	becomes	less
did	by	each	here	across	before
at	next	third	whoever	its	enough
down	between	about	had	ever	several
serious	now	really	since	were	both
such	those	became	him	again	myself
they	an	does	therein	nevertheless	except
three	your	under	kg	thence	a
also	which	unless	yours	us	seems
side	itself	these	show	already	seem
detail	always	inc	seemed	whether	her
than	seeming	none	whither	so	anyone
besides	this	our	much	too	themselves

Tabela 7.2: Stop Words

whenever	con	he	didn	indeed
anything	becoming	first	the	elsewhere
back	computer	in	me	or
thru	top	don	have	front
various	cant	sometime	otherwise	just
never	hereupon	thick	found	wherever
hasnt	used	whatever	five	move
whereby	was	un	thereupon	therefore
amoungst	either	done	however	twelve
go	what	many	it	must
beyond	two	out	thereby	everywhere
hereafter	we	same	can	get
keep	am	name	well	ie
further	somewhere	system	together	are
for	i	anywhere	if	become
give	herein	where	nothing	another
most	towards	describe	quite	everything
mill	himself	do	whose	neither
own	ourselves	often	might	throughout

Tabela 7.3: Stop Words

Referências Bibliográficas

[1] Aditya Siddhant and Zachary C. Lipton. Deep Bayesian Active Learning for Natural Language Processing: Results of a Large-Scale Empirical Study. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, 2018, pp. 2904 - 2909.

[2] Charu C. Aggarwal. Machine Learning for Text, Springer, March 2018.

[3] Ian J. Goodfellow, Yoshua Bengio and Aaron Courville. Deep Learning. MIT Press, 2016.

[4] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global Vectors for Word Representation. en. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics, 2014, pp. 1532 - 1543.

[5] Jessica Kubrusly, Ana Luiza Neves, Thamires Louzada Marques. A Statistical Analysis of Textual E-Commerce Reviews Using Tree-Based Methods. Open Journal of Statistics, 2022, 12, 357-372

[6] Kevin P. Murphy. Probabilistic Machine Learning: An introduction. MIT Press, 2022.

[7] Tomas Mikolov et al. Efficient Estimation of Word Representations in Vector Space. In: 1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings. 2013.

[8] Tomas Mikolov, Quoc Le. Distributed Representations of Sentences and Documents. en. in: Proceedings of the 31st International Conference on Machine Learning, PMLR 32(2):1188-1196, 2014.

[9] Ye Zhang, Matthew Lease, and Byron C. Wallace. Active Discriminative Text Representation Learning. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence. AAAI-17. AAAI Press, 2017, pp. 3386 - 3392.

[10] Yoon Kim. Convolutional Neural Networks for Sentence Classification. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics, 2014, pp. 1746?1751. URL: https://www.aclweb.org/anthology/D14-1181.

[11] Zhang X., Haitao Wang, Jie He, Shufen Liu. A Short Text Classification Method Based on N-Gram and CNN. March 2020Chinese Journal of Electronics 29(2):248-254