Pontifícia Universidade Católica
do Rio de Janeiro

David Evandro Amorim Martins

Segmentação semântica de vagas de emprego: estudo comparativo de algoritmos clássicos de aprendizado de máquina

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática da PUC-Rio.

Orientador: Prof. Eduardo Sany Laber

Rio de Janeiro
Maio de 2020

Pontifícia Universidade Católica
do Rio de Janeiro

David Evandro Amorim Martins

Segmentação semântica de vagas de emprego: estudo comparativo de algoritmos clássicos de aprendizado de máquina

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática da PUC-Rio. Aprovada pela Comissão Examinadora abaixo.

Prof. Eduardo Sany Laber
Orientador
Departamento de Informática – PUC-Rio

Profa. Aline Medeiros Saetler
Pesquisador Autônomo – Jobzi

Prof. Hélio Côrtes Vieira Lopes
Departamento de Informática – PUC-Rio

Rio de Janeiro, 12 de Maio de 2020

David Evandro Amorim Martins

Graduou-se em Engenharia de Computação pelo Instituto Tecnológico de Aeronáutica (ITA). Fez Curso de Extensão em Engenharia de Armamento Aéreo no Instituto Tecnológico de Aeronáutica (ITA). Fez Curso de Formação em Engenharia de Petróleo na Petrobras. Trabalha como Engenheiro de Petróleo na Petrobras com ênfase em ciência de dados e métodos estatísticos.

Ficha Catalográfica

Martins, David Evandro Amorim

Segmentação semântica de vagas de emprego: estudo comparativo de algoritmos clássicos de aprendizado de máquina / David Evandro Amorim Martins; orientador: Eduardo Sany Laber. – Rio de janeiro: PUC-Rio, Departamento de Informática, 2020.

v., 113 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Departamento de Informática – Teses. 2. Otimização e Raciocínio Automático – Teses. 3. Aprendizado de Máquina;. 4. Processamento de Linguagem Natural;. 5. Vagas de Emprego.. I. Laber, Eduardo Sany. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD004

Agradecimentos

A minha querida esposa, Débora, por toda a paciência, compreensão e apoio, sem a qual nada disso seria possível.

Aos meus queridos pais, Euvaldo e Silvia, que sempre estiveram ao meu lado me incentivando a seguir esta jornada até o fim.

Ao meu amigo e antigo chefe, Julio Leite, por me autorizar a participar deste aprimoramento pessoal e profissional, bem como por ter resolvido muitos dos problemas relacionados.

Ao meu ex-chefe, Rodrigo Ossemer, por viabilizar a conclusão deste Mestrado me permitindo elaborar esta dissertação e resolvendo os problemas burocráticos que surgiram ao longo da jornada.

A Petrobras por me autorizar a participar deste aprimoramento pessoal, bem como por financiar este Mestrado.

Ao meu professor orientador, Eduardo Laber, que sempre colaborou comigo ao longo do Mestrado, tanto nas matérias quanto na elaboração deste trabalho.

A PUC-RIO por possibilitar a execução deste trabalho de dissertação.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001

Resumo

Martins, David Evandro Amorim; Laber, Eduardo Sany. **Segmentação semântica de vagas de emprego: estudo comparativo de algoritmos clássicos de aprendizado de máquina**. Rio de Janeiro, 2020. 113p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Este trabalho demonstra como *web mining*, processamento de linguagem natural e aprendizado de máquina podem ser combinados para melhorar a compreensão de vagas de emprego segmentando semanticamente os textos de suas descrições. Para atingir essa finalidade, foram coletados dados textuais de três grandes *sites* de vagas de emprego: Catho, LinkedIn e VAGAS.com.br. Baseado na literatura, este trabalho propõe uma estrutura semântica simplificada em que cada sentença da descrição da vaga de emprego pode pertencer a uma dessas classes: Responsabilidades, Requisitos, Benefícios e Outros. De posse dessa ideia, a tarefa de segmentação semântica pode ser repensada como uma segmentação de sentenças seguida de uma classificação. Usando o Python como ferramenta, são experimentadas algumas formas de construção de atributos a partir de textos, tanto léxicas quanto semânticas, e quatro algoritmos clássicos de aprendizado de máquina: Naïve Bayes, Regressão Logística, Máquina de Vetores de Suporte e Floresta Aleatória. Como resultados, este trabalho traz um classificador (Regressão Logística com representação binária) com 95.58% de acurácia, sem sobreajuste de modelo e sem degenerar as classificações por desbalanceio de classes, que é comparável ao estado da arte para Classificação de Texto. Esse classificador foi treinado e validado usando dados do Catho, mas foi testado também nos dados do VAGAS.com.br (88.60%) e do LinkedIn (91.14%), apresentando uma evidência de que seu aprendizado é generalizável para dados de outros *sites*. Além disso, o classificador foi usado para segmentação semântica das vagas de emprego e obteve uma métrica P_k de 3.67% e uma métrica *WindowDiff* de 4.78%, que é comparável ao estado da arte de Segmentação de Texto. Por fim, vale salientar duas contribuições indiretas deste trabalho: 1) uma estrutura para pensar e analisar vagas de emprego e 2) uma indicação de que algoritmos clássicos também podem alcançar o estado da arte e, portanto, sempre devem experimentados.

Palavras-chave

Aprendizado de Máquina; Processamento de Linguagem Natural; Vagas de Emprego.

Abstract

Martins, David Evandro Amorim; Laber, Eduardo Sany (Advisor). **Semantic Job Vacancy Segmentation: Comparative Study of Classical Machine Learning Algorithms**. Rio de Janeiro, 2020. 113p. Dissertação de mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

This dissertation demonstrates how web mining, natural language processing, and machine learning can be combined to improve understanding of job openings by semantically segmenting the texts of their descriptions. To achieve this purpose, textual data were collected from three major job sites: Catho, LinkedIn and VAGAS.com.br. Based on the literature, this work proposes a simplified semantic structure in which each sentence of the job description can belong to one of these classes: Responsibilities, Requirements, Benefits and Others. With this idea, the semantic segmentation task can be rethought as a sentence segmentation followed by a classification. Using Python as a tool, some ways of constructing features from texts are tried out, both lexical and semantic, and four classic machine learning algorithms: Naïve Bayes, Logistic Regression, Support Vector Machine, and Random Forest. As a result, this work presents a classifier (Logistic Regression with binary representation) with 95.58% accuracy, without model overfitting and without degeneration by class unbalance, which is comparable to state-of-the-art for Text Classification. This classifier was trained and validated using Catho data, but was also tested on VAGAS.com.br (88.60%) and LinkedIn (91.14%) data, providing evidence that its learning is generalizable to data from other sites . In addition, the classifier was used for semantic segmentation of job openings and obtained a P_k metric equals to 3.67% and a WindowDiff metric equals to 4.78%, which is comparable to state-of-the-art for Text Segmentation. Finally, it is worth highlighting two indirect contributions of this work: 1) a structure for thinking and analyzing job openings and 2) an indication that classical algorithms can also reach the state of the art and therefore should always be tried.

Keywords

Machine Learning; Natural Language Processing; Job Vacancies.

Sumário

Lista de figuras

Lista de tabelas

Lista de Abreviaturas

BERT – *Bidirectional Encoder Representations from Transformers*
BiLSTM – *Bidirectional Long Short Term Memory*
BiRNN – *Bidirectional Recursive Neural Network*
BOW – *Bag-of-Words*
CBOW – *Continuous Bag-of-Words*
CNN – *Convolutional Neural Network*
CSPCA – *Convex Sparse Principal Component Analysis*
DF – *Document Frequency*
ELMo – *Embeddings from Language Models*
GloVe – *Global Vectors*
IDF – *Inverse Document Frequency*
IGFSS – *Improved Global Feature Selection Scheme*
ISA – *Imprecise Spectrum Analysis*
JM – *Job Matching*
KNN – *K-Nearest Neighbours* LDA – *Linear Discriminant Analysis*
LGT – *Local, Global, Topical*
LI – *Least Information*
LSI – *Latent Semantic Indexing*
MBFS – *Meaning Based Feature Selection*
MLE – *Maximum Likelihood Estimator*
MRDC – *Multivariate Relative Discrimination Criterion*
NB – *Naïve* Bayes
NDM – *Normalized Difference Measure*
NLP – *Natural Language Processing*
PCA – *Principal Component Analysis*
RSLP – Removedor de Sufixos da Língua Portuguesa
SGNS – *Skip-gram Negative Sampling*
SVD – *Singular Value Decomposition*
SVM – *Support Vector Machine*
TF – *Term Frequency*
TF.IDF – *Term Frequency - Inverse Document Frequency*
t-SNE – *t-Distributed Stochastic Neighbour Embedding*
ULMFiT – *Universal Language Model Fine-Tuning*

"Essentially, all models are wrong, but some are useful"

George E. P. Box

1
Introdução

A adequação à função ou *job matching (JM)* diz respeito a compatibilidade entre alguém que busca emprego e uma vaga de emprego. Segundo (1, tradução nossa), JM é "o processo de garantir a compatibilidade entre os atributos individuais relacionados a trabalho e as características correspondentes do ambiente de trabalho" [1].

Obter um JM que atenda às expectativas dos candidatos e dos recrutadores requer um completo entendimento da vaga e do candidato em consideração (2). Para isso, primeiramente, é preciso que tanto o currículo do candidato quanto a descrição da vaga de emprego tenham uma estrutura clara e objetiva.

Existem diferenças constitutivas entre o texto de um currículo e o texto de uma descrição de vaga que precisam ser consideradas no JM. Isso pode ser percebido, por exemplo, na forma como as informações aparecem em ambos os documentos: os candidatos tendem a fornecer informações específicas em seus currículos (e.g. conhecimento em Java e C++), enquanto as empresas tendem a fornecer informações gerais em suas vagas (e.g. conhecimento em programação orientada a objetos) (3).

Este trabalho foca nas descrições de vaga de emprego e suas divisões estruturais. Aqui as descrições são entendidas como "um breve texto que descreve o emprego listando suas principais responsabilidades e especificando a mínima qualificação necessária para executá-lo" [2] (4, tradução nossa). Suas divisões estruturais, por sua vez, são consideradas como uma simplificação da ontologia apresentada em (5) e consistem em quatro partes básicas: responsabilidades, requisitos, benefícios e outros.

A disponibilidade de grandes volumes de dados, que já estão em formato eletrônico e prontos para uso analítico, e técnicas de análise preditivas ajudam usuários a tomarem melhores decisões, de forma mais consistente e com menos custo. Este trabalho demonstra como *web mining*, processamento de linguagem

[1] "[...]*job matching* (JM) denotes the process of ensuring compatibility between an individual's work-related attributes and the corresponding characteristics of their work environment."

[2] "A job description is a brief statement that describes a job by listing major duties and specifying the minimum qualifications needed to do it [...]"

natural (NLP) e aprendizado de máquina podem ser combinados para melhorar a compreensão de descrições de vagas de emprego utilizando atributos extraídos de dados textuais obtidos na internet.

Depois de discutir os fundamentos teóricos e aplicações práticas existentes na literatura, este trabalho desenvolve uma forma de segmentar vagas de emprego em suas divisões estruturais, utilizando técnicas tradicionais de aprendizado de máquina aplicadas a dados oriundos de diversos sites de recrutamento *online*, a saber: Catho, LinkedIn e VAGAS.com.br.

No contexto deste trabalho, aprendizado de máquina é todo sistema que, "ao ser apresentado a muitos exemplos relevantes de uma determinada tarefa, encontra uma estrutura estatística que eventualmente permite que o sistema obtenha regras para automatização da tarefa" [3] (6, tradução nossa). Aqui é considerada a tarefa de classificação das sentenças de vagas de emprego como forma de segmentação delas em suas divisões estruturais.

Este capítulo apresenta uma visão geral da dissertação estabelecendo o contexto, evidenciando os problemas, formulando as questões e objetivos perseguidos por esta pesquisa em conjunto com sua relevância e significância. Ele também introduz os métodos empregados, destaca as contribuições desta pesquisa para o conhecimento científico e finaliza apresentando a estrutura da dissertação.

1.1 Contexto

Com a emergência da internet e das redes sociais, passaram a ser geradas quantidades gigantescas de dados a cada segundo. Esses dados podem ser coletados de diversas fontes usando diferentes ferramentas e métodos como formulários *web* e documentos. Essa quantidade abundante de dados contém conhecimento escondido que pode ser utilizado pelas organizações (empresas, governo) para melhorar estratégias organizacionais de muitas formas, como através da criação e do desenvolvimento de produtos, serviços e processos inovadores (7). Com a disponibilidade de grandes volumes de dados em formato eletrônico, se faz muito importante o uso de técnicas complexas de análise de dados, que levem em conta tanto a quantidade quanto a natureza dos dados (numérico, textual, estruturado, desestruturado, ...), para permitir um processo de recrutamento mais otimizado e mais consistente.

A tarefa de *Job Matching* tem uma alta complexidade inerente a tarefas envolvendo preferências e satisfação pessoal de pessoas físicas e jurídicas. Além

[3] "It's presented with many examples relevant to a task, and it finds statistical structure that eventually allows the system to come up with rules for automating the task."

disso, é bastante sensível dado seu impacto social, sendo uma das causas de desemprego natural, de subutilização de funcionários e de vagas em aberto (8). Por essas razões, JM tem sido analisado por vários pesquisadores em diferentes disciplinas, como economia, psicologia e sociologia (9), e tem sido objeto de preocupação dos Estados desde pelo menos a década de 70 (10).

A necessidade de automatizar a tarefa de *Job Matching* também não é recente, como é possível perceber na manifestação do Secretário de Estado americano na década de 70: "fazer o máximo uso possível do processamento de dados eletrônicos e de sistemas de telecomunicações para armazenamento, recuperação e comunicação entre o empregador e o trabalhador" [4](10, tradução nossa). Pensando na automatização, sem esquecer a complexidade humana, surge na Europa o projeto Eduworks (11) que se propõe a lidar com os resultados encontrados por pesquisadores independentes de várias disciplinas, como gestão de recursos humanos, economia do trabalho, sociologia, utilizando aprendizado de máquina para melhorar o JM. Esse processo de compatibilidade candidato-vaga recebe como entrada:

- Dados da vaga de emprego: a fim de determinar o conjunto de requisitos de habilidades e as especificações dos trabalhos;
- As habilidades e perfis dos candidatos: a fim de determinar o conjunto de habilidades adquiridas pelo candidato que o tornem apto para um determinado trabalho.

De posse dessas informações, o serviço oferecido por Eduworks faz, entre outras coisas, uma análise de compatibilidade entre os requisitos da vaga de emprego e o conjunto de habilidades do candidato, sugerindo, por fim, estratégias de desenvolvimento pessoal que preencham as lacunas do candidato.

Aqui no Brasil, a tendência é a mesma: automatização do processo de recrutamento. A primeira fase desse processo de transformação digital se dá, a partir da década de 90, quando começam a surgir sites de anúncios de emprego, como Catho e VAGAS.com.br (12, 13). Essas empresas, atualmente com bancos de currículos e de descrições de vagas contendo milhões de registros, aproveitaram a onda tecnológica mais recente e se engajaram em transformação desses dados em informações e conhecimentos, utilizando ciência de dados e aprendizado de máquina, para as empresas e para os usuários (14). Além dessas empresas que já são consolidadas no mercado de vagas de emprego no Brasil, empresas internacionais, como a rede social profissional LinkedIn criada

[4] "[...]the Secretary was directed further to make maximum possible use of electronic data processing and telecommunications systems for the storage, retrieval, and communication of job and worker[...]"

em 2003 (15), chegam ao Brasil a partir de 2010 para oferecer serviços que melhoram a conexão entre recrutador e candidato utilizando também técnicas complexas de análise de dados. Entretanto, não só as grandes empresas estão nesse negócio, segundo (16), existem hoje 122 *HR Techs* (*startups* de recursos humanos) no Brasil trazendo uma abordagem mais assertiva e direcionada para o uso de tecnologias mais inovadores no processo de recrutamento. Elas representam 1.5% do total de *startups* brasileiras, e, pelo menos, 33% delas estão envolvidas de alguma forma na tarefa de *Job Matching*, seja atuando em recrutamento e seleção de efetivos e temporários, em entrevistas, em plataformas de vagas e análise de perfil de candidatos (16).

Tomando como base esse cenário, este trabalho tem a intenção de permitir um melhor entendimento semântico de vagas de emprego, aplicando técnicas de aprendizado de máquina a grande quantidade de dados textuais desestruturados de descrição de vagas disponíveis na internet.

1.2 Motivação

Atualmente, é normal que empresas analisem perfis *online* de potenciais candidatos em sites de emprego ou rede sociais profissionais. Elas publicam vagas de emprego e produzem grandes volumes de dados sobre descrições de vagas que podem ser explorados para melhorar o entendimento semântico das vagas (17). Alguns sites de empregos, como o Catho e VAGAS.com.br, já começaram a usar seus dados de vagas de emprego para traçar as trajetórias de carreira mais comuns no mercado brasileiro, apontando características como gênero, graduação, salário e próximas promoções (18, 19).

Apesar desses avanços, um significante número de vagas de emprego ainda não é ocupado por que as empresas não encontram candidato adequadamente qualificado (20). Por outro lado, é difícil para o candidato encontrar uma vaga compatível com suas habilidades e qualificações devido a grande quantidade de vagas de emprego publicamente disponíveis. Por exemplo, como mostrado na Figura 1.1, a busca por uma vaga com *"Web"* retorna mais de 1000 resultados, dos quais as vagas mais relevantes tem título bastante variado (*Web Analytics*, Programador *Web* e Desenvolvedor *Web*-Drupal) (21). A partir disso, fica claro que a quantidade de resultados, 1335 vagas nesse caso, e a diversidade das vagas apresentadas tornam difícil ao candidato conseguir o trabalho mais adequado e ao empregador alcançar o candidato mais adequado.

Além do volume de dados, a falta de estrutura nas descrições de vaga de emprego compromete o *job matching*, não deixando claro para o candidato quais são as responsabilidades do emprego e as qualificações mínimas, por

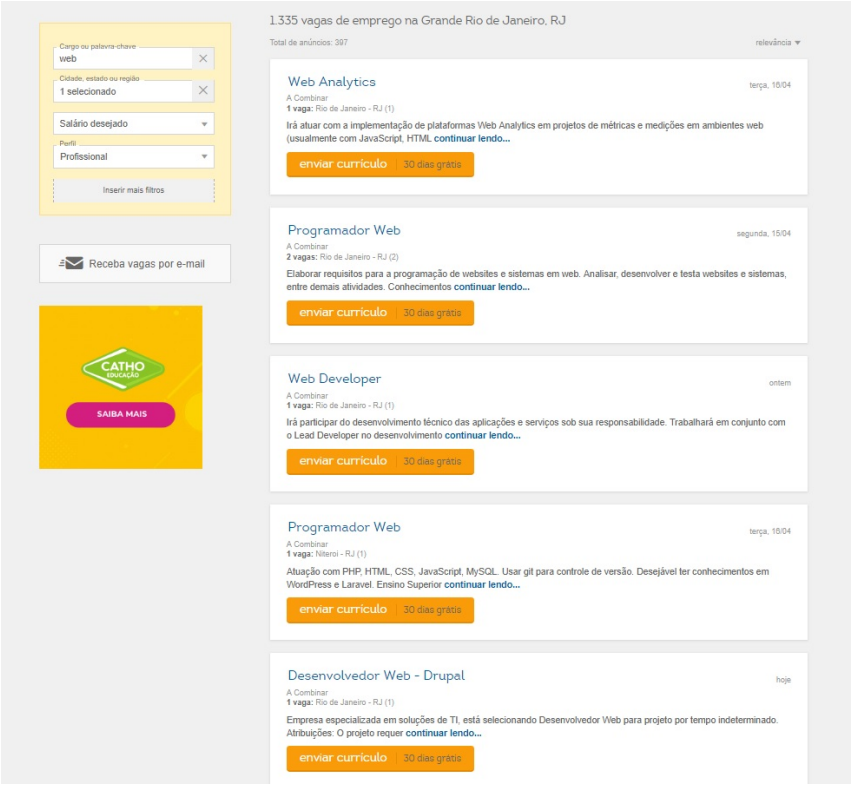

Figura 1.1: Primeira página de resultados para a busca *"Web"* no *site* Catho

exemplo. Para permitir que as vagas tenham uma estrutura básica, de forma que possam ser usadas para extrair informações relevantes para os candidatos, é necessário "dividir o texto em segmentos, de forma que cada segmento tenha tópicos coerentes"[5] (22, tradução nossa). Essa tarefa é chamada de **Segmentação de Texto**.

Muitas abordagens foram utilizadas para resolver essa tarefa de forma automática, sendo que as mais tradicionais utilizam aprendizado não supervisionado e são baseadas no conceito de coesão léxica. No contexto deste trabalho, coesão léxica é o uso de palavras relacionadas semanticamente em cada segmento que compõe o texto (23). Como medir a coesão léxica e como inferir a segmentação a partir da sua medida tem sido objeto de estudo na área de processamento de linguagem natural há algum tempo (24, 25, 26, 27)

Apesar disso, há alguns problemas em aplicar essas técnicas clássicas de segmentação de texto a descrições de vagas de emprego. O primeiro deles é o que os textos de vagas de emprego são curtos, contendo em média entre 60 e 300 palavras, e não são formais em estrutura sintática e semântica. Isso implica que assumir alguma forma de coesão é otimista. Outro problema é que mesmo que essas técnicas fossem utilizadas, ainda restaria a tarefa de classificar cada

[5] "[...]dividing text into segments, such that each segment is topically coherent[...]"

segmento com uma categoria que permitisse a extração de informação (28).

Pensando na natureza do texto da vaga de emprego e na tarefa de extração de informações básicas estruturadas, uma outra forma de resolver o problema é utilizar uma segmentação do texto simplificada (como a segmentação em sentenças ou em itens de uma lista) e classificar de forma automática cada segmento simplificado. Assim, transforma-se a tarefa em **Classificação de Texto** e isso permitiu a este trabalho focar na semântica dos segmentos em vez de somente na segmentação do texto.

Essas questões relativas a estruturação das descrições de vaga de emprego, de forma a torná-las mais informativas e facilitar a posterior extração de conhecimento delas, levantaram as questões desta pesquisa apresentadas na seção 1.3.

1.3 Questões da Pesquisa

Para que a tarefa de *job matching* possa ser otimizada e automatizada, um primeiro passo é entender melhor as vagas de emprego a partir de suas descrições postadas em sites como Catho e VAGAS.com.br. Dessa forma, a principal questão endereçada por este trabalho é: como os dados de descrições de vaga de emprego disponíveis na internet podem ser coletados, processados e utilizados para extrair automaticamente uma estrutura semântica de vagas de emprego?

Mais especificamente, este trabalho foca nas seguintes questões que são derivadas da questão principal:

- Como coletar os dados dos diversos sites de vagas de forma mais genérica possível?
- Qual estrutura semântica tem uma vaga de emprego?
- Como utilizar os dados para treinar um algoritmo de aprendizado de máquina capaz de segmentar a vaga e especificar o sentido semântico de cada segmento?
- Como fazer engenharia de atributos para dados textuais que não seguem a norma culta?
- Qual tipo de aprendizado de máquina é mais adequado: tradicional ou *deep learning*?
- Qual a capacidade de generalização do algoritmo treinado para vagas de emprego de outras fontes?

Portanto, este trabalho tem como objetivo estudar Classificação de Texto de descrições de vagas extraídas de grandes *sites* de emprego no Brasil. Como parte do estudo, é realizado um levantamento bibliográfico de abordagens utilizadas para este problema e outros problemas relacionados como *Job Matching* e Segmentação de Texto.

1.4 Contribuições do Trabalho

As principais contribuições deste trabalho são:

- Um fluxo de trabalho para coleta, processamento e análise de vagas de emprego, que foi experimentado com vários cenários, métodos e algoritmos clássicos de aprendizado de máquina;
- Uma estrutura geral para coleta de dados de vagas de emprego utilizando Python;
- Uma análise bibliográfica extensa sobre como modelar vaga de emprego, classificação e segmentação de texto;
- Uma evidência de que algoritmos clássicos de aprendizado de máquina podem alcançar resultados comparáveis ao estado da arte das tarefas, devendo portanto sempre serem experimentados;
- Um classificador simples (regressão logística com representação binária e sem pré-processamento) com 95.58% de acurácia, comparável com o estado da arte para Classificação de Texto. Este classificador, apesar de treinado com as vagas do Catho, generaliza bem para vagas vindas de outros sites de emprego: LinkedIn (91.14%) e VAGAS.com.br (88.6%);
- Uma forma diferente de pensar o problema de segmentação de texto como um problema de classificação de texto. O classificador simples treinado foi usado para segmentação e alcançou uma métrica P_k de 3.67% e uma métrica $WindowDiff$ de 4.78%, comparável com o estado da arte para Segmentação de Texto.

1.5 Estrutura da Dissertação

Esta dissertação está dividida em sete capítulos.

O Capítulo 1 - Introdução, que se encerra aqui, introduz a pesquisa, provê o contexto e a motivação deste trabalho. Ele também enumera as questões que este trabalho visa responder.

O Capítulo 2 - Descrição do Problema enuncia o problema que este trabalho se propõe a resolver e apresenta uma estrutura simplificada da solução utilizada neste trabalho.

O Capítulo 3 - Revisão Bibliográfica apresenta toda a pesquisa realizada sobre os temas de Modelo de Vaga de Emprego, Segmentação e Classificação de Texto.

O Capítulo 4 - Fundamentação Teórica descreve detalhes teóricos dos principais algoritmos, heurísticas e métricas utilizadas neste trabalho.

O Capítulo 5 - Implementação traz explicações sobre os códigos implementados para os *crawlers* dos *sites* de vagas de emprego e para os algoritmos, heurísticas e métricas descritas no Capítulo 4.

O Capítulo 6 - Resultados apresenta e discute os experimentos em busca da melhor combinação entre pré-processamento, construção de atributos e algoritmo de aprendizado. Ele também apresenta os resultados da aplicação do melhor classificador treinado para a tarefa de segmentação semântica da vaga de emprego.

O Capítulo 7 - Conclusões e Trabalhos Futuros encerra este trabalho trazendo comentários e soluções para as perguntas proposta por esta pesquisa, indicando algumas linhas de trabalho futuro para dar continuidade ao que foi desenvolvido.

2
Descrição do Problema

De forma mais abstrata, *Job Matching*, como foi visto no Capítulo 1, pode ser entendido como "o processo de garantir a compatibilidade entre os atributos individuais relacionados a trabalho e as características correspondentes do ambiente de trabalho" (1). Entretanto, esse enunciado é insuficiente para que o problema seja modelado porque apresenta conceitos mal definidos como compatibilidade, atributos e características . Dessa forma, para esclarecer e evitar ambiguidades, este trabalho, daqui em diante, trata o problema como um enunciado mais formal a seguir apresentado.

Seja $J = \{J_1, J_2, ..., J_n\}$ um conjunto de n descrições de vagas de emprego e $C = \{C_1, C_2, ..., C_m\}$ um conjunto de m currículos de candidatos. Para melhorar o *Job Matching* para o currículo C_i é preciso encontrar uma ordenação de J baseada em uma medida de similaridade entre C_i e J_k para todo $J_k \in J$. Como tanto os elementos de J quanto os de C são documentos, para que uma medida de similaridade entre eles seja calculada é necessário haver uma representação numérica num espaço adequado. Esse processo é representado em forma de fluxograma na Figura 2.1.

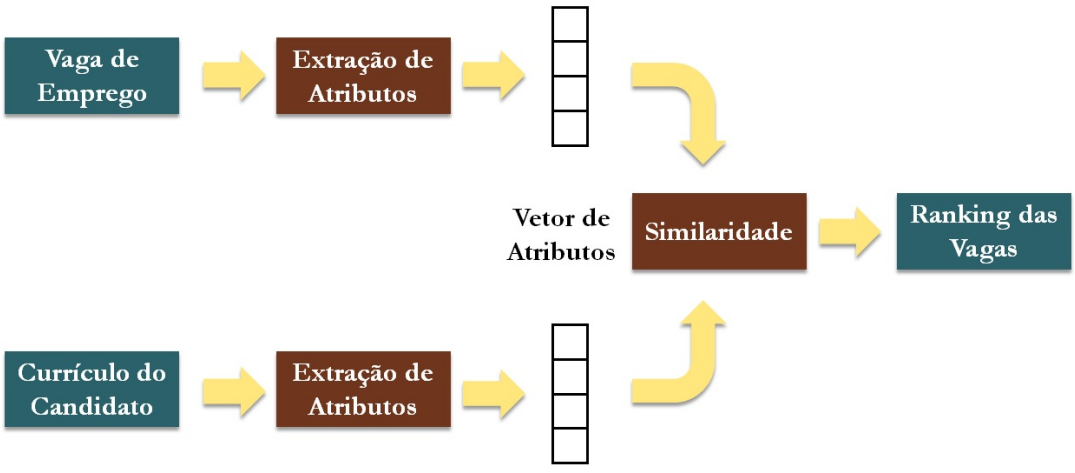

Figura 2.1: Representação em Fluxograma do Processo de Ordenação de Vagas

A representação numérica de um documento é chamada de **Modelo de Espaço Vetorial** e ela consiste em representar um documento por um

vetor de palavras (atributos) cujos valores são os pesos dessas palavras no documento (29). Existem diversas formas de calcular esses pesos e algumas delas serão explicadas no Capítulo 4. Apesar de muito utilizada essa forma de representação dos documentos, quando exclusivamente baseada em contagem de palavras, permite, apenas, o cálculo de similaridade léxica.

Para que a ordenação das vagas tenha relevância para o problema de *Job Matching* é importante que ela seja baseada em similaridade semântica. Uma técnica amplamente utilizada para endereçar essa questão é a **Análise de Semântica Latente**, que consiste em induzir tópicos latentes a partir da distribuição de palavras nos documentos de um conjunto de documentos, expandindo assim o Modelo de Espaço Vetorial para vetores de tópicos.

O problema dos tópicos latentes é que, por eles serem oriundos de aprendizado não supervisionado, eles não são explicitamente interpretáveis, necessitando de trabalho humano em rotular os tópicos. Buscando suplantar essa necessidade, este trabalho propõe uma estrutura ilustrada na Figura 2.2, já mencionada no Capítulo 1. Ela consiste em dividir o texto da vaga em sentenças classificadas, para a partir delas extrair atributos semânticos relevantes como atividades executadas e habilidades requeridas.

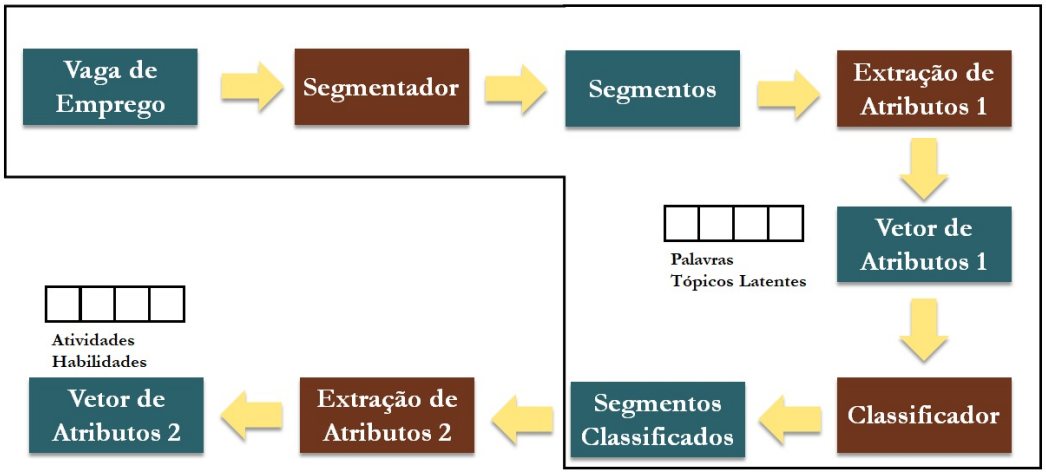

Figura 2.2: Representação em Fluxograma do Processo de Extração de Atributos Semânticos

Este trabalho foca nas etapas destacadas na Figura 2.2. De forma mais específica, elas consistem em:

1. Segmentar a descrição da vaga de emprego em segmentos semânticos ou sentenças. Neste trabalho, é considerada a segmentação em sentenças, por simplicidade, uma vez que normalmente dentro de uma sentença não havia mais de uma classe;

2. Extrair atributos dos segmentos gerando um vetor de atributos. Neste trabalho, são estudadas diversas formas de extrair os atributos de uma sentença, como Modelo de Espaço Vetorial e *Word Embeddings* (vetores de palavras baseadas em contexto);

3. Treinar um classificador de sentenças. Neste trabalho, são estudados algoritmos clássicos de classificação, como Regressão Logística e *Random Forest*.

Cada segmento é considerado como pertencendo a uma classe entre quatro: Responsabilidades, Requisitos, Benefícios e Outros. Essas classes foram escolhidas baseadas em (5) e na disponibilidade de geração automática de classificação dos dados coletados dos *sites* de vagas de emprego. A semântica de cada classe é apresentada a seguir e é baseada em (5):

– **Responsabilidades** - são os deveres que o empregado precisa executar no emprego, como: 1 - responsável por gerenciar as vendas diárias; 2 - elaborar relatórios diários e emiti-los para a diretoria;
– **Requisitos** - são todos os atributos que são requeridos do candidato para que ele seja aceito em um emprego, como habilidades, experiência, nível de educação e posse de bens materiais (*notebook*, celular, carro);
– **Benefícios** - são as vantagens que o empregado tem por trabalhar no emprego, sejam elas complementares ao salário, como vale-alimentação, ou competitivas com relação a outros empregos, como carro da empresa e vestimenta informal;
– **Outros** - são todas as outras coisas que não se enquadram em nenhuma das classes anteriores.

Para tratar esse problema de Classificação de Texto, é necessário que exista um conjunto de dados de vagas de emprego com gabarito. Entretanto, não havia nenhum *dataset* em língua portuguesa disponível. Por isso, este trabalho também se propõe a criar esse *dataset* coletando de forma automática dados de três grandes *sites* de emprego no Brasil: Catho, VAGAS.com.br e LinkedIn. O gabarito da classificação foi gerado automaticamente, sempre que possível, durante a coleta. Maiores detalhes sobre a coleta e os *sites* são apresentados no Capítulo 5.

3
Revisão Bibliográfica

Para permitir um melhor posicionamento sobre o problema descrito no capítulo anterior, este trabalho levantou na literatura como três grandes problemas relacionados são tratados: *Job Matching*, Segmentação de Texto e Classificação de Texto.

3.1
Modelo de Vaga de Emprego

Essa seção apresenta como os trabalhos anteriores relacionados a *Job Matching* modelam a descrição da vaga de emprego para medir a similaridade entre empregos ou entre emprego e candidato. De uma forma geral, existem três formas de modelar as vagas de emprego na literatura:

- **Estruturada** - elabora uma estrutura para informações contidas em vagas de emprego e extrai as informações do dado textual através de regras;
- **Semi-estruturada** - elabora uma lista de tópicos contidos em vagas de emprego, categoriza semanticamente segmentos do dado textual e a partir desses segmentos extrai as informações através de regras ou induz modelos de representação vetorial;
- **Não estruturada** - utiliza o dado textual bruto e induz diretamente modelos de representação vetorial do dado em espaços adequados, sejam eles léxicos e/ou semânticos.

A presença de uma estrutura em dados textuais uniformiza a terminologia e, portanto, facilita a busca de informações pelo computador (30). No caso de texto de vaga de emprego, a literatura apresenta estruturas que podem ser divididas pela **cobertura** (quanto da vaga de emprego é coberta pela estrutura) e pela **abrangência** (para que tipos de vaga de emprego a estrutura é válida). Tomando como base esses critérios, a estrutura pode ser classificada como de:

- **Cobertura Total** - estrutura que engloba todas as seções da vaga de emprego;

- **Cobertura Parcial** - estrutura que não engloba todas as seções da vaga de emprego;
- **Abrangência Universal** - estrutura que pode ser aplicada para qualquer tipo de vaga de emprego;
- **Abrangência Específica** - estrutura que pode ser aplicada para tipos específicos de vaga de emprego.

Alguns trabalhos buscam uma estrutura de cobertura total com abrangência universal, como em (5), por exemplo, em que os autores se propõem a prover uma estrutura geral para descrições de vaga de emprego pensando em melhorar o *Job Matching*. Para isso eles utilizam uma metodologia para desenvolvimento de ontologias e ela é aplicada ao domínio de vagas de emprego. Com base nessa ontologia elaborada, este trabalho pensou as classes em que segmenta o texto da vaga de emprego, conforme apresentado no Capítulo 2.

Outros trabalhos buscam uma estrutura de cobertura parcial com abrangência específica, como em (31, 32, 33, 34), em que os autores se propõem a analisar as habilidades de vagas de emprego para vagas de TI, Arquivista, *Controller* e Gerente, respectivamente. Em (31), desenvolve-se uma ontologia hierarquizada para habilidades de vagas de TI usando dados da DBpedia, que é uma base de dados estruturados extraídos da Wikipedia (35). Por outro lado, em (32, 33, 34), busca-se entender melhor as habilidades solicitadas nas vagas específicas supracitadas, utilizando a estrutura elaborada para coleta de informações e posterior análises estatísticas.

Pensando no problema complementar, sob a ótica do *Job Matching*, que é desenvolver uma estrutura para o texto do currículo, segue a mesma divisão de pesquisa entre estrutura de cobertura total com abrangência universal (36) e estrutura de cobertura parcial com abrangência específica (37). Em (36), os autores desenvolvem uma ontologia de todas as propriedades dos currículos, reaproveitando ontologias já desenvolvidas para conceitos mais amplos como educação, localização e organização. Por outro lado, a IBM, em (37), desenvolve uma estrutura simplificada para extração de informações de currículos de candidatos de TI buscando facilitar seu processo seletivo.

Apesar de buscar uma estrutura ser a forma mais intuitiva de começar a tornar dados textuais inteligíveis ao computador, ela limita a informação extraída ao que é julgado relevante e isso pode limitar o resultado dos algoritmos de aprendizado posteriormente aplicados. Além disso, a estrutura em si introduz uma complexidade em computar similaridade semântica dos dados (38), o que é uma complicação para a vaga de emprego e o problema de *Job Matching*.

Partir do texto da vaga de emprego diretamente para a estrutura, independente do tipo de cobertura ou de abrangência, é uma tarefa complexa e, por isso, na literatura se adota normalmente uma etapa intermediária em que o texto é dividido em tópicos. Nesse sentido, a abordagem semi-estruturada consiste numa segmentação semântica do texto para que através do uso de técnicas complementares se consiga extrair as informações necessárias.

Alguns dos trabalhos já citados na abordagem estruturada são exemplos de como a abordagem semi-estruturada é uma etapa intermediária. Em (31), para alcançar a estrutura, os autores primeiramente segmentam as vagas de emprego em sentenças/parágrafos representando título, descrição, local e habilidades, e depois extraem do segmento de habilidades os atributos necessários utilizando transdutores de estados finitos. Por outro lado, retomando o problema complementar supracitado, tem-se em (36) que os autores também segmentam o texto em seções como experiência profissional, educação, treinamentos, línguas estrangeiras e linguagens de computação, e depois extraem dos segmentos os atributos necessários utilizando regras baseadas em ontologias de conceitos mais amplos.

Outros trabalhos, entretanto, buscam segmentar o texto da vaga de emprego em tópicos e a partir desses segmentos computa métricas de similaridades. Em (39), o texto da vaga de emprego é dividido em segmentos (empregador, localização, tarefas e requisitos) para que seja possível computar a similaridade entre duas vagas através de uma métrica de distância aplicada a vetores semânticos induzidos por uma Análise de Semântica Latente. Em (40), os autores dividem o texto da vaga de emprego em onze campos numéricos e nove segmentos textuais, e, em seguida, relaciona vagas de emprego com técnicas de recuperação de informação usando os campos como consultas ao conjunto de vagas.

A busca de uma estrutura baseada em regras, a partir do texto segmentado ou não, é um tipo de solução pertencente ao primeiro movimento do processamento de linguagem natural (NLP), chamado Racionalista. Nele, se pressupõe que o entendimento de um texto somente é possível com o uso de regras artesanais elaboradas por seres humanos. Esse modo de pensar o texto, entretanto, foi atualizado num segundo movimento, chamado Empirista, no qual o entendimento do texto pode ser inferido, através de alguns algoritmos de aprendizado de máquina, a partir do próprio texto (41). Pensando nisso, é que surgem trabalhos que utilizam uma abordagem não estruturada para entender os textos de vaga de emprego (42, 43, 44, 45) ou de currículo (46). Todos esses trabalhos citados utilizam Modelo de Espaço Vetorial com algum processamento para representar o texto como um vetor de atributos.

Este trabalho se encaixa em uma perspectiva Racionalista com abordagem semi-estruturada, na medida em que se propõe a segmentar o texto da vaga de emprego e classificá-lo em tópicos. Entretanto, também se encaixa em uma perspectiva Empirista, pois a proposta é utilizar de técnicas de aprendizado de máquina, ao invés de regras, para dividir a vaga de emprego em segmentos semânticos.

3.2 Segmentação de Texto

Essa seção apresenta como os trabalhos anteriores relacionados a segmentação de texto se dividem e que tipo de técnicas eles utilizam para dividir o texto em tópicos. De uma forma geral, os métodos de solução do problema de segmentação de texto podem ser dividido em duas categorias (47):

- **Não Supervisionados** - aprendem a segmentar um texto a partir da variação de alguma propriedade latente do texto, logo depende apenas do conjunto de dados;
- **Supervisionados** - aprendem a segmentar um texto a partir de textos previamente segmentados, logo pressupõe a existência de um conjunto de dados com gabarito.

Existem diversas formas, na literatura, para resolver o problema de segmentação com métodos não supervisionados. Uma das primeiras delas aparece em (48, 49), em que o autor propõe um método para detecção de mudanças de tópicos em textos com vários parágrafos, utilizando as ideias de que: i) a ocorrência de palavras semelhantes delimita um tópico, e ii) a distribuição das palavras em cada tópico deve ser diferente entre si. Essas ideias são uma propriedade latente do texto chamada de **coesão léxica**. Muitos trabalhos são baseados nessa propriedade (26, 50, 51), variando entre eles o modo de calcular a semelhança entre palavras/frases e o modo de decidir em que ponto há a mudança de segmento.

Outra forma de pensar o problema é assumir que a língua ou o texto tem como propriedade latente um **modelo estatístico**. Em (52), os autores induzem modelos estatísticos de linguagem a partir dos textos, um para contexto local e outro para contexto global, e, com isso, calculam uma medida chamada de topicalidade, cuja variação abrupta indica uma mudança de segmento. Em (53), por outro lado, os autores introduzem um modelo de probabilidade condicional de uma segmentação, dado o texto e através de um algoritmo de busca, encontram a segmentação de máxima probabilidade.

Além desses, existe um grupo de trabalhos (54, 55) que considera que o agrupamento de palavras e sentenças baseado em uma medida de **semelhança semântica** é suficiente para segmentar um texto. Em (54), é introduzida uma forma de segmentar texto baseada em um algoritmo que, a partir de uma representação semântica das frases, agrupa-as em clusters, obtendo o número de clusters automaticamente. Essa técnica é chamada de clusterização por propagação de afinidade. Em (55), os autores utilizam esse mesmo algoritmo de clusterização mas se baseiam em representações semânticas das sentenças que são mais elaboradas.

Encerrando as soluções através de método não supervisionado, há trabalhos (51, 56, 57, 58) que assumem que cada palavra se refere a um tópico e, portanto, cada segmento tem uma **distribuição de tópicos** própria, a partir da qual é possível verificar a transição de segmentos. A principal vantagem de métodos desse tipo é que além dos segmentos, se obtém também uma distribuição de tópicos que pode ser utilizada para entendimento semântico do segmento.

Do lado dos métodos supervisionados, são propostos diversos tipos de classificadores/regressores como solução para a tarefa de segmentação de texto. Alguns trabalhos (59, 60) utilizam **árvores de decisão** como algoritmo de classificação e segmentação de texto de discurso falado. As diferenças entre os dois trabalhos são os atributos elaborados pelos autores ou induzidos dos dados. Enquanto (59) trabalha exclusivamente com atributos relacionados a entonação, (60) utiliza também atributos léxicos vindos de modelos estatísticos de linguagem.

Outros trabalhos propõem o uso de **modelos probabilísticos** clássicos de aprendizado de máquina como solução. Em (61), a partir dos dados e do gabarito de segmentação são treinados modelos de máxima entropia com atributos elaborados pelos autores. Em (62), os autores também criam seus atributos, mas treinam um modelo de regressão múltipla para fazer a previsão da segmentação.

Por fim, existe um grupo de trabalhos que busca resolver o problema de segmentação com **modelos neurais**. Eles variam principalmente no tipo de engenharia de atributos utilizada e na arquitetura da rede neural. Em (63), os autores utilizam redes neurais recorrentes bidirecionais (BiRNN) para codificar o texto e selecionar os segmentos. Em (64), por outro lado, são utilizadas redes neurais convolutivas (CNN) seguidas de redes neurais de memória de longo prazo bidirecionais (BiLSTM) com atenção para obter representações dos textos que permitam detectar melhor os segmentos.

É possível perceber que há uma variedade de soluções diferentes para

o problema de segmentação, entretanto a maioria delas trata do problema de segmentação para textos longos. No caso deste trabalho, tem-se textos de vagas de emprego que são curtos, contendo em média entre 60 e 300 palavras,o que torna difícil utilizar esses métodos. Além disso, este trabalho se propõe a agregar uma semântica explícita (uma classe) aos segmentos e isso não é feito nem pelos modelos de tópicos, que agregam uma semântica implícita, necessitando que alguém dê um rótulo aos tópicos encontrados. Por isso, neste trabalho a segmentação experimentada é uma consequência da classificação das sentenças, pois um conjunto de sentenças da mesma classe será considerado como um segmento.

Para fins de comparação com a literatura no resultado da tarefa de segmentação de texto, este trabalho usa as duas principais métricas P_k e *WindowDiff* (65) da área de segmentação. Elas são definidas e discutidas no Capítulo 4. Na Tabela 3.1 são apresentados os resultados dos três melhores algoritmos aplicados aos conjuntos de dados *Choi dataset* e *Clinical* descritos em (26, 64).

Tabela 3.1: Estado da Arte da Segmentação de Texto

Artigo	***Dataset***	P_k **(%)**	***WindowDiff (%)***
Attention-based Neural Text Segmentation (64)	*Clinical*	31.8	29.4
Bayesian Unsupervised Topic Segmentation (51, 54)	*Clinical*	-	35.3
SEGBOT: A Generic Neural Text Segmentation Model with Pointer Network (47)	*Choi dataset*	0.11	-
TopicTiling: A Text Segmentation Algorithm based on LDA (58)	*Choi dataset*	0.88	0.98

3.3 Classificação de Texto

Essa seção apresenta uma visão geral da estrutura de solução para o problema de classificação de texto e como os trabalhos anteriores se dividem com relação ao uso de diferentes técnicas em cada uma das etapas da solução. De uma forma geral, o processo de classificação envolve as seguintes etapas:

1. **Aquisição de dados** - etapa obrigatória em que se busca um conjunto de dados que represente um processo físico ou de negócio;

2. **Pré-processamento** - etapa opcional na qual se prepara o texto para gerar a representação requerida pelo método de aprendizado selecionado;

3. **Construção de atributos** - etapa obrigatória em que se representa o texto de forma inteligível ao computador;

4. **Seleção de atributos** - etapa opcional na qual são atribuídos pesos aos atributos através de algum critério e esses pesos são utilizados para selecionar o melhor conjunto de atributos para representar os dados textuais;

5. **Projeção de atributos** - etapa opcional em que os atributos são projetados em espaços de menor dimensão para obtenção de uma representação ótima dos dados;

6. **Treinamento de classificador** - etapa obrigatória na qual se treina uma função de classificação para ser capaz de reconhecer alguns conceitos, representados através de probabilidades ou pesos, que permitam definir uma decisão de classificação;

7. **Avaliação de classificador** - etapa obrigatória na qual se estima a qualidade do classificador e sua capacidade de generalização através de diversas métricas.

A etapa de aquisição de dados é mais artesanal e, portanto, varia bastante conforme o assunto e o tipo dos textos. Não é interesse deste trabalho explorar as diversas formas de realizar esta etapa. Por outro lado, a etapa de pré-processamento é bem consistente entre os trabalhos. Desta forma, ela é apresentada no Capítulo 4 como base teórica.

Na etapa de construção de atributos, o conjunto de dados devidamente categorizado é representado apropriadamente para o algoritmo de aprendizado. Duas representações bem conhecidas de dado textual são:

– **Modelo de Espaço Vetorial** - no qual um documento é representado como um vetor de palavras, termos ou frases (atributos) cujos valores são os pesos desses atributos no documento (29);
– **Modelo de Grafos** - no qual um documento é modelado como um grafo, por exemplo, com os nós representando palavras e as arestas representando as relações entre as palavras (66, 67).

Ambas as representações são baseadas em atributos e pesos. Na literatura, existem numerosas abordagens para a **construção dos atributos**, algumas das quais serão explicadas no Capítulo 4. A seguir são apresentadas algumas das abordagens mais comuns:

- Atributos simples (palavras-chave ou frases-chave) incluindo unigramas, bigramas e n-gramas (68, 69, 70, 71, 72);
- Taxonomias ou ontologias de atributos (73, 74, 75, 76, 77, 78, 79, 80);
- Atributos de domínios específicos, como listas de palavras associadas a emoção ou a opinião (81, 82, 83);
- Atributos embutidos, derivados com técnicas de *word embedding* (84), como Word2Vec (85, 86, 87, 88, 89), GloVe (90) e FastText(91, 92);
- Atributos embutidos, derivados com modelos de linguagem, como ULM-FiT (93), BERT(94), ELMo(95) e XLNet(96);
- Atributos semânticos simples, como entidades nomeadas ou frases nominais (73, 97, 98, 99);
- Atributos extraídos com modelos de tópicos (100, 101, 102, 103) ou espaços de dissimilaridade (104, 105);
- Outras informações, como o conhecimento extraído dos textos da Wikipedia (106) ou a importância de sentenças (107).

Depois de determinados os atributos, é necessário a designação de valores numéricos para eles. Essa questão de descobrir o valor numérico de um atributo, incluindo seu impacto na classificação de texto, é amplamente discutida na literatura. Os métodos existentes para isso podem ser agrupados em i) clássicos e ii) modernos. Os métodos clássicos, como binário, TF (*term frequency*), IDF (*inverse document frequency*) e TF.IDF (*term frequency - inverse document frequency*), são descritos no Capítulo 4. Por outro lado, alguns dos métodos modernos são referenciados a seguir.

- Esquema LGT (*local, global, topical*) (108);
- Esquemas modificados de ponderação de termos baseados em frequência(109);
- Abordagens adaptativas de ponderação de termos para classificadores de texto do tipo Naïve Bayes (110);
- Frequência de termos e momento de gravidade inverso (111);
- Esquemas de ponderação de termos induzidos por modelos probabilísticos (112);
- Ponderação profunda de termos (113);
- Frequência de termos baseada em densidade de classes (114);
- Modelos de mínima informação (em inglês, LI, *least information*) para frequência de termos (115);

- Algoritmos genéticos para combinar um conjunto de pesos básicos para gerar esquemas de ponderação de termos discriminativos (116);
- Ponderação de termos baseada em indexação de classes (117);
- Ponderação semântica de termos (118).

Seguindo no processo, as técnicas de **seleção de atributos** mantém apenas os atributos mais relevantes ou descritivos, descartando os restantes. A teoria clássica divide-as em três grupos principais chamados de filtros, *wrappers* e híbridas (119, 120, 121).

Os métodos de filtro utilizam uma função de *ranking* para selecionar os melhores atributos. Essa função quantifica a relevância baseada no conjunto de dados. De forma intuitiva, um atributo mais relevante estará melhor posicionado no *ranking*. Subsequentemente, os n melhores atributos são mantidos ou os n piores atributos são removidos.

Os métodos de *wrapper* são algoritmos genéricos empregados para realizar busca no espaço de subconjuntos de atributos. Além disso, eles testam a performance de cada subconjunto usando um algoritmo de aprendizado. Por fim, o subconjunto de atributos que obtiver a melhor performance é selecionado para o uso.

Os métodos híbridos aprendem quais atributos contribuem mais para a acurácia de um modelo enquanto o modelo está sendo treinado. Alguns algoritmos de aprendizado (como as árvores de decisão) incluem um método de seleção híbrido de tal forma que ele faça implicitamente parte do processo de aprendizado.

Existem diversas implementações que são o estado da arte das técnicas acima. Algumas delas são: *ranking* de Fisher, coeficientes de correlação, informação mútua, informação mútua pontual normalizada, χ^2, regressões de lasso, de *ridge* e *elastic net* (122, 123, 124, 125, 126, 127). Também é possível citar soluções projetadas especialmente para a classificação de texto, como:

- Critério de discriminação relativa multivariada (em inglês, *multivariate relative discrimination criterion* - MRDC) (128);
- Unificação de atributos (129);
- Busca por palavras discriminantes em espaço de atributos contínuos multidimensionais (130);
- Medida de diferença normalizada (em inglês, *normalized difference measure* - NDM) (131);
- Esquema variável de seleção global de atributos (132);

– Seleção de atributos baseada em significado (em inglês, *meaning based feature selection* - MBFS) (133);
– Seleção híbrida de atributos baseada em algoritmos genéticos melhorados (134);
– Esquema melhorado de seleção global de atributos (em inglês, *improved global feature selection scheme* - IGFSS) (135).

Na próxima etapa, métodos de **projeção de atributos** projetam os atributos existentes em dimensões diferentes. O objetivo aqui é obter novos atributos de tal forma que a estrutura do novo conjunto de dados e sua variância retenham a estrutura do conjunto de dados original tanto quanto possível (136, 137). A seguir, é apresentada uma lista de várias das principais técnicas de projeção:

– Análise linear de componentes principais (em inglês, apenas *principal component analysis* - PCA) (138);
– Análise discriminante linear (em inglês, *linear discriminant analysis* - LDA) (138);
– Análise convexa esparsa de componentes principais (em inglês, *convex sparse principal component analysis* - CSPCA) (139);
– Análise espectral imprecisa (em inglês, *imprecise spectrum analysis* - ISA) para análise espectral linear de documentos (140);
– t-SNE (*t-Distributed stochastic neighbour embedding*) (141, 142);
– Indexação de semântica latente (em inglês, *latent semantic indexing* - LSI) (143), que é baseada em decomposição em valores singulares (em inglês, *singular value decomposition* - SVD);
– Análise de componentes principais baseada em *kernel*, que é uma versão não linear do PCA (144);
– Projeções aleatórias (145).

A etapa mais importante no processo de classificação de texto é o **treinamento do classificador** que consiste em escolher e treinar o melhor classificador. Sem um entendimento conceitual de cada algoritmo, é impossível determinar efetivamente qual modelo é mais eficiente. No Capítulo 4, as técnicas mais populares de classificação de texto são discutidas. Um dos algoritmos de classificação mais simples é a Regressão Logística (LR) que foi utilizada na maioria dos domínios de mineração de dados (146, 147, 148, 149).

No começo da história da área de recuperação de informação, o classificador Naïve Bayes (NB) era muito popular. No Capítulo 4 é apresentada uma

visão geral desse classificador que não é caro computacionalmente e que precisa de pouca quantidade de memória (123).

Outra técnica popular é a Máquina de Vetores de Suporte (em inglês, *Support Vector Machines* - SVM) (150, 151) que emprega um classificador discriminativo para categorizar documentos. Essa técnica também pode ser usada em todos os domínios de mineração. Normalmente este modelo é usado como *baseline* para que os pesquisadores possam comparar com seu próprio trabalho e possam enfatizar suas contribuições.

Algoritmos de classificação baseados em árvores, como Árvore de Decisão (em inglês, *Decisiton Tree* - DT) e Floresta Aleatória (em inglês, *Random Forest* - RF), também foram utilizados para categorização de texto (152). Por fim, antes de partir para abordagens neurais, técnicas não paramétricas, como KNN (*K-Nearest Neighbours* (153) também foram aplicadas para resolver o problema de classificação de texto.

Recentemente, métodos que se baseiam em redes neurais profundas tem chamado atenção por obter resultados melhores que todos os algoritmos anteriores de aprendizado de máquina. O sucesso desses algoritmos reside na sua capacidade de modelar relações complexas e não lineares a partir dos dados (154). Entretanto, eles exigem uma quantidade massiva de dados para que tenham capacidade de aprender. Apesar disso, o estado da arte na área de classificação de texto é atingido por trabalhos utilizando redes neurais de arquiteturas profundas (93, 94, 96, 155, 156, 157, 158, 159, 160).

Tabela 3.2: Estado da Arte da Classificação de Texto

Artigo	**Modelo**	***Dataset***	**Acurácia (%)**
XLNet: Generalized Autoregressive Pretraining for Language Understanding (96)	XLNet	DBPedia	99.38
	XLNet	*AG News Corpus*	95.51
Universal Language Model Fine-tuning for Text Classification (93)	ULMFiT	DBpedia	99.20
	ULMFiT	*AG News Corpus*	94.99
Supervised and Semi-Supervised Text Categorization using LSTM for Region Embeddings (156)	CNN	DBpedia	99.16
	CNN	*AG News Corpus*	93.43

Depois que o modelo de classificação está treinado, deve-se escolher indicadores para **avaliação do classificador**, a última etapa do processo de classificação de texto. Primeiramente, mede-se um indicador ou grupo de indicadores, incluindo precisão, *recall*, acurácia, F-score, especificidade,

área sob a curva (em inglês, *Area Under Curve* - AUC) e erro (161, 162). Eles são descritos em um nível macro ou micro (162) e seus métodos de cálculo estão relacionados ao tipo de problema de classificação considerado, i.e. binário, múltiplas classes ou múltiplos rótulos (123, 162). Além disso, é possível selecionar indicadores mais orientados a performance computacional, como memória total alocada para o modelo de classificação ou o tempo de CPU utilizado para treinamento e teste (163).

Para fins de comparação com a literatura no resultado da tarefa de classificação de texto, este trabalho usa a acurácia como métrica. Na Tabela 3.2 são apresentados os resultados dos três melhores algoritmos aplicados aos conjuntos de dados *AG News Corpus* e DBPedia descritos em (155).

4
Fundamentação Teórica

Para aprofundar o entendimento sobre os algoritmos e técnicas implementadas/utilizadas, este trabalho apresenta uma visão teórica sobre os problemas de Segmentação e Classificação de Texto, com ênfase no segundo, que é principal objeto de estudo desta dissertação.

4.1
Segmentação de Texto

Como este trabalho se propõe a tratar o Problema de Segmentação como um problema de Classificação, essa seção é simplificada, focando apenas em conceituar as principais métricas de avaliação de segmentadores: P_k e *WindowDiff* e explicar porque elas são melhores que métricas como precisão e *recall*.

No contexto da segmentação, precisão é a porcentagem de fronteiras identificadas pelo algoritmo que são fronteiras reais dos segmentos, enquanto *recall* é a porcentagem de fronteiras verdadeiras que foram identificadas pelo algoritmo (65). Apesar de ambas as métricas serem eventualmente utilizadas para segmentadores de texto, elas não são as principais por dois motivos. O primeiro é que há uma complementariedade natural entre precisão e *recall*, ou seja, aumentar uma das métricas tende a reduzir o valor da outra. No caso da segmentação, posicionar mais fronteiras, por exemplo, aumenta o valor do *recall*, mas reduz o valor da precisão, conforme é possível perceber no comportamento dos segmentadores hipotéticos *SegA* e *SegB* na Figura 4.1. Para tentar compensar isso, alguns trabalhos optam por usar uma combinação ponderada das duas métricas, conhecida como F-score (123), mas essa medida é de difícil interpretação (52).

O segundo problema com a precisão e o *recall* é que nenhuma das duas métricas é sensível ao quase acerto. Seja, por exemplo, a segmentação de referência e os resultados obtidos por dois algoritmos de segmentação diferentes, como mostra a Figura 4.2. Em ambos os casos, os algoritmos falham em encontrar precisamente todas as fronteiras e, portanto, ambos tem precisão e *recall* iguais a 0. Entretanto, o algoritmo *SegA* está mais próximo da segmentação correta em todas as fronteiras, enquanto o algoritmo *SegB* erra

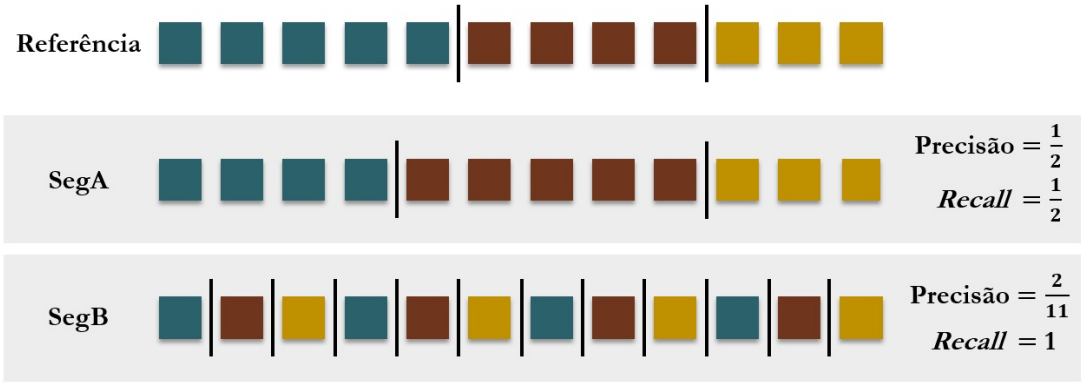

Figura 4.1: Representação da Natureza Complementar entre Precisão e *Recall*

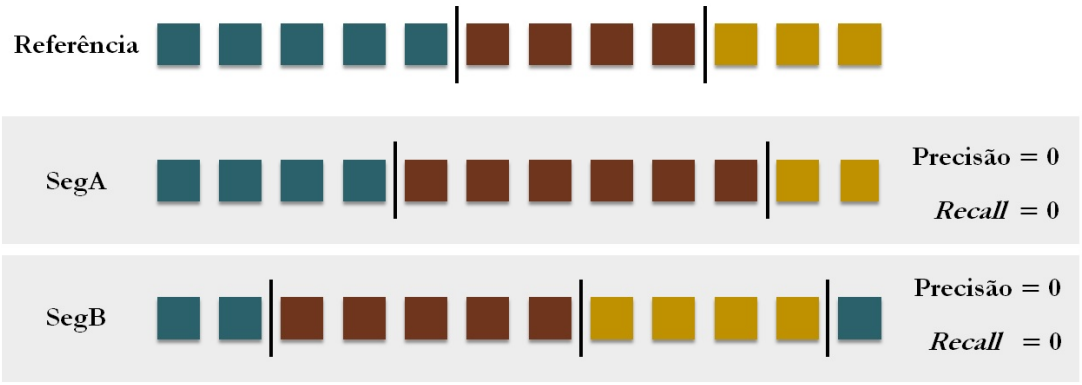

Figura 4.2: Representação da Incapacidade de Precisão e *Recall* de Distinguir Algoritmos Quase Errado

bastante a posição das fronteiras, adicionando inclusive mais segmentos que a referência. Dessa forma, é valioso que exista uma métrica de avaliação que penalize o segmentador *SegA* menos que o segmentador *SegB*.

Em (52), é introduzida uma nova métrica que tenta resolver os problemas com a precisão e o *recall*, incluindo contabilizar algum crédito pelos quase acertos. Essa métrica, P_D, é definida como a probabilidade de que duas sentenças sorteadas aleatoriamente sejam corretamente identificadas como pertencentes ou não a um mesmo segmento. Mais formalmente, dadas duas segmentações *ref* e *hip* para um texto com n sentenças,

$$P_D(ref, hip) = \sum_{1 \leq i \leq j \leq n} D(i,j)(\delta_{ref}(i,j) \bar{\oplus} \delta_{hip}(i,j))$$

Aqui a função δ_{ref} é uma função indicador que vale 1 se i e j são sentenças que pertencem ao mesmo segmento na segmentação de referência e vale 0, caso contrário. De forma semelhante, δ_{hip} vale 1 se o segmentador afirmar que

as sentenças i e j pertencem ao mesmo segmento e vale 0, caso contrário. O operador $\bar{\oplus}$ é função XNOR, negação da função ou-exclusivo, que vale 1 somente quando ambos operandos são iguais. Por último, a função D é a **distribuição de probabilidade da distância** sobre o conjunto de todas as possíveis distâncias entre sentenças escolhidas aleatoriamente no documento.

Essa métrica, entretanto, tem como complicação associada a escolha da função D. Para suplantar isso, é proposta uma versão simplificada, chamada P_k, que é calculada da seguinte forma:

1. Atribui-se a k um valor igual a metade do tamanho médio dos segmentos de referência;

2. Inicia-se um movimento de uma janela deslizante de tamanho k para computar penalidades;

3. Para cada sentença, determina-se se as sentenças na extremidade da janela deslizante estão em segmentos iguais ou diferentes;

4. Aumenta-se um contador se a segmentação estiver diferente da referência;

5. Divide-se o resultado do contador pelo número de medidas realizadas, modificando sua escala para [0,1].

A métrica P_k é padrão entre pesquisadores da área de segmentação de texto e tem seu funcionamento ilustrado pela Figura 4.3. Nela é possível perceber uma janela de tamanho 2 (metade do tamanho médio de segmento) que é representada em linha pontilhada quando há incremento do contador de penalidade.

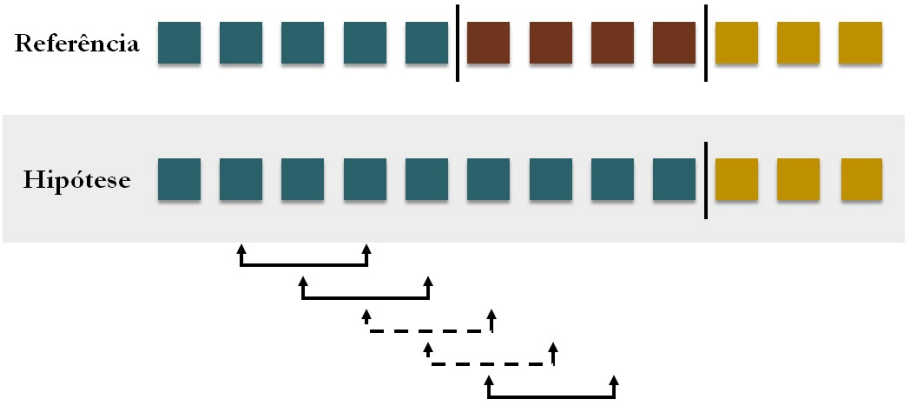

Figura 4.3: Representação do Funcionamento da Métrica P_k na Presença de Falsos Negativos

Apesar de fácil de implementar e de ser melhor para avaliar que a precisão e o *recall*, a métrica P_k também apresenta alguns pontos de crítica (65). O primeiro deles é que falsos negativos são mais penalizados que falsos positivos na maioria dos casos. Analisando a Figura 4.3, é possível perceber que um falso negativo é penalizado k vezes. Isso vale desde que os segmentos em questão tenham tamanho maior ou igual a k. Por outro lado, na Figura 4.4, tem-se falsos positivos em duas posições com $k/2$ e k penalizações. Essa penalização assimétrica não é desejável, pois leva a uma comparação viesada dos segmentadores, privilegiando aqueles com falsos positivos.

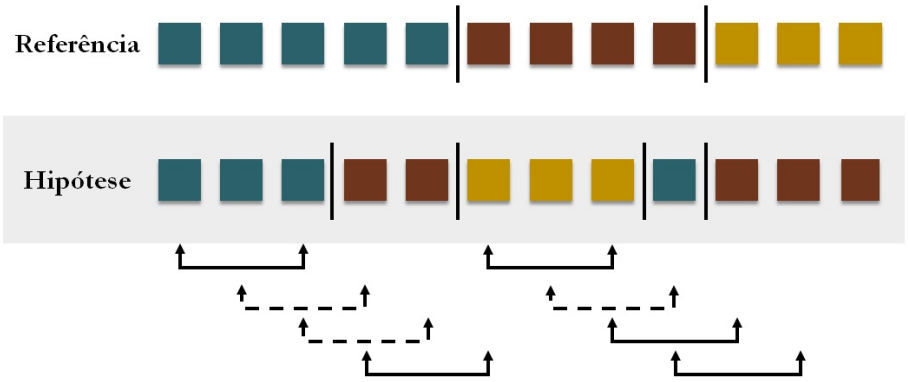

Figura 4.4: Representação do Funcionamento da Métrica P_k na Presença de Falsos Positivos

Outro problema importante com a métrica P_k é que ela permite que alguns erros não sejam penalizados. Em particular, ela não leva em consideração a quantidade de fronteiras em uma janela deslizante. É possível perceber que se o falso positivo estiver a menos de k sentenças das fronteiras verdadeiras mais próximas, ele não será penalizado. A Figura 4.5 ilustra esse fato.

Tudo que foi discutido como fragilidade da métrica P_k é ainda sensível a variações no tamanho do segmento e isso é um problema também. Para fins de discussão, considere que no texto existem dois segmentos, A e B, e o algoritmo não percebeu a fronteira que existe entre eles. Se o $tamanho(A) + tamanho(B) \geq 2k$, tem-se o caso, mostrado na Figura 4.3 e já discutido, em que o falso negativo é penalizado k vezes. A partir disso, a penalização vai cair linearmente com $tamanho(A) + tamanho(B)$ desde que $k \leq tamanho(A) + tamanho(B) < 2k$. Por fim, se $tamanho(A) + tamanho(B) < k$, a penalização desaparece completamente. A Figura 4.6 ilustra esses novos casos.

Considere um segmento A contendo um falso positivo. Se $tamanho(A) > 2k$ e o falso positivo está a distância d da fronteira mais próxima, a penalização será d se $d < k$ e k se $d \geq k$. Conforme $tamanho(A)$ decresça de $2k$ para k,

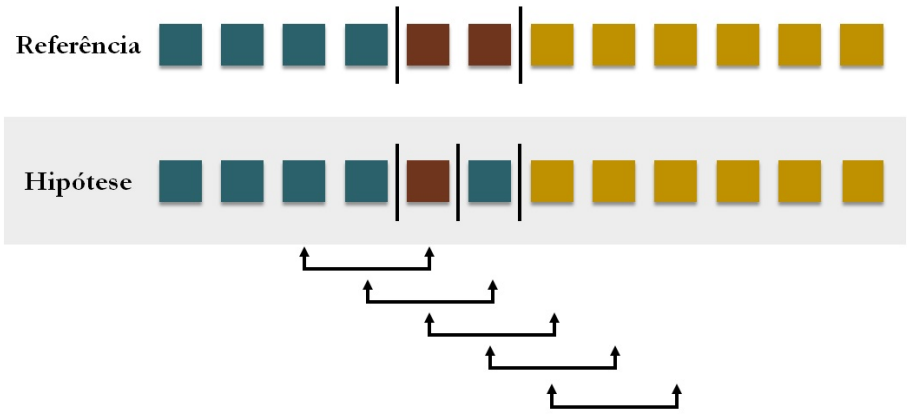

Figura 4.5: Representação de Falsos Positivos que a Métrica P_k é Incapaz de Detectar

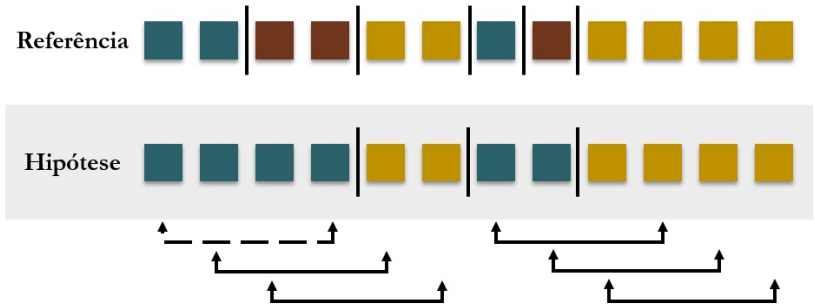

Figura 4.6: Representação do Funcionamento da Métrica P_k na Presença de Falsos Negativos para Diferentes Tamanhos de Segmentos

a máxima penalização possível para um falso positivo é menor do que k, e este número continua a decrescer com o $tamanho(A)$. Esse comportamento é ilustrado na Figura 4.4. Quando $tamanho(A) < k$, a penalização por falso positivo some, identicamente a situação para os falsos negativos.

Um outro problema com a métrica P_k é que apesar de ela penalizar de forma diferenciada os quase acertos, ela ainda os penaliza muito. Considere os segmentadores presentes na Figura 4.7. Cada algoritmo comete pelo menos um erro no posicionamento das fronteiras. Como cada segmentação dessas deve ser penalizada? Para a análise a seguir será considerado que é importante não introduzir fronteiras espúrias.

O algoritmo *SegE* é sem dúvida o pior deles, uma vez que ele tem, simultaneamente, um falso positivo e um falso negativo. Os algoritmos *SegA* e *SegC* contém, respectivamente, um falso negativo e um falso positivo. Comparando os algoritmos *SegB* e *SegD*, é possível perceber que o *SegD* é o melhor, porque ele reconhece que existe apenas uma fronteira em vez de duas,

enquanto o algoritmo *SegB* não só não reconhece isso, como ainda introduz um segmento extra.

Considere, agora, como a métrica P_k trata cada um dos erros apresentado na Figura 4.7. Como era esperado, a segmentação *SegE* é a mais penalizado, recebendo 5 penalizações (3 relativas ao falso negativo e 2 relativas ao falso positivo). As segmentações *SegA* e *SegC* tem cada uma 3 penalizações, devido ao seus falso negativo e falso positivo, respectivamente. Por fim, apesar de *SegD* ser uma segmentação de melhor qualidade, uma vez que um quase acerto é melhor que um falso positivo, ela tem 4 penalizações contra 2 penalizações da segmentação *SegB*.

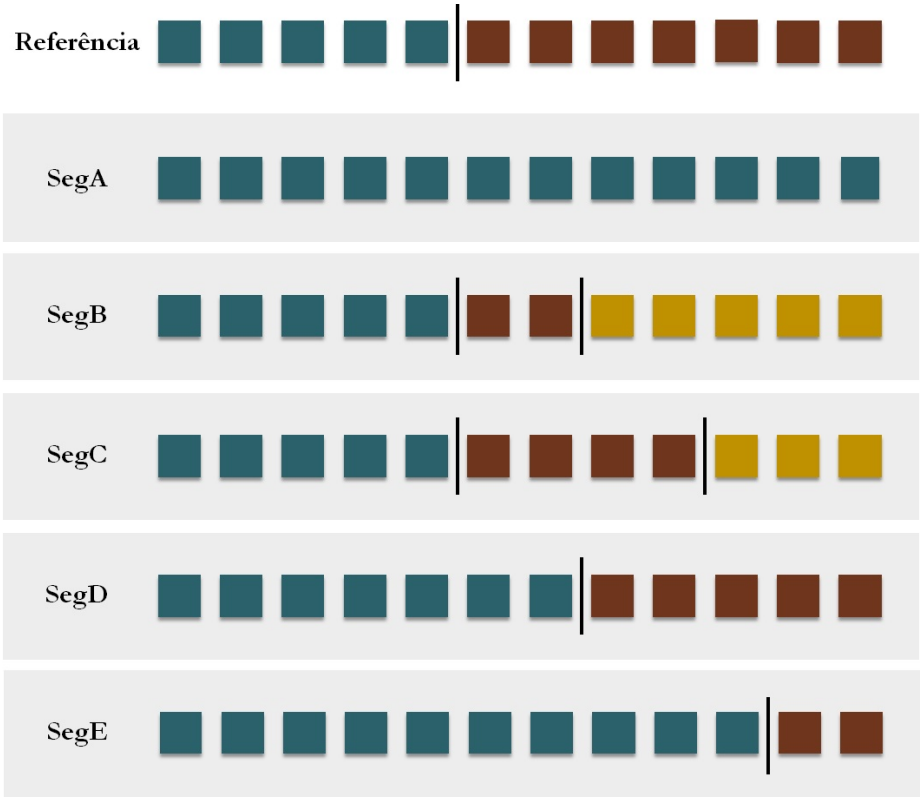

Figura 4.7: Representação de uma Segmentação de Referência e de Cinco Segmentadores

Por fim, a última questão complicadora do uso da métrica P_k é o seu significado. Ela é uma medida indireta da performance, porque ela mede a probabilidade de duas sentenças, com distância k entre si, serem incorretamente categorizadas como pertencentes a segmentos distintos. Apesar de algoritmos perfeitos terem $P_k = 0$, e vários algoritmos degenerados terem $P_k = 0.5$, comparações numéricas são estranhas porque não está claro como é a escala dessa medida.

Buscando resolver esses problemas apresentados acima, elaborou-se uma

nova métrica chamada de *WindowDiff* (65). Ela é calculada com o auxílio de uma janela deslizante de tamanho k, assim como a medida P_k. Para cada posição da janela, são contadas quantas fronteiras da segmentação de referência estão na janela (r_i) e quantas fronteiras foram designadas pelo algoritmo (a_i). O algoritmo é penalizado se $r_i \neq a_i$. Mais formalmente,

$$WindowDiff(ref, hip) = \frac{1}{N-k} \sum_{i=1}^{N-k} (|b(ref_i, ref_(i+k) - b(hip_i, hip_{i+k})| > 0)$$

onde $b(i, j)$ representa o número de fronteiras entre as posições i e j do texto e N representa o número de sentenças no texto.

Apesar de *WindowDiff* ser uma métrica mais adequada e justa de acordo com os critérios apresentados nessa seção, por questões históricas e para permitir comparação com os trabalhos anteriores, é comum que cada trabalho calcule ambas as métricas (P_k e *WindowDiff*).

4.2 Classificação de Texto

O problema de Classificação de Texto é essencial a este trabalho e, portanto, nesta seção é definido formalmente e suas etapas são apresentadas em detalhes.

Considere um documento $d \in \mathbb{X}$, onde $\mathbb{X}$ é o espaço de documentos, e um **conjunto de classes** $\mathbb{C} = \{c_1, c_2, ..., c_n\}$. De uma forma geral, $\mathbb{X}$ é um espaço multidimensional e as classes são definidas por pessoas e variam conforme a aplicação. Como exemplo, pode-se citar as quatro classes que este trabalho utiliza: *Responsabilidades, Requisitos, Benefícios* e *Outros*. A partir de um conjunto de dados de treinamento $\mathbb{D}$ de documentos devidamente classificados (d, c), onde $(d, c) \in \mathbb{X} \times \mathbb{C}$, busca-se um algoritmo de aprendizado capaz de aprender uma **função de classificação** γ que mapeie documentos a classes:

$$\gamma : \mathbb{X} \to \mathbb{C}$$

Esse tipo de aprendizado é chamado de supervisionado porque um supervisor (a pessoa que define as classes e categoriza os documentos) direciona o processo de aprendizagem como um professor (123).

A Figura 4.8 mostra um exemplo de classificação no contexto deste trabalho. Existem quatro classes, cada uma com três documentos de treinamento. São mostradas algumas palavras-chave para representar o conteúdo dos documentos. O conjunto de treinamento provê alguns exemplos típicos de cada classe, de forma a permitir que a função de classificação γ seja aprendida.

Uma vez que ela foi aprendida, ela pode ser aplicada aos dados de teste, por exemplo, o novo documento *Enviar relatórios* cuja classe é desconhecida. Na Figura 4.8, a função de classificação designa o novo documento à classe $\gamma(d)$ =Responsabilidades, que é a classe correta.

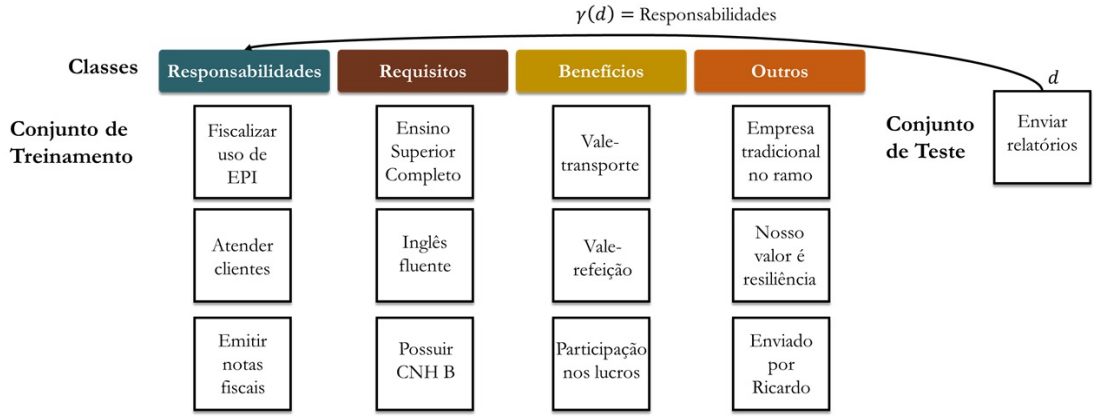

Figura 4.8: Elementos do Processo de Classificação de Texto: Conjunto de Treinamento, Conjunto de Teste, Classificador e Classes

Para treinar um classificador, são necessárias as etapas apresentadas na Figura 4.9. Elas foram brevemente descritas no Capítulo 3, mas são objeto de estudo das subseções a seguir, onde algumas delas são melhores descritas.

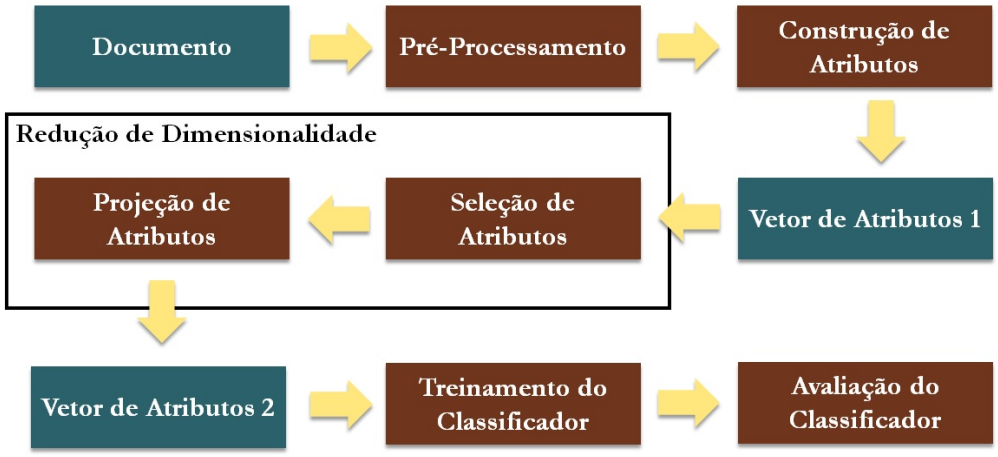

Figura 4.9: Representação do Processo de Classificação de Texto

4.2.1 Pré-processamento

Muito desse processamento que leva a preparação do texto tem suas raízes no processamento de linguagem natural. O pré-processamento de texto inclui (1) unitização e tokenização, (2) normalização e limpeza do texto, (3) remoção

de *stop words* e (4) stemização e lematização (164). A cada passo, informação redundante ou desnecessária é removida do texto original, reduzindo o número de dimensões necessárias para representar o texto. Portanto, existe um ponto ótimo entre informações mantidas e complexidade reduzida nas escolhas feitas ao longo do pré-processamento. De forma simplificada, o pré-processamento é uma etapa que recebe como entrada o texto bruto e retorna *tokens* normalizados.

4.2.1.1 Unitização e Tokenização

A sub-etapa de **unitização** compreende decidir qual unidade do texto se vai analisar. A unidade mais comum para um texto é a palavra, que, de forma prática, pode ser definida como pedaço do texto situado entre dois espaços em branco, ou entre espaço em branco e sinal de pontuação. Outras unidades como sentenças e parágrafos são recorrentes, mas todas essas unidades ainda são baseadas na legibilidade do texto.

Buscando uma forma mais abrangente de definir a unidade do texto a ser analisado, surge o conceito de **n-grama** como uma sequência de itens consecutivos no texto. Nesse contexto, um item pode ser entendido como uma letra, uma sílaba ou uma palavra. Dessa forma, pode-se propor unidades mais complexas que as citadas.

Basicamente, na literatura, existem dois tipos de unidades utilizadas para analisar texto: n-gramas de palavras e n-gramas de caracteres, sendo n o parâmetro a ser decidido pelo estudo. No caso em que $n = 1$, as unidades se tornam a palavra e o caractere, respectivamente. Decidir qual a melhor unidade para analisar um texto é um assunto complexo que neste trabalho é tratado de forma experimental em vez de teórica, ou seja, a melhor unidade é aquela que leva ao melhor resultado do algoritmo de classificação.

De uma forma genérica, a unidade do texto é chamada de *token* e, por isso, a sub-etapa que consiste em dividir o texto em suas unidades básicas é chamada de **tokenização**. Essa sub-etapa é ilustrada na Figura 4.10.

4.2.1.2 Normalização e Limpeza do Texto

Depois de obtidos os *tokens*, é necessário que eles sejam normalizados e limpos. A ideia é tornar comparáveis os termos presentes em cada documento. Por exemplo, deseja-se que *ensino*, *ensino.* e *Ensino* sejam idênticos independente da capitalização das letras e da presença de sinais de pontuação.

A primeira sub-etapa consiste em **uniformizar a capitalização** do texto. Por questões de simplicidade, a solução mais comum é converter o texto inteiramente para minúsculo. Entretanto, deve-se ter cuidado com o uso indistinto dessa técnica, uma vez que a capitalização pode ser um fator importante para a morfologia e a semântica das palavras. Por exemplo, ela permite distinguir substantivos comuns de próprios. Uma solução alternativa, que busca manter a informação sobre a capitalização, consiste em incluir *tokens* específicos para indicar que a palavra inicialmente tinha inicial maiúscula (xxMAJxx) ou era toda composta de letras maiúsculas (xxUPxx), conforme exemplificado na Figura 4.10.

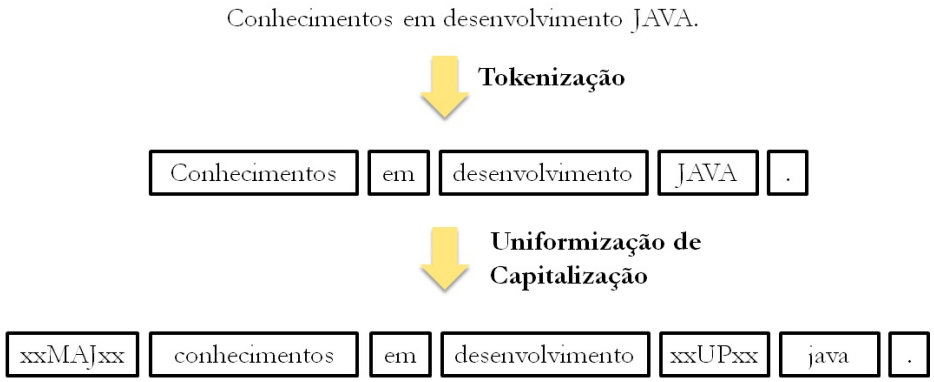

Figura 4.10: Etapas do Pré-processamento de Texto: Tokenização e Uniformização de Capitalização

A sub-etapa seguinte, chamada de **limpeza de caracteres**, remove os caracteres especiais, números e pontuações. Mais uma vez, é preciso cuidado, uma vez que os caracteres removidos podem fazer parte do *token*. Como exemplos, tem-se: (a) P&D, sigla para pesquisa e desenvolvimento, da qual o caractere especial '&' faz parte; e (b) guarda-roupa, ou qualquer palavra composta, da qual o sinal de pontuação hífen faz parte. O mesmo tipo de solução apresentada anteriormente na uniformização de capitalização vale nessa sub-etapa: ao remover um número, um caractere especial ou um sinal de pontuação, incluir um *token* informativo que os represente, como xxNUMxx, xxSPExx e xxPUNCxx, respectivamente.

4.2.1.3 Remoção de *Stop Words*

Existem algumas palavras que, por serem tão comuns, agregam pouco ou nenhum valor para semântica do texto. Essas palavras serão chamadas de ***stop words***. Dado que elas carregam pouca semântica latente, faz sentido

removê-las do texto. De uma forma geral, são consideradas *stop words*: todos os artigos, todas as preposições, todas as conjunções e alguns pronomes (165). Outras palavras podem ser acrescentadas a lista de *stop words* baseada em sua frequência no texto.

Apesar de parecer uma atividade lógica, a remoção de *stop words* também traz perda de informação, sendo uma tendência atual do processamento de texto manter todas as palavras até a etapa de construção de atributos, na qual técnicas permitem que palavras muito frequentes tenham pouco impacto na análise dos documentos (123).

4.2.1.4 Stemização e Lematização

Por questões gramaticais, documentos usam diferentes formas de uma palavra, como *organizar*, *organiza* e *organizado*. Além disso, existem famílias de palavras derivadas com significados semelhantes, como *democracia*, *democrático* e *democratização*. Parece útil, portanto, que ao comparar textos com essas palavras derivadas, eles tenham alguma semelhança apesar de conter formas diferentes de uma mesma palavra(123).

O objetivo da stemização e da lematização é reduzir as formas flexionadas e, as vezes, as formas derivadas de uma palavra a uma forma básica comum. Apesar do objetivo comum, as duas sub-etapas diferem no modo em que resolvem esse problema. A **stemização** é a sub-etapa em que, através de heurísticas, se busca cortar o final das palavras na esperança de atingir o objetivo com alguma incerteza. Isso normalmente funciona bem para as formas flexionadas, mas erra bastante nas formas derivadas (123). Por outro lado, a **lematização** é mais sofisticada, uma vez que em suas heurísticas utiliza também informações sobre a classe morfológica da palavra para decidir sobre a raiz da palavra. Essa sofisticação tem um preço, pois a lematização é um processo extremamente dependente de um analisador morfológico (165).

O algoritmo de stemização mais comum para língua inglesa é o algoritmo de Porter (166), ele é considerado um trabalho pioneiro na área e foi adaptado para uso em várias línguas, inclusive português. É importante salientar que realizar essa sub-etapa em língua portuguesa é mais complicado que em língua inglesa devido a sua morfologia mais complexa (167). Algumas dificuldades relevantes são:

– **Quantidade de exceções** - Para quase toda regra de remoção de sufixos, existe uma exceção. Por exemplo, o sufixo *ão* normalmente representa aumentativo (e.g. porcão), mas nem toda palavra terminada em *ão* está na forma de aumentativo (e.g. campeão);

- **Homógrafos** - Existem diversos casos de palavras que tem a mesma forma escrita, mas significado diferentes. Por exemplo, casais é o plural do substantivo casal, mas também é a segunda pessoa do plural do verbo casar;
- **Verbos Irregulares** - Existem muitos verbos que não se encaixam nos modelos de conjugação verbal e sofrem alterações no radical e na terminação ao serem conjugados. A diferença principal para a língua inglesa, nesse caso, é a quantidade de irregularidades considerando a quantidade de tempos verbais e variedade de conjugação entre as pessoas verbais;
- **Mudanças na Raiz Morfológica** - Existem casos em que o processo de inflexão modifica o radical da palavra. Por exemplo, emitir e emissão são semanticamente relacionadas, mas, após stemização, a primeira se reduz a emit e a segunda se reduz a emis;
- **Nomes Próprios** - Existem muitos nomes próprios em língua portuguesa que também designam outras coisas, como Nogueira, um sobrenome comum, e nogueira, árvore cujo fruto é a noz. Isso é um problema porque nomes próprios não tem sufixos, enquanto a outra coisa designada pode ter.

Considerando essas dificuldades, este trabalho utilizou um algoritmo específico de stemização para língua portuguesa, chamado RSLP (167), e ele foi escolhido por se apresentar como o melhor algoritmo considerando as métricas de *overstemming* (remoção excessiva de caracteres) e *understemming* (remoção precária de caracteres). Esse algoritmo se encontra esquematizado na Figura 4.11 e suas etapas são descritas a seguir.

1. **Redução de Plural** - busca reduzir as palavras ao singular através da remoção do "s" no final das palavras não listadas como exceção. Eventualmente também aplica pequenas modificações, como em *capins*→*capim*;

2. **Redução de Feminino** - busca reduzir as palavras ao masculino através da remoção do "a" no final das palavras com sufixos de feminino listados;

3. **Redução de Aumentativo/Diminutivo/Superlativo** - busca reduzir as palavras para o grau normal através da remoção de sufixos de aumentativo/diminutivo/superlativo listados;

4. **Redução de Advérbio** - busca reduzir os advérbios terminados em "mente" através da remoção dessa terminação no final das palavras não listadas como exceção;

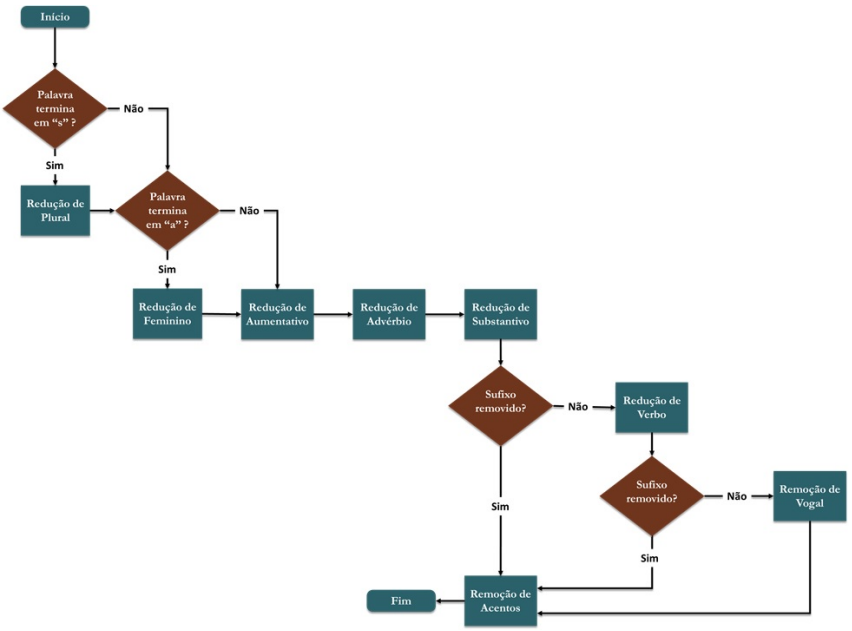

Figura 4.11: Etapas da Stemização com Algoritmo RSLP

5. **Redução de Substantivo/Adjetivo** - busca reduzir os substantivos e adjetivos a seu radical através da remoção de sufixos listados. Se for removido um sufixo nessa etapa, as próximas duas etapas não são executadas;

6. **Redução de Verbo** - busca reduzir os verbos a seu radical através da aplicação de regras para remoção de vogal temática, de sufixos de pessoa e de tempo verbal;

7. **Remoção de Vogal** - busca reduzir ao radical palavras que não tenham sido tratadas nas duas regras anteriores através da remoção de sua última vogal. Um exemplo de palavra que é tratada por essa etapa é *menino* e ela remove a última vogal da palavra, levando-a a *menin*;

8. **Remoção de Acentos** - busca reduzir ao mesmo radical palavras que tenham variação acentuada e não acentuada, como *psicólogo* e *psicologia* que devem ser ambas reduzidas para *psicolog*.

Pensando na outra sub-etapa, a lematização, este trabalho não encontrou nenhum lematizador com resultado satisfatório para os textos de vaga de emprego e, portanto, não apresenta em detalhes nem utiliza nenhum lematizador.

4.2.2 Construção de Atributos

A partir do texto pré-processado é necessário que ele seja transformado de alguma forma em um formato inteligível ao computador. Neste trabalho, somente é abordado o **Modelo de Espaço Vetorial**, entretanto, utilizando atributos léxicos, que são baseados exclusivamente em palavras ou n-gramas, e atributos semânticos, que são baseados em uma semântica latente das palavras e sentenças.

4.2.2.1 Espaço Vetorial Léxico

Essa é a representação mais tradicional de um texto em um espaço vetorial. Ela considera que o texto pode ser descrito pelo seu léxico, ou seja, seus n-gramas de palavras ou caracteres. Cada componente do léxico é chamado de **termo de indexação** (123).

Considere que $\mathbb{D}$ é o conjunto de documentos de treinamento, que t é o número de termos de indexação em $\mathbb{D}$ e que k_i é um termo de indexação genérico. Dessa forma, na representação léxica, o espaço vetorial de documentos $\mathbb{X}$ tem t dimensões e cada dimensão é um termo k_i. Ao conjunto de todos os termos $\mathbb{V} = \{k_1, k_2, ..., k_t\}$ é dado o nome de vocabulário.

Seja um documento $d_i \in \mathbb{D}$ e um vocabulário $\mathbb{V}$. Portanto, a representação vetorial de d_i é $\vec{(d_i)} = (p_1^i, p_2^i, ..., p_t^i)$, onde p_j^i se refere ao peso do termo k_j no documento d_i. Independente da forma de se determinar os pesos p_j^i, esse tipo de representação léxica ignora a ordem que os termos aparecem no texto, por isso também é conhecida como *bag-of-words* (BOW).

Existem diversos métodos para calcular os pesos p_j^i, alguns deles já citadas no Capítulo 3. Para discuti-los didaticamente, este trabalho chama de palavras os termos de indexação daqui em diante. O método mais simples é chamado de **frequência binária** ou **booleana**. Ele indica se uma palavra aparece ou não em um documento, sem considerar a quantidade de vezes que ela aparece e é calculada da seguinte forma:

$$p_j^i = \begin{cases} 1, & \text{se a palavra } k_j \in d_i \\ 0, & \text{caso contrário} \end{cases}$$

A representação binária do documento apresenta como principal desvantagem a incapacidade de evidenciar palavras mais representativas de um documento, pois atribui os mesmos pesos para todas as palavras presentes no documento. Por exemplo, considere o seguinte documento d_1 = "gerenciar equipes de terceirizados, gerenciar equipes de funcionários próprios e gerenciar

a linha de produção da fábrica". Apesar do documento enfaticamente repetir a palavra "gerenciar", indicando as responsabilidades de um gerente, nenhum algoritmo conseguiria perceber isso com o uso da representação binária.

Dessa forma, surge a necessidade de uma ponderação que reflita essa repetição de palavras em um mesmo documento. A **frequência de termos** (TF) vem suprir essa carência da frequência binária. Ela indica quantas vezes uma palavra aparece no documento e é calculada da seguinte forma:

$$p_j^i = \begin{cases} n_j^i, & \text{se a palavra } k_j \text{ aparece } n_j^i \text{ vezes em } d_i \\ 0, & \text{caso contrário} \end{cases}$$

A frequência de termos agrega mais significado a representação do texto, mas ainda carece da habilidade de distinguir palavras com contagens idênticas mas capacidades informativas diferentes. Considere o mesmo documento d_1 supracitado e, por simplificação, considere que as *stop words* não foram removidas. Assim, tanto a palavra "gerenciar" quanto a palavra "de" aparecem três vezes e, portanto, seriam igualmente importantes. Entretanto, é sabido que preposições são bem menos informativas que verbos, uma vez que são mais comuns.

Para estimar o quanto uma palavra é comum na língua, costuma-se utilizar como aproximação o quanto ela é comum no conjunto de documentos $\mathbb{D}$. Um modelo muito usado para representar a frequência relativa de palavras em um conjunto de documentos $\mathbb{D}$ é a **Lei de Zipf**. Ela enuncia que a **frequência de documentos** (DF), ou frequência relativa, da i-ésima palavra mais frequente é proporcional a $1/i$. Baseado nessa lei de potência, é possível definir um novo peso chamado de **frequência inversa de documentos** (IDF), que é calculado da seguinte forma:

$$IDF_i = \log \frac{m}{DF_i} \text{ , onde } m \text{ é o total de documentos em } \mathbb{D}$$

Agregando a frequência inversa de documentos à frequência de termos, obtém-se um modo de ponderação que considera a contagem de palavras em cada documento e sua frequência relativa no conjunto de documentos. Ele é chamado de TF.IDF e, nesse caso, os pesos são calculados da seguinte forma:

$$p_j^i = \begin{cases} n_j^i \log \frac{m}{DF_j}, & \text{se a palavra } k_j \text{ aparece } n_j^i \text{ vezes em } d_i \\ 0, & \text{caso contrário} \end{cases}$$

Apesar de resolver todos os problemas citados anteriormente, a representação TF.IDF discutida tem problemas por utilizar como termo a palavra. Existem algumas palavras compostas e expressões na língua que contém uma ou mais palavras e são destruídas em uma representação baseada na contagem de unigramas de palavras. Como exemplo, tem-se "São Paulo" que caso seja separada passa a designar um santo em vez de uma cidade. Outro exemplo é "Ensino Médio" que caso seja separada perde seu significado de grau de escolaridade para indicar um ensino de qualidade média. Buscando modelar essas palavras e incluir um pouco da ordem das palavras, pode-se tokenizar o documento em n-gramas de palavras.

Uma questão complicadora em todas as representações acima é o tamanho t do vocabulário. Tipicamente, um vocabulário de unigramas de palavras tem dezenas de milhares de palavras e, portanto, um de n-gramas tem a n-ésima potência disso. Isso torna a matriz que relaciona documentos e termos muito grande, apesar de muito esparsa. Existem diversas formas de representar matrizes esparsas de forma computacionalmente eficiente, entretanto é possível também aumentar a dimensão t de forma mais escalável, sem perder poder de representação. Para isso, pode-se tokenizar o texto em n-gramas de caracteres, levando a dimensão máxima a n-ésima potência de uma centena de caracteres, caso se inclua letras maiúsculas, letras minúsculas, letras acentuadas, pontuação, números e caracteres especiais.

Por questões didáticas, esta seção apresenta as formas de representação léxica com um encadeamento lógico de melhorias até a representação TF.IDF de n-gramas de caracteres. Mesmo assim, este trabalho assume que cada uma delas pode ser melhor ou pior dependendo do conjunto de dados, e, portanto, essa determinação de qual a melhor representação deve ser feita de forma empírica.

4.2.2.2 Espaço Vetorial Semântico: *Word Embeddings*

A representação léxica não consegue lidar com questões de polissemia, ou seja, não consegue separar palavras com mesma grafia e sentidos diferentes. Para suplantar isso, busca-se uma representação em um espaço vetorial semântico. Existem diversas formas de solucionar esse problema, algumas envolvendo decomposição de matrizes, outras envolvendo treinamento de redes neurais (165). Este trabalho se utiliza de espaços vetoriais semânticos treinados com as seguintes técnicas de *word embedding* *Word2Vec* e *FastText* e, portanto, somente elas são explicados a seguir.

A ideia principal de qualquer técnica de *word embedding* é encontrar

representações que consigam capturar relações semânticas entre palavras. Por exemplo, seja uma palavra i e sua representação x_i. Focando na relação singular/plural, o espaço vetorial semântico deve ser tal que $x_{\text{pessoa}} - x_{\text{pessoas}} \approx x_{\text{curso}} - x_{\text{cursos}} \approx x_{\text{equipe}} - x_{\text{equipes}}$.

A técnica ***Word2Vec*** é apresentada em (85) e ela consiste em uma rede neural rasa, de apenas duas camadas, treinada para reconstruir contextos a partir de palavras ou palavras a partir de seus contextos. Existem, portanto, duas variações do problema a ser resolvido (168):

1. **Predizer palavras a partir de seu contexto**: a rede neural é treinada para predizer a i-ésima palavra, w_i, em um sentença, utilizando como entrada uma janela de tamanho k ao redor da palavra, a qual é chamada de contexto. Assim, as palavras $w_{i-k}, ..., w_{i-1}, w_{i+1}, ..., w_{i+k}$ são utilizadas para predizer a palavra w_i conforme representado na Figura 4.12. Esse modelo também é chamado de *bag-of-words* contínuo (CBOW);

2. **Predizer contextos a partir de palavras**: a rede neural é treinada para predizer o contexto $w_{i-k}, ..., w_{i-1}, w_{i+1}, ..., w_{i+k}$ ao redor da palavra w_i, utilizando como entrada a i-ésima palavra w_i conforme representado na Figura 4.13. Esse modelo é chamado de *skip-gram*.

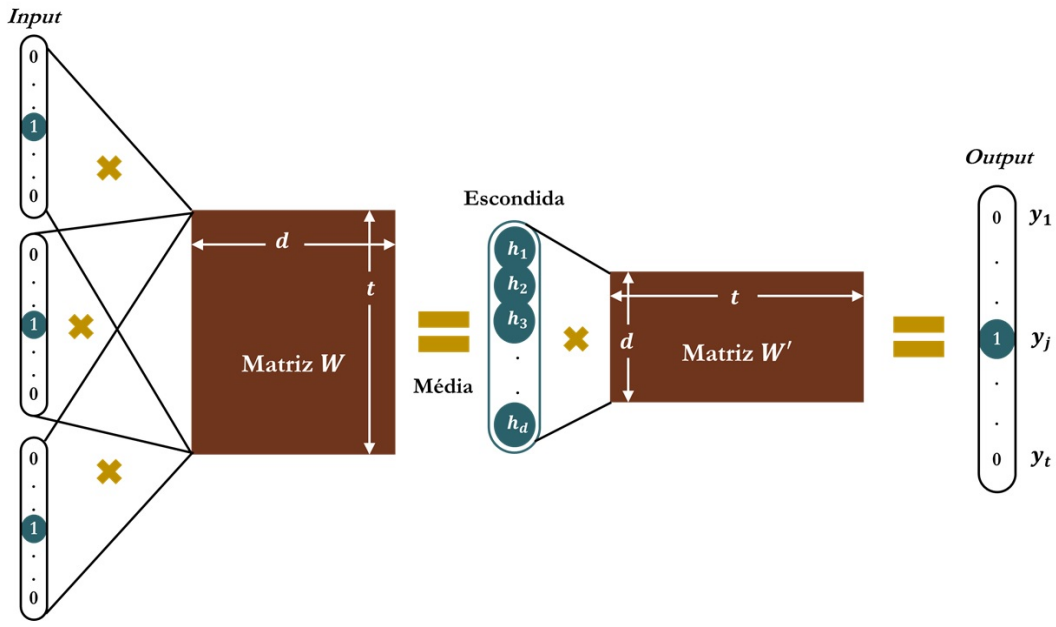

Figura 4.12: Representação de Rede Neural de *Word2Vec* com Modelo CBOW

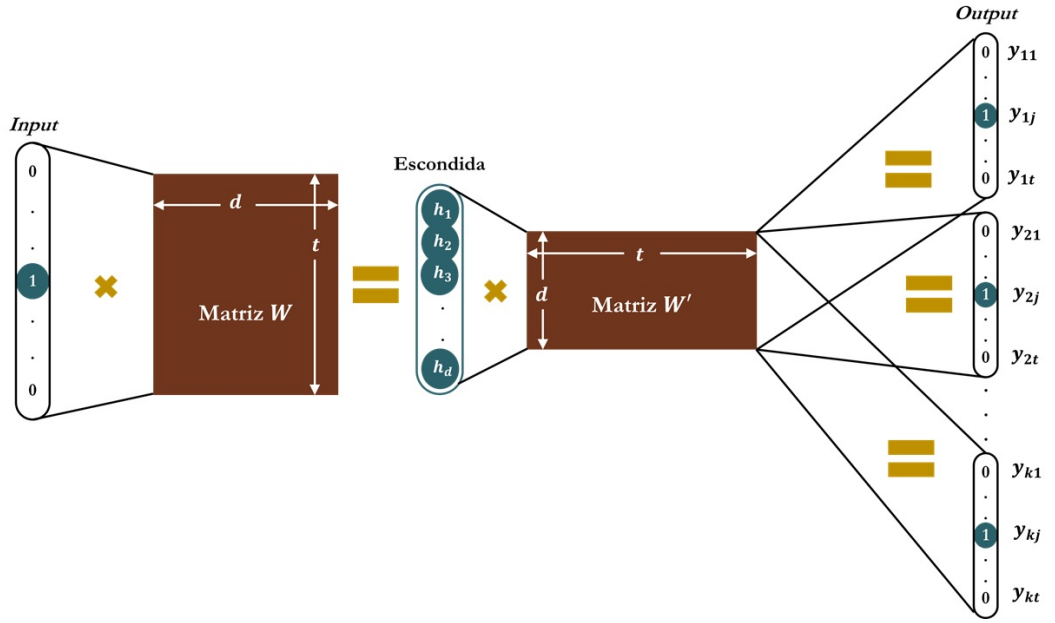

Figura 4.13: Representação de Rede Neural de *Word2Vec* com Modelo *Skip-gram*

Cada um dos modelos é descrito a seguir considerando suas particularidades. Entretanto, sem perda de generalidade e com fins de simplificação, a i-ésima palavra é representada daqui em diante como w e as palavras de contexto são representadas como $\{w_1, w_2, ..., w_k\}$.

Considere que t é o tamanho do vocabulário $\mathbb{V}$, k é o tamanho da janela de contexto e que d é a dimensão escolhida para a representação *Word2Vec*. Nesse caso, para o **modelo CBOW**, a arquitetura geral da rede neural, apresentada na Figura 4.12, contém uma camada de entrada com $k \times t$ nós, uma camada escondida com d nós e uma camada de saída com t nós. Com essa configuração de rede neural, se busca calcular a probabilidade $P(w|w_1, w_2, ...w_k)$ e maximizar o produto dessas probabilidades ao longo de todos os exemplos de treinamento.

Na camada de entrada são utilizadas representações binárias de cada uma das palavras no contexto, por isso, a dimensão é $k \times t$. Dessa forma, é possível representar, sem perda de generalidade, as entradas como $x_{ij} \in \{0, 1\}$, onde $i \in \{1, ..., k\}$ é a posição da palavra na janela de contexto e $j \in \{1, ..., t\}$ é a posição da palavra no vocabulário.

Na camada escondida ou intermediária, existem d nós, sendo d, portanto, a dimensionalidade da representação embutida. As conexões entre cada uma das k palavras de entrada e a camada escondida é definida pela matriz de pesos U, cuja dimensão é $t \times d$.

Sejam $h_1, h_2, ..., h_d$ as saídas dos nós da camada escondida e u_{jq} o peso da conexão entre a j-ésima palavra do vocabulário e o q-ésimo nó da camada escondida. Assim, é possível dizer que $\vec{u_j} = (u_{j1}, u_{j2}, ..., u_{jd})$ é a representação embutida de dimensão d da j-ésima palavra e $\vec{h} = (h_1, h_2, ..., h_d)$ é a representação embutida do contexto provido como entrada. Considerando essa notação, a saída da camada escondida pode ser calculada segundo a seguinte fórmula:

$$h_q = \frac{1}{k} \sum_{i=1}^{k} \left[\sum_{j=1}^{t} u_{jq} x_{ij} \right] \ \forall q \in \{1, ..., d\}$$

A camada escondida $(h_1, h_2, ..., h_d)$ é utilizada para predizer a probabilidade $P(w|w_1, w_2, ..., w_k)$ com o uso de uma função *softmax*. Os pesos camada de saída são calculados com a matriz $U' = [u'_{qj}]$, cuja dimensão é $d \times t$. A camada de saída deve perseguir como gabarito uma representação binária da palavra alvo w_i, simbolizada como $y = (y_1, y_2, ..., y_t)$. Entretanto, após aplicar a função *softmax*, o que ela consegue é um vetor $\hat{y} = (\hat{y_1}, \hat{y_2}, ..., \hat{y_t})$ de valores reais e pertencentes ao intervalo (0,1). Esses valores somam 1 e, portanto, podem ser interpretados como probabilidades. A função *softmax* calcula a probabilidade $P(w_i|w_{i-k}, ..., w_{i-1}, w_{i+1}, ..., w_{i+k})$ da seguinte forma:

$$\hat{y_i} = P(y_i = 1|w_1, w_2, ..., w_k) = \frac{\exp\left(\sum_{q=1}^{d} h_q u'_{qi}\right)}{\sum_{j=1}^{t} \exp\left(\sum_{q=1}^{d} h_q u'_{qj}\right)}$$

De uma forma mais teórica, é possível definir como objetivo para a rede neural maximizar as probabilidades condicionais de cada palavra alvo dado seu contexto. Uma forma de agregar isso numa função única mantendo o objetivo é através do produto dessas probabilidades condicionais. Essa agregação é chamada de função de verossimilhança. Entretanto, buscando simplificar a função de perda e evitar problemas de *underflow*, usa-se o negativo do logaritmo para transformar o produto de probabilidades em uma função de perda aditiva. Assim, para uma palavra particular w_r, $r \in \{1, ..., t\}$, a função de perda é dada por $L = -\log[P(y_r = 1|w_1, w_2, ..., w_k)] = -\log(\hat{y_r})$.

As atualizações das matrizes de peso durante o treinamento são feitas utilizando o algoritmo de *backpropagation*, em que os pesos são atualizados de forma aditiva com o gradiente da função de perda e uma taxa de aprendizado α. Sem entrar em detalhes de etapas intermediárias e definindo $\epsilon_j = y_j - \hat{y_j}$, tem-se que as equações de atualização são dadas por:

$$\vec{u_i} \leftarrow \vec{u_i} + \alpha \sum_{j=1}^{t} \epsilon_j \vec{u'_j} \ [\forall \text{ palavras } i \text{ na janela de contexto}]$$

$$\vec{u_j'} \leftarrow \vec{u_j'} + \alpha\epsilon_j\vec{h} \text{ [}\forall \text{ palavras } j \text{ no vocabulário]}$$

É possível notar que a repetição de uma palavra i na janela de contexto traz múltiplas atualizações para $\vec{u_i}$. Além disso, como $\vec{h}$ faz uma agregação da matriz U, ambas as atualizações dependem dos elementos da matriz U associados as palavras presentes na janela de contexto. Esse tipo de agregação traz um efeito de suavização para o método CBOW, que é particularmente útil para evitar sobre-ajuste de modelos com pequenos conjuntos de dados (165).

Vale salientar que a técnica *Word2Vec* fornece duas representações vetoriais semânticas, as t linhas da matriz U e as t colunas da matriz U'. A primeira é chamada de *embedding* da entrada e a segunda de *embedding* de saída. No método CBOW, como o *embedding* de entrada carrega consigo informações sobre o contexto das palavras, faz mais sentido representar as palavras utilizando o *embedding* de saída.

Encerradas as explicações sobre o modelo CBOW, cabe apresentar as explicações sobre o **modelo *Skip-gram***. A arquitetura geral da rede neural para esse modelo, apresentada na Figura 4.13, contém uma camada de entrada com t nós, uma camada escondida com d nós e uma camada de saída com $k \times t$ nós. Com essa configuração de rede neural, se busca calcular a probabilidade $P(w_1, w_2, ..., w_k|w)$.

Na camada de entrada são utilizadas também representações binárias de cada uma das palavras, por isso, a dimensão é t. Dessa forma, é possível representar, sem perda de generalidade, as entradas como $x_j \in \{0, 1\}$ onde $j \in \{1, ..., t\}$ é a posição da palavra no vocabulário.

Na camada escondida ou intermediária, existem d nós, sendo d, portanto, a dimensionalidade da representação embutida. As conexões entre a palavra de entrada e a camada escondida é definida pela matriz de pesos U, cuja dimensão é $t \times d$, assim como no modelo CBOW.

Considerando as notações $\vec{u_j}$ e $\vec{h}$, introduzidas na apresentação do modelo CBOW, a saída da camada escondida para o modelo *Skip-gram* pode ser calculada segundo a seguinte fórmula:

$$h_q = \sum_{j=1}^{t} u_{jq}x_j \ \forall q \in \{1, ..., d\}$$

A camada escondida $(h_1, h_2, ..., h_d)$ é utilizada para predizer a probabilidade $P(w_1, w_2, ..., w_k|w)$ com o uso de uma função *softmax*. Os pesos camada de saída são calculados com a matriz $U' = [u'_{qj}]$, cuja dimensão é $d \times t$. A camada de saída deve perseguir como gabarito representações binárias das palavras $\{w_{i-k}, ..., w_{i-1}, w_{i+1}, ..., w_{i+k}\}$ do contexto alvo , simbolizadas como

$y_{ij} \in \{0, 1\}$, onde $i \in \{1, ..., k\}$ é a posição da palavra na janela de contexto e $j \in \{1, ..., t\}$ é a posição da palavra no vocabulário. Entretanto, após aplicar a função *softmax*, o que ela consegue é um vetor $\hat{y_{ij}}$ de valores reais e pertencentes ao intervalo (0,1). Esses valores somam 1 para cada valor de i e, portanto, podem ser interpretados como probabilidades. Como a matriz U' é compartilhada por cada representação binária das palavras de contexto, a rede neural prevê a mesma distribuição multinomial para cada palavra de contexto e, portanto, $\hat{y_{ij}}$ é calculado da seguinte forma:

$$\hat{y_{ij}} = P(y_{ij} = 1|w) = \frac{\exp\left(\sum_{q=1}^{d} h_q u'_{qj}\right)}{\sum_{l=1}^{t} \exp\left(\sum_{q=1}^{d} h_q u'_{ql}\right)} \ \forall i \in \{1, ..., k\}$$

Note que a probabilidade $\hat{y_i}j$ independe da localização i da palavra na janela de contexto, uma vez que o lado direito da equação acima não depende de i.

A função de perda nesse modelo é o negativo do logaritmo da função de verossimilhança dos valores de gabarito $y_{ij} \in \{0, 1\}$ de um exemplo do conjunto de treinamento. Essa perda L é dada pela seguinte expressão, que é a forma geral da função de perda apresentada para o modelo CBOW:

$$L = -\sum_{i=1}^{k}\sum_{j=1}^{t} y_{ij} \log(\hat{y_{ij}})$$

Como y_{ij} para um determinado i é a representação binária da palavra de contexto na posição i, a função de perda tem apenas k termos diferentes de zero.

Utilizando o algoritmo de *backpropagation* e definindo $\epsilon_{ij} = y_{ij} - \hat{y_{ij}}$, tem-se que as equações de atualização são dadas por:

$$\vec{u_r} \leftarrow \vec{u_r} + \alpha \sum_{j=1}^{t} \left[\sum_{i=1}^{k} \epsilon_{ij}\right] \vec{u'_j} \text{ [apenas para a palavra de entrada } r]$$

$$\vec{u'_j} \leftarrow \vec{u'_j} + \alpha \left[\sum_{i=1}^{k} \epsilon_i j\right] \vec{h} \text{ [}\forall \text{ palavras } j \text{ no vocabulário]}$$

O modelo *Skip-gram* é recomendado para textos mais longos, uma vez que não tem o mesmo efeito de suavização do modelo CBOW. Esse efeito impede o sobre-ajuste, mas também impede o modelo de tirar proveito da grande quantidade de dados em textos mais longos (165).

Apesar disso, a atualização da matriz U' é computacionalmente cara e, por isso, formas alternativas de se resolver o problema foram encontradas. Uma

delas, chamada de **modelo *Skip-gram* com Amostragem Negativa** (em inglês, *Skip-gram Negative Sampling* - SGNS), contorna esse custo computacional modificando levemente a filosofia do modelo *Skip-gram*.

Primeiramente, o modelo SGNS busca prever se cada uma das t palavras do vocabulário está presente na janela de contexto, em vez de tentar prever quais são as k palavras que estão presentes. Essa mudança traz alterações imediatas para a lógica de treinamento, para a arquitetura da rede neural e para a função de perda.

Enquanto o modelo *Skip-gram* utiliza no treinamento apenas os pares palavra-contexto presentes no texto, no modelo SGNS são utilizados tanto os pares presentes, chamados de pares positivos, quanto os ausentes no texto, chamados de pares negativos. Para gerar esses pares negativos, palavras que não pertencem ao contexto são geradas artificialmente através de amostragem no vocabulário $\mathbb{V}$. A técnica de amostragem para gerar a amostra negativa não é abordada neste trabalho.

Para lidar com a mudança de filosofia, a arquitetura da rede neural também é modificada. A camada de saída, em vez de ser uma função *softmax*, passa a ser um conjunto de $t \times k$ neurônios com função de ativação sigmoide. Dessa forma, a rede neural passa a prever, de forma independente, qual a probabilidade de cada palavra do vocabulário pertencer ou não a uma determinada posição da janela de contexto.

Por fim, a função de perda também é modificada para refletir o novo objetivo perseguido pela rede neural, considerando os pares positivos e negativos. Seja $\mathcal{P}$ o conjunto de pares positivos em um determinada janela de contexto e $\mathcal{N}$ o conjunto de pares negativos criado através de amostragem. A função de perda, nesse caso, é dada pela seguinte expressão:

$$L = -\sum_{(i,j)\in\mathcal{P}} \log\left(\frac{1}{1+\exp(-\vec{u_i}\cdot\vec{u_j}')}\right) - \sum_{(i,j)\in\mathcal{N}} \log\left(\frac{1}{1+\exp(\vec{u_i}\cdot\vec{u_j}')}\right)$$

O modelo SGNS é a variação do modelo *Skip-gram* que é mais eficiente e tem melhores resultados (165). Apesar disso, modelos *Word2Vec* tem uma limitação fundamental que é a incapacidade de lidar com palavras que não tenham aparecido no conjunto de treinamento. Isso se dá porque todas suas implementações modelam cada palavra do vocabulário como um vetor embutido distinto, ignorando a estrutura interna das palavras.

Buscando superar isso, é proposto o **modelo *FastText***, que acrescenta informações a respeito dos n-gramas de caracteres à representação semântica da palavra. Isso o torna especialmente relevante para o uso em linguagem

morfologicamente ricas, como o português. Com essa adaptação é agregada ao modelo semântico informação sobre sufixos, prefixos e radicais, permitindo que palavras inéditas sejam representadas através da junção das representações dos seus n-gramas de caracteres.

Assim, o vocabulário $\mathbb{V}$ passa a conter t elementos únicos, juntando as palavras e seus n-gramas. A representação semântica de uma palavra com o modelo FastText é dada pela soma das representações semânticas de seus n-gramas e da sequência especial de caracteres correspondente a própria palavra. Por exemplo, considerando $n = 3$, representação semântica da palavra *supervisão* é dada pela soma da representação semântica das seguintes sequências de caracteres:

$$\langle \text{su, sup, upe, per, erv, rvi, vis, isã, são, ão, supervisão} \rangle$$

4.2.3 Treinamento de Classificador

Após a construção dos atributos, é necessário que o conjunto de dados seja submetido a um algoritmo de aprendizado para treinamento. Neste trabalho, são considerados apenas algoritmos de aprendizado supervisionados, o que implica que cada sentença é submetida em conjunto com sua classe para que o algoritmo consiga aprender uma função de classificação γ, conforme representado na Figura 4.8.

Treinar um classificador para que ele seja capaz de generalizar a tarefa para fora dos dados de treinamento requer alguns cuidados, como dividir o conjunto de dados em treino, validação e teste. Eles são melhor discutidos nos capítulos subsequentes.

Essa subseção apresenta fundamentos teóricos das versões mais básicas, sem regularização, dos quatro algoritmos clássicos estudados neste trabalho: Naïve Bayes, Regressão Logística Multinomial, Florestas Aleatórias e Máquinas de Vetores de Suporte (SVM). Isso é essencial para entender as premissas assumidas por cada algoritmo e suas limitações. Entretanto, excede ao escopo deste trabalho apresentar a teoria completa sobre cada um dos algoritmos.

4.2.3.1 Naïve Bayes

O primeiro classificador estudado neste trabalho é o Naïve Bayes Multinomial. Isso tem uma motivação histórica, uma vez que ele é um dos mais antigos algoritmos, sendo estudado desde o começo da década de 60 (170).

Para entender seu funcionamento, é importante entender porque o algoritmo é *naïve* e porque ele é bayesiano. Seja uma sentença de vaga de emprego $\vec{d}$ formada pelas n_d palavras $(w_1, ..., w_{n_d})$ e seja $\mathbb{C} = \{c_1, c_2, ..., c_n\}$ o conjunto de todas as classes. Para fins didáticos, seguindo a lógica apresentada em (170), considere inicialmente que $n_d = 1$. Nesse caso, a probabilidade da sentença $\vec{d} = w_1$ pertencer a classe c_j é dada por:

$$P(c_j|w_1) = \frac{P(c_j) \times P(w_1|c_j)}{P(w_1)} = k \times P(c_j) \times P(w_1|c_j)$$

onde $P(c_j)$ é a probabilidade a priori de uma sentença ser classificada como c_j, $P(w_1|c_j)$ é a probabilidade de uma sentença conter a palavra w_1 dado que ela foi classificada como c_j e k é uma constante que funciona como fator de escala.

A equação acima é chamada de Teorema de Bayes e é a base desse algoritmo, por isso ele é bayesiano. Prosseguindo com as considerações, com $n_d = 2$, a probabilidade da sentença $\vec{d} = (w_1, w_2)$ pertencer a classe c_j passa a ser dada por:

$$P(c_j|w_1 \cdot w_2) = \frac{P(c_j) \times P(w_1|c_j) \times P(w_2|c_j \cdot w_1)}{P(w_1) \times P(w_2|w_1)}$$

Novamente o denominador é constante e, abusando um pouco da notação, também vai ser referenciado como $1/k$, levando a seguinte expressão para a probabilidade condicional da sentença $\vec{d}$ pertencer a classe c_j:

$$P(c_j|w_1 \cdot w_2) = k \times P(c_j) \times P(w_1|c_j) \times P(w_2|c_j \cdot w_1)$$

Assumindo que, para uma dada classe c_j, duas palavras são independentes, a equação acima se reduz a:

$$P(c_j|w_1 \cdot w_2) = k \times P(c_j) \times P(w_1|c_j) \times P(w_2|c_j)$$

A premissa de independência é reconhecidamente falsa, entretanto traz uma simplificação que facilita a computação das medidas envolvidas. Por assumir uma premissa falsa, o algoritmo é dito ingênuo ou *naïve* (em inglês).

A partir do que foi discutido, por indução, pode-se dizer que para o caso geral em que $\vec{d} = (w_1, ..., w_{n_d})$, a probabilidade da sentença $\vec{d}$ pertencer a classe c_j é dada por:

$$P(c_j|w_1 \cdot w_2 \cdot ... \cdot w_{n_d}) = k \times P(c_j) \times \prod_{i=1}^{n_d} P(w_i|c_j)$$

Na classificação de texto, o objetivo é encontrar a melhor classe para cada

documento e no caso do algoritmo Naïve Bayes ela é chamada de classe mais provável ou classe máxima a posteriori (MAP) (123). Essa classe é determinada da seguinte forma:

$$c_{MAP} = \underset{c_j \in \mathbb{C}}{\operatorname{argmax}}\, \hat{P}(c_j|\vec{d}) = \underset{c_j \in \mathbb{C}}{\operatorname{argmax}} \left[\hat{P}(c_j) \times \prod_{i=1}^{n_d} \hat{P}(w_i|c_j) \right]$$

Mais uma vez, para evitar *underflow* e simplificar as operações, é possível aplicar o logaritmo na expressão acima e, com isso, c_{MAP} passa a ser dada por:

$$c_{MAP} = \underset{c_j \in \mathbb{C}}{\operatorname{argmax}} \left[\log \hat{P}(c_j) + \sum_{i=1}^{n_d} \log \hat{P}(w_i|c_j) \right]$$

Resta determinar uma forma de estimar $\hat{P}(c_j)$ e $\hat{P}(w_i|c_j)$. Uma primeira ideia é usar um estimador de máxima verossimilhança (em inglês, *maximum likelihood estimator* - MLE), que, no caso, vai ser dado simplesmente pela frequência relativa e corresponde ao valor mais provável de cada parâmetro dado o conjunto de dados de treinamento. Para a probabilidade a priori, essa estimativa é:

$$\hat{P}(c_j) = \frac{m_{c_j}}{m}$$

onde m_{c_j} é número total de sentenças da classe c_j e m é o número total de sentenças.

Para a probabilidade condicional $\hat{P}(w_i|c_j)$, o estimador de máxima verossimilhança é dado por:

$$\hat{P}(w_i|c_j) = \frac{T(w_i, c_j)}{\sum_{w \in \mathbb{V}} T(w, c_j)}$$

onde $T(w, c)$ é o total da palavra w em sentenças pertencentes a classe c e $\mathbb{V}$ é o vocabulário.

O problema com o MLE é que ele estima zero na probabilidade condicional para combinações palavra-classe que não apareceram no conjunto de treinamento e nunca o conjunto de treinamento vai ser grande o suficiente para representar adequadamente a frequência de palavras raras (123). Para resolver isso, basta suavizar a estimativa de máxima verossimilhança da seguinte forma (171):

$$\hat{P}(w_i|c_j) = \frac{\alpha + T(w_i, c_j)}{\alpha n_d + \sum_{w \in \mathbb{V}} T(w, c_j)}$$

onde α é chamado de parâmetro de suavização.

4.2.3.2 Regressão Logística

A Regressão Logística é um método, que apesar de ter sua origem nos primórdios da década 30, se difundiu na década de 60 (173) para estimar o valor de uma variável dependente categórica a partir de uma função das variáveis independentes. Ela é uma extensão natural da regressão linear para o caso da variável dependente discreta e apesar de ser chamada de regressão, ela é usada para resolver problemas de classificação.

Para entender seu funcionamento, segue-se a lógica de partir do modelo de regressão binária para o multinomial. Considere uma sentença de vaga de emprego $\vec{d}$ e sua representação vetorial $(p_1, p_2, ..., p_d)$, com d sendo o tamanho do vocabulário $\mathbb{V}$. Por simplificação, sem perda de generalidade, considere que as classes são representadas através de uma variável dependente $\vec{y} = (y_1, y_2, ..., y_n)$, tal que se a sentença pertence a classe c_j, então $y_j = 1$.

É possível definir uma grandeza, chamada de *odds ratio* (OR), como sendo a razão entre a probabilidade de acontecer um evento sobre a probabilidade de não ocorrer esse evento (174). Partindo do problema binário de classificação, com classes c_1 e c_2, e tomando como base a classe c_1, é possível calcular a OR da seguinte forma:

$$OR = \frac{P(c_1|\vec{d})}{1 - P(c_1|\vec{d})} = \frac{P(c_1|\vec{d})}{P(c_2|\vec{d})}$$

Utilizando o Teorema de Bayes nas definições das probabilidades condicionais acima, tem-se que:

$$OR = \frac{P(c_1) \times P(\vec{d}|c_1)}{P(c_2) \times P(\vec{d}|c_2)}$$

A Regressão Logística assume como premissa que o logaritmo da *odds ratio* depende linearmente das variáveis independentes, ou seja, dos pesos p_i de cada palavra na representação vetorial da sentença (147). Nesse sentido, esse modelo é uma extensão da regressão linear. Representando essa premissa em forma de equação:

$$\log(OR) = \beta_0 + \sum_{i=1}^{d} \beta_i p_i = \vec{\beta} \cdot \vec{d} \rightarrow \log\left[\frac{P(c_1) \times P(\vec{d}|c_1)}{P(c_2) \times P(\vec{d}|c_2)}\right] = \vec{\beta} \cdot \vec{d}$$

$$\rightarrow \frac{P(c_1) \times P(\vec{d}|c_1)}{P(c_2) \times P(\vec{d}|c_2)} = \exp(\vec{\beta} \cdot \vec{d})$$

Retomando o Teorema de Bayes para calcular $P(c_1|\vec{d})$, após alguma manipulação algébrica e considerando a equação acima, encontra-se a expressão clássica do modelo utilizando a função sigmóide $\sigma(.)$:

$$P(c_1|\vec{d}) = \frac{1}{1 + \exp\left(-\beta \cdot \vec{d}\right)} = \sigma\left(\beta \cdot \vec{d}\right)$$

Resta expandir os conceitos para o caso multinomial. Para isso, é necessário analisar o comportamento das probabilidades condicionais $P(c_1|\vec{d})$ e $P(c_2|\vec{d})$ considerando que $\vec{\beta}$ pode ser escrito da seguinte forma $\vec{\beta_2} - \vec{\beta_1}$. Assim, as probabilidades condicionais passam a ser dadas por:

$$P(c_1|\vec{d}) = \frac{\exp(\vec{\beta_1} \cdot \vec{d})}{\exp(\vec{\beta_1} \cdot \vec{d}) + \exp(\vec{\beta_2} \cdot \vec{d})}$$

$$P(c_2|\vec{d}) = \frac{\exp(\vec{\beta_2} \cdot \vec{d})}{\exp(\vec{\beta_1} \cdot \vec{d}) + \exp(\vec{\beta_2} \cdot \vec{d})}$$

Assim, de uma forma geral, pode-se dizer que a probabilidade condicional $P(c_j|\vec{d})$ é dada por:

$$P(c_j|\vec{d}) = \frac{\exp(\vec{\beta_j} \cdot \vec{d})}{\sum_{i=1}^{n} \exp(\vec{\beta_i} \cdot \vec{d})}$$

De posse desse modelo, que é chamado de Regressão Logística Multinomial, a classe mais provável para cada sentença é aquela com maior probabilidade $P(c_j|\vec{d})$. Para encontrar os valores ótimos para os vetores de parâmetros $\vec{\beta_j}$ é utilizado o método do gradiente descendente minimizando uma função erro do tipo *cross-entropy*, entretanto, discutir maiores detalhes sobre isso foge ao escopo deste trabalho.

4.2.3.3 Máquinas de Vetores de Suporte (SVM)

O SVM é um método mais moderno que os dois anteriormente apresentados. Ele é utilizado para classificar conjuntos de dados linearmente separáveis, ou seja, existe uma reta capaz de separar os dados em suas classes. Esse algoritmo não busca qualquer separador linear, mas sim aquele que esteja a máxima distância de qualquer um dos dados. A distância entre o dado mais próximo e o separador encontrado é chamada de margem do classificador. Esse modo de buscar um separador é dependente de uma parcela dos dados, que é utilizada explicitamente na definição de sua posição. Esses dados são chamados de vetores de suporte e daí vem o nome do algoritmo (123).

Esse algoritmo é complexo de explicar, sendo necessário lançar mão de uma série de conceitos de Álgebra Linear e Geometria Analítica, que foge ao escopo deste trabalho. Para este trabalho, basta entender que a versão básica do SVM busca um hiperplano de decisão linear e, portanto, só tem bom resultado em conjuntos de dados linearmente separáveis. Além disso, é importante também entender que, apesar de ele ser pensado para um problema de classificação binário, ele pode ser utilizado para problemas não binários caso seja usada uma estratégia de um contra todos. Com ela, são treinados modelos para separar cada uma das classes de todas as outras. Isso implica um aumento no custo computacional, uma vez que para um problema com quatro classes, como o considerado neste trabalho, ele teria que treinar quatro classificadores em vez de um só.

4.2.3.4 Floresta Aleatória

Para entender a Floresta Aleatória, que é o método mais novo entre os apresentados, é importante ter uma breve noção de Árvores de Decisão, afinal o método tem esse nome porque sua previsão é um agregado da previsão de várias árvores de decisão.

A árvore de decisão é um grafo em forma de árvore, cujos nós internos são testes, cada ramificação é um resultado do teste e cada nó final, chamado de nó folha, representa uma classe. Os caminhos da raiz até o nó folha representam regras de classificação.

Existem diversas maneiras de induzir uma árvore de decisão a partir de um conjunto de dados, algumas focam no ganho de informação agregado por dividir os dados em um determinado nó, outras focam em uma medida de impureza, que avalia a capacidade de separar os dados em cada teste.

A Floresta Aleatória, de uma forma geral, induz várias árvores de decisão a partir de subconjuntos de atributos dos dados de treinamento. Ele introduz aleatoriedade ao gerar os subconjuntos de atributos aleatoriamente. Portanto, para cada árvore de decisão que compõe a Floresta Aleatória, apenas um subconjunto aleatório dos atributos é considerado.

4.2.4 Avaliação de Classificador

Antes de discutir as medidas de avaliação de um classificador, é importante apresentar alguns conceitos:

– **Verdadeiro Positivo** (VP_j) para classe c_j - mede a quantidade de sentenças classificadas como positivas corretamente, ou seja, o classificador

afirmou que a classe da sentença é c_j e a classe da sentença é c_j;

- **Verdadeiro Negativo** (VN_j) para classe c_j - mede a quantidade de sentenças classificadas como negativas corretamente, ou seja, o classificador afirmou que a classe da sentença não é c_j e e a classe da sentença não é c_j;
- **Falso Positivo** (FP_j) para classe c_j - mede a quantidade de sentenças classificadas como positivas incorretamente, ou seja, o classificador afirmou que a classe da sentença é c_j e a classe da sentença é $c_i, i \neq j$;
- **Falso Negativo** (FN_j) para classe c_j - mede a quantidade de sentenças classificadas como negativas incorretamente, ou seja, o classificador afirmou que a classe da sentença não é c_j e a classe da sentença é c_j.

Baseado nisso é possível definir as três principais métricas utilizadas para avaliar classificadores. A primeira delas, a **acurácia**, avalia a capacidade do modelo de classificar corretamente as sentenças e é dada por:

$$Acc = \frac{\sum_{i=1}^{n} VP_i}{VP_j + VN_j + FN_j + FP_j}, \forall j \in \{1, ..., n\}$$

Essa medida requer uma atenção especial, uma vez que pode trazer bons resultados com classificadores degenerados quando há desbalanceio entre as classes. Um exemplo disso é um conjunto de dados em que 99% das sentenças pertencem a classe c_0 e o restante a classe c_1. Nesse caso, um classificador que atribui toda sentença a classe c_0 tem 99% de acurácia apesar de degenerado.

A segunda métrica considerada, a **precisão** para a classe c_j, avalia percentualmente quantas sentenças foram previstas como positivas sendo positivas e é dada por:

$$Prec_j = \frac{VP_j}{VP_j + FP_j}, \forall j \in \{1, ..., n\}$$

Com essa métrica, é possível perceber que o classificador degenerado apresentado acima não é bom, pois $Prec_0 = 99\%$, mas $Prec_1 = 0\%$, ou seja, o classificador não consegue classificar corretamente nenhuma sentença da classe c_1.

A terceira métrica considerada, o ***recall*** para a classe c_j, avalia percentualmente quantas sentenças positivas conseguiram ser previstas pelo algoritmo e é dada por:

$$Rec_j = \frac{VP_j}{VP_j + FN_j}, \forall j \in \{1, ..., n\}$$

Com essa métrica, também é possível perceber que o classificador degenerado apresentado acima não é bom, pois $Rec_0 = 1$, mas $Rec_1 = 0\%$, ou seja, o

classificador não consegue classificar corretamente nenhuma sentença da classe c_1, mas classifica corretamente todas as sentenças da classe c_0.

Neste trabalho, foi utilizada a acurácia como métrica principal de comparação dos algoritmos apesar do desbalanceio entre as classes. Isso foi possível porque sempre em conjunto com ela, foram analisadas também as outras duas métricas para cada classe e foi observado que tanto a precisão quanto o *recall* estavam sempre consistentes com a acurácia encontrada. Por isso, daqui em diante, a única métrica apresentada e citada é a acurácia dos modelos nos conjuntos de treino, validação e teste.

5
Implementação

Este trabalho foi implementado em Python 3.6 utilizando dois ambientes:

- Um ambiente de desenvolvimento Anaconda instalado em um Notebook com CPU Intel Core i7 @ 2.7GHz e 2.9 GHz, memória RAM de 16 Gb;
- Um ambiente de desenvolvimento configurado no Google Colab com CPU Intel Core Xeon @ 2.3Ghz, memória RAM de 13 Gb, GPU Tesla K80 com 12 Gb de memória.

5.1
Coleta de Dados

Para ser possível treinar um classificador utilizando aprendizado supervisionado, foi visto que é necessário que o conjunto de dados tenha um gabarito. Assim, diversos *sites* de vagas de emprego foram estudados para se encontrar algum cuja descrição seguisse uma regra ou padrão, de forma que a coleta automática também gerasse um gabarito da classificação de suas sentenças. Foram consideradas as quatro classes apresentadas no Capítulo 2: Responsabilidades, Requisitos, Benefícios e Outros. Outras unidades de segmentação poderiam ter sido escolhidas, entretanto uma análise dos textos das vagas de emprego permitiu perceber que normalmente não havia mais de uma classe por sentença.

De uma forma geral, o *site* nais adequado a necessidade foi o Catho e ele foi usado para gerar o conjunto de dados principal utilizado neste trabalho. Além dele, foram utilizados os *sites* LinkedIn e VAGAS.com.br para construir um conjunto de dados para teste dos algoritmos treinados com o conjunto de dados principal e apontar uma evidência de generalização. Esses últimos, entretanto, tiveram a elaboração de gabarito manual, pois não havia regra geral automatizável e por isso se apresentam também são menores conjuntos de dados.

A coleta de dados em cada um dos *sites* de vagas de emprego foi feita através de programas que automaticamente percorriam as vagas e coletavam suas informações. Esse tipo de programa é chamado de *crawler*, *robot* ou *spider* (169) e é específico para um determinado *site*. Portanto, a seguir estão

detalhados a estrutura geral de um *crawler* e as especificidades dos *crawlers* elaborados para os *sites* Catho, LinkedIn e VAGAS.com.br.

5.1.1 Estrutura Geral do *Crawler*

Neste trabalho, o *crawler* é um programa que além de navegar automaticamente pelas páginas da internet coletando informações, também as salva em um arquivo no formato *csv* para posterior utilização das informações. Considerando isso, a estrutura geral do *crawler* está ilustrada na Figura 5.1, com ênfase em amarelo nas etapas que são discutidas em mais detalhes.

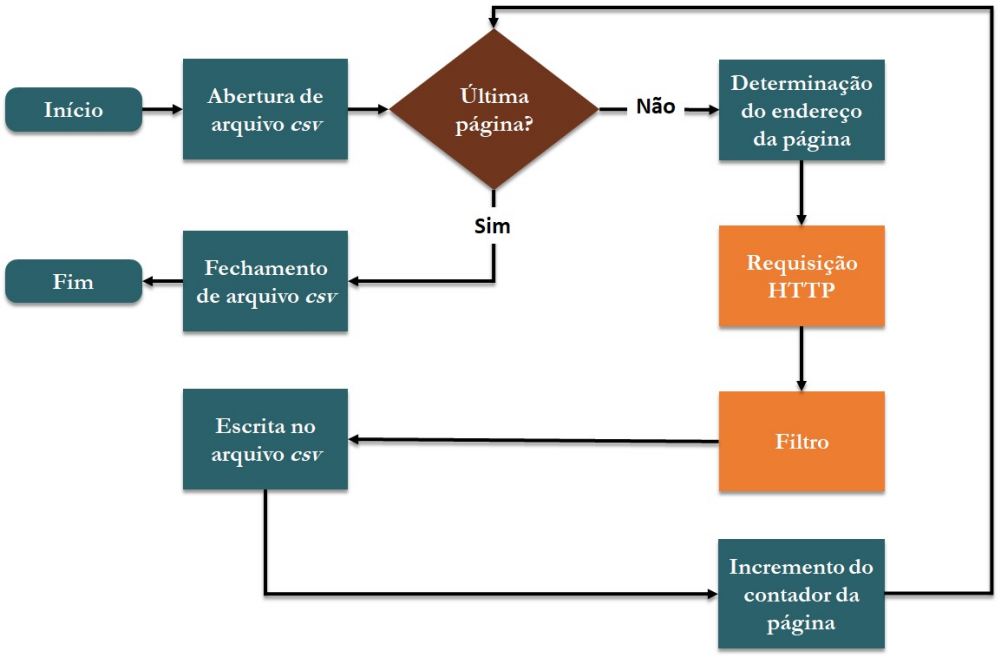

Figura 5.1: Representação de Estrutura Geral de um *Crawler*

De uma forma breve, as explicações das etapas mais simples do *crawler* estão apontadas a seguir:

1. Para abertura e fechamento do arquivo *csv* é utilizada a função nativa ***open()*** associada a uma declaração ***with*** para garantir que o arquivo seja fechado após o uso, mesmo em caso de exceções;

2. Para escrita no arquivo *csv*, é utilizado o pacote ***csv*** e a função ***writerow()*** do seu objeto ***writer***;

3. Para a estrutura de *loop*, é utilizado uma declaração ***for*** por causa da estrutura sequencial das páginas de um *site* de busca de vagas de emprego.

```
import csv

def getVagas(filename):
    #Abre e fecha o arquivo
    with open(filename, 'a+', newline = '') as f:
        #Cria uma instancia do objeto writer
        writer = csv.writer(f, delimiter = ';')

        #Itera entre as paginas
        for i in count_pages:
            #Encontra proxima pagina
            pageurl = encontraPagina(i)
            #Faz requisicao do codigo da pagina
            page = requisicaoHTTP(pageurl)
            #Encontra os dados de cada vaga de emprego
            dados = filtro(page)
            #Escreve os dados no arquivo csv
            writer.writerow(dados)
```

Estrutura Geral do *Crawler*

O código acima representa o que está descrito nas explicações, mas com funções ainda genéricas para encontrar o endereço da próxima pagina ***encontraPagina()***, para efetuar a requisição HTTP ***requisicaoHTTP()*** e para filtrar as informações necessárias na página ***filtro()***. Todas elas são bem dependentes do *site* em que o *crawler* vai navegar, portanto são detalhas em subseções relativas a cada *site* de vaga de emprego considerado neste trabalho.

Apesar disso, nessa seção, ainda cabe discutir questões gerais consideradas na implementação da função ***requisicaoHTTP()***. Antes de pensar na requisição em si, é interessante esclarecer as possibilidades de páginas que se pode obter como resposta. Considerando a interatividade, pode-se dizer que existem basicamente dois tipos de páginas: estática, aquela cujo conteúdo é estático e não muda com a ação do usuário, e dinâmica, aquela cujo conteúdo é dinâmico e muda com a ação do usuário. Páginas estáticas são facilmente recuperáveis via requisição HTTP e, no Python, isso pode ser feito usando o pacote ***requests***. Entretanto, neste trabalho, todas as páginas consideradas são dinâmicas, e o problema disso é que para que elas exibam integralmente o conteúdo das vagas, é necessário que haja ação do usuário. Portanto, é necessário emular em código a ação de um usuário em um navegador para que se consiga recuperar as informações de vaga de emprego desses *sites*.

Para reproduzir essas ações de usuário, este trabalho utiliza o pacote ***Selenium*** se comunicando com o navegador *Google Chrome*. Por questões de eficiência e por usar um servidor que não possui ambiente gráfico instalado

(*Google Colab*), o navegador foi sempre executado em modo *headless*, o que significa que ele é executado sem abrir uma interface gráfica. O código abaixo apresenta de forma mais concreta o que foi discutido sobre a requisição HTTP, exceto a interação do usuário, que está representada pela função genérica ***interacaoUsuario()*** por ser dependente do *site* analisado pelo *crawler*.

```
from selenium import webdriver

def requisicaoHTTP(pageurl):
    #Muda as opcoes do Chrome para permitir modo headless
    chrome_options = webdriver.ChromeOptions()
    chrome_options.add_argument('--headless')
    chrome_options.add_argument('--no-sandbox')
    chrome_options.add_argument('--disable-dev-shm-usage')

    #Abre um navegador Chrome sem interface grafica
    driver = webdriver.Chrome('chromedriver',options=
    chrome_options)

    #Abre a pagina considerada
    driver.get(pageurl)

    #Para mostrar todo conteudo da pagina
    page = interacaoUsuario(driver)

    return(page)
```

Requisição HTTP

5.1.2 *Crawler* do Catho

O Catho tem uma estruturação para as páginas de busca que permite ao *crawler* navegar facilmente entre todos os resultados. Ele se utiliza de um atributo (***page***) passado no próprio endereço para indexar as páginas de resultado de busca. O código abaixo representa a função ***encontraPagina()*** para o Catho.

```
def encontraPagina(i):
    #Encontra a proxima pagina a ser visitada
    pageurl = 'https://www.catho.com.br/vagas/?page='+str(i)
    return(pageurl)
```

Formação do endereço da página de resultados no Catho

Prosseguindo com os detalhes do *crawler*, é importante entender porque a página para resultados de vagas de emprego do Catho é dinâmica. Isso é facilmente percebido na Figura 1.1, pois a descrição da vaga aparece inicialmente incompleta até que o usuário clique em um *link* ***continuar lendo***. Além disso, existe uma propaganda temporizada que aparece aleatoriamente e põe toda a página em plano de fundo. Ela desaparece quando o usuário clica em um botão de fechar depois de aguardar um determinado tempo. Essas duas interações precisaram ser incluídas na função ***interacaoUsuario()*** para o Catho, representada no código abaixo.

```
def interacaoUsuario(driver):
    #Encontra a referencia de todas as vagas
    ref_vagas =  driver.find_elements_by_class_name('boxVaga')
    #Loop para clicar em todas as vagas
    for vaga in ref_vagas:
        #Testa se a vaga esta visivel
        if vaga.is_displayed():
            #Clica na vaga visivel
            vaga.click()
        else:
            #Propaganda temporizada
            #Aguarda o tempo da propaganda
            time.sleep(8)
            #Alterna o foco para a propaganda
            f = driver.find_element_by_class_name('cboxIframe')
            driver.switch_to.frame(f)
            #Encontra e clica no botao de fechar
            btn = driver.find_element_by_class_name('botao-
   fechar')
            btn.click()
            #Retorna o foco para a pagina
            driver.switch_to_default_content()

            #Clica na vaga visivel
            vaga.click()

    #Carrega codigo fonte da pagina
    page = driver.page_source
    return(page)
```

Interação do Usuário no Catho

Após analisar o código-fonte da página de resultados do Catho, se percebe que as vagas são representadas em elementos da classe "boxVaga", que a propaganda aparece em um elemento da classe "cboxIframe" e que o botão

para fechar a propaganda é um elemento da classe “botao-fechar”. Essas classes são utilizadas no código da função ***interacaoUsuario()*** para encontrar as referências dos elementos que necessitam de interação automatizada do usuário.

Por fim, resta entender os atributos da vaga de emprego do Catho para saber como filtrar as informações necessárias para a construção do conjunto de dados. Na Figura 5.2, é representado um exemplo de vaga do Catho e alguns de seus atributos devidamente classificados. Além desses atributos, a vaga opcionalmente pode ter Idioma, Regime de Contratação e Informações Adicionais. Todos eles são de fácil recuperação, exceto Responsabilidades e Requisitos que pertencem a um mesmo atributo chamado Descrição. Para resolver esse problema, foi observada uma regra que permite separar os dois atributos na maioria dos casos: a primeira quebra de linha no atributo Descrição corresponde a separação entre Responsabilidades e Requisitos. Por simplicidade, essa regra é aplicada para determinar se a vaga é bem ou mal formada e, portanto, se vai ser ou não coletada. No Capítulo 6 é apresentado o quantitativo de cada uma dessas vagas.

Figura 5.2: Alguns atributos de vaga de emprego do Catho

Para filtrar o código-fonte da página e recuperar os atributos apontados acima, é necessário um analisador de HTML e este trabalho utiliza o ***BeatifulSoup*** para essa tarefa. Decidida a tecnologia a ser utilizada, fez-se um mapeamento entre cada atributo e a classe do seu respectivo elemento HTML para extrair as informações e construir o conjunto de dados, lembrando que vagas sem quebra de linha no elemento Descrição são consideradas mal formadas e não são coletadas.

O código abaixo representa a função ***filtro()*** para os dados de vaga de emprego vindos do Catho. No fim, é utilizada uma função genérica ***trataDados()*** que basicamente checa a regra do atributo Descrição apresentada anteriormente e determina se a vaga vai ser ou não coletada.

```
from bs4 import BeatifulSoup

def filtro(page):
    #Usa o analisador no codigo-fonte da pagina
    soup = BeautifulSoup(driver.page_source, 'html.parser')
    #Recupera todas as vagas da pagina
    ref_vagas = soup.find_all(class_ = 'boxVaga')
    #Lista com os dados de todas as vagas
    dados = []
    #Loop para percorrer todas as vagas
    for vaga in ref_vagas:
        #Recupera o titulo da vaga
        el = vaga.find(class_ = 'viewVagaAction')
        tit = el.get_text()
        #Recupera o salario
        el = vaga.find(class_ = 'salarioLocal')
        sal = el.get_text()
        #Recupera a descricao da vaga
        el = vaga.find(class_ = 'descriptionIncloplete')
        desc = el.get_text()
        #Recupera os beneficios da vaga
        el = vaga.find(class_ = 'beneficios')
        ben = el.get_text() if el else ''
        #Recupera o regime de contratacao
        el = vaga.find(class_ = 'regimeContratacao')
        reg = el.get_text() if el else ''
        #Recupera informacoes adicionais
        el = vaga.find(class_ = 'informacoesAdicionais')
        info = el.get_text() if el else ''
        #Recupera horario de trabalho
        el = vaga.find(class_ = 'horario')
        hor = el.get_text() if el else ''
        #Recupera local de trabalho
        el = vaga.find(class_ = 'cidades')
        loc = el.get_text() if el else ''
        #Recupera idioma requisitado
        el = vaga.find(class_ = 'idioma')
        idio = el.get_text() if el else ''
        dados.append([tit, sal, desc, ben, reg, info, hor, loc,
    idio])
    return(trataDados(dados))
```

Extração dos dados da vaga de emprego do Catho

5.1.3 *Crawler* do VAGAS.com.br

O VAGAS.com.br tem uma estruturação diferenciada para as páginas de busca, pois em vez de navegar entre várias páginas com resultados diferentes, ele traz uma página única de resultados contendo um botão que, ao ser acionado, mostra mais vagas. Portanto, a navegação para mostrar mais resultados vai para a ***interacaoUsuario()*** e a função ***encontraPagina()*** é simplificada para o VAGAS.com.br conforme mostrado no código abaixo.

```
def encontraPagina(i):
    #Encontra a proxima pagina a ser visitada
    pageurl = 'https://www.vagas.com.br/vagas-de-'
    return(pageurl)
```

Formação do endereço da página de resultados no VAGAS.com.br

Prosseguindo com os detalhes do *crawler*, já está claro um dos motivos da página ser dinâmica: a necessidade de interação com usuário para mostrar mais as páginas de resultados da busca por vagas. Além disso, assim como no Catho, as vagas são mostradas inicialmente com descrição incompleta, só que para obter a descrição completa, é preciso navegar para a página de cada uma das vagas. Essas duas interações precisaram ser incluídas na função ***interacaoUsuario()*** para o VAGAS.com.br, representada no código abaixo.

```
def interacaoUsuario(driver):
    #Encontra a referencia de todas as vagas (antes)
    ref_vagas =  driver.find_elements_by_class_name('link-
   detalhes-vaga')
    #Encontra total de vagas
    n_antes = len(ref_vagas)
    #Encontra a referencia do botao Mais Vagas
    btn = driver.find_element_by_class_name('btMaisVagas')
    #Clica no botao Mais Vagas
    btn.click()
    #Encontra a referencia de todas as vagas (depois)
    ref_vagas =  driver.find_elements_by_class_name('link-
   detalhes-vag')
    #Encontra total de vagas
    n_depois = len(ref_vagas)

    #Abre outra instancia do navegador sem interface grafica
    driver2 = webdriver.Chrome('chromedriver',options=
   chrome_options)

```

```
    page = []
    #Loop para clicar em todas as vagas
    for vaga in ref_vagas[n_antes+1:ndepois]:
        #Abre a pagina da vaga
        driver2.get(vaga.get_property('href'))
        #Carrega codigo fonte da pagina
        page.append(driver2.page_source)

    #Retorna uma lista com os codigos-fontes
    #das paginas de vagas de emprego
    return(page)
```

Interação do Usuário no VAGAS.com.br

Após analisar o código-fonte da página de resultados do VAGAS.com.br, se percebe que as vagas são representadas em elementos da classe "link-detalhes-vaga" e que o botão para mostrar mais vagas é um elemento da classe "btMaisVagas". Além das classes diferentes, outra particularidade na função ***interacaoUsuario()*** para o *site* VAGAS.com.br é que ela retorna uma lista com os códigos-fontes das páginas de todas as vagas da iteração, em vez do código-fonte de uma única página com todas as vagas.

Por fim, resta entender os atributos da vaga de emprego do VAGAS.com.br para extrair informações e complementar o conjunto de dados. Diferentemente das vagas do Catho, essas são mais simples e contém apenas dois atributos: Título e Descrição. Dessa vez, não foi observada nenhuma regra para a divisão da descrição em classes e, portanto, essa tarefa foi feita de forma manual. O código abaixo representa a função ***filtro()*** para vagas de emprego vindas do VAGAS.com.br.

```
from bs4 import BeatifulSoup
def filtro(page):
    #Lista com os dados de todas as vagas
    dados = []
    #Loop para percorrer todas as vagas
    for pg in page:
        #Usa o analisador no codigo-fonte da pagina
        vaga = BeautifulSoup(pg, 'html.parser')
        #Recupera o titulo da vaga
        tit = vaga.get_property('title')
        #Recupera a descricao da vaga
        el = vaga.find(class_ = 'texto')
        desc = el.get_text()
        dados.append([tit,desc])
    return(dados)
```

Extração dos dados da vaga de emprego do VAGAS.com.br

5.1.4 *Crawler* do LinkedIn

O LinkedIn tem uma estruturação para as páginas de busca, semelhante a do Catho, que permite ao *crawler* navegar facilmente entre todos os resultados. Ele se utiliza de dois atributos (***pageNum*** e ***start***) passados no próprio endereço para indexar as páginas de resultado de busca. O código abaixo representa a função ***encontraPagina()*** para o LinkedIn.

```
def encontraPagina(i):
    #Encontra a proxima pagina a ser visitada
    pageurl = 'https://br.linkedin.com/jobs/linkedin-vagas?
   position=1&count=25&pageNum='+str(i)+'&start='+str(25*i)
    return(pageurl)
```

Formação do endereço da página de resultados no LinkedIn

Prosseguindo com os detalhes do *crawler*, é importante entender porque a página para resultados de vagas de emprego do LinkedIn é dinâmica. Assim como o VAGAS.com.br, as vagas são mostradas inicialmente com descrição incompleta, só que para obter a descrição completa, é preciso clicar em cada uma das vagas para abrir um elemento dinâmico que mostra a descrição completa. Essa interação precisa ser incluída na função ***interacaoUsuario()*** para o LinkedIn, representada no código abaixo.

```
def interacaoUsuario(driver):
    #Encontra a referencia de todas as vagas
    ref_vagas =  driver.find_elements_by_class_name('listed-job
   -posting')
    #Abre outra instancia do navegador sem interface grafica
    driver2 = webdriver.Chrome('chromedriver',options=
   chrome_options)

    page = []
    #Loop para clicar em todas as vagas
    for vaga in ref_vagas:
        #Abre a pagina da vaga
        driver2.get(vaga.get_property('href'))
        #Carrega codigo fonte da pagina
        page.append(driver2.page_source)

    #Retorna uma lista com os codigos-fontes
    #das paginas de vagas de emprego
    return(page)
```

Interação do Usuário no LinkedIn

Após analisar o código-fonte da página de resultados do LinkedIn, se percebe que as vagas são representadas em elementos da classe se percebe que as vagas são representadas em elementos da classe "listed-job-posting". Além disso, a função ***interacaoUsuario()*** para o *site* LinkedIn tem comportamento similar a do VAGAS.com.br, ou seja, ela retorna uma lista com os códigos-fontes das páginas de todas as vagas da iteração, em vez do código-fonte de uma única página com todas as vagas.

Por fim, resta entender os atributos da vaga de emprego do LinkedIn para extrair informações e complementar o conjunto de dados. Nesse caso, assim como nas vagas do *site* VAGAS.com.br, elas são mais simples e contém apenas dois atributos: Título e Descrição. Também não foi observada nenhuma regra para a divisão da descrição em classes e, portanto, essa tarefa foi feita de forma manual. O código abaixo representa a função ***filtro()*** para vagas de emprego vindas do LinkedIn.

```
from bs4 import BeatifulSoup
def filtro(page):
    #Lista com os dados de todas as vagas
    dados = []
    #Loop para percorrer todas as vagas
    for pg in page:
        #Usa o analisador no codigo-fonte da pagina
        vaga = BeautifulSoup(pg, 'html.parser')
        #Recupera o titulo da vaga
        el = vaga.find(class_ = 'topcard__title')
        tit = el.get_text()
        #Recupera a descricao da vaga
        el = vaga.find(class_ = 'description__text
   description__text--rich')
        desc = el.get_text()
        dados.append([tit,desc])
    return(dados)
```

Extração dos dados da vaga de emprego do LinkedIn

5.2 Tratamento dos Dados

Coletados os dados de vaga de emprego dos três *sites* citados, é preciso tratar esses dados e transformá-los em conjuntos de dados de sentenças de vagas de emprego. Para dividir os textos das vagas de emprego em sentenças existe uma variedade de pacotes/funções.

Este trabalho analisou os pacotes ***NLTK***, ***TextBlob*** e ***Spacy*** para a separação do texto em sentenças. Os dois primeiros são idênticos, pois o pacote

TextBlob usa implicitamente a função ***sent_tokenize()*** do pacote ***NLTK***. Analisando o seu comportamento, é possível perceber que a função é adequada para os textos de vaga de emprego, errando apenas em casos de abreviação pontuada, como " Trabalhar em Sto. Antonio" que é separado em duas sentenças. Por outro lado, a função oferecida pelo ***Spacy*** traz um problema mais difícil de conviver, pois ela utiliza, além da pontuação, a presença de letras maiúsculas para separar sentenças. Assim, por exemplo, a sentença "Conhecimentos em desenvolvimento JAVA, principalmente especificação padrão JEE" é dividida em três sentenças. Portanto, foi utilizado o pacote ***NLTK*** conforme ilustrado no código abaixo, no qual além de dividir o texto em sentenças, trata as mesmas para remover duplicatas. Isso evita que haja sentenças idênticas no conjunto de treino e de teste, eliminando essa fonte de viés para o estimador.

```
from nltk.tokenize import sent_tokenize
import pandas as pd

#Le o arquivo com as vagas
df_vagas = pd.read_csv('vagas.csv')
df_sents = []

#Percorre todas as vagas
for vaga in df_vagas.text:
    for sent in sent_tokenize(vaga):
        df_sents.append(sent)

#Remove as sentencas duplicadas
df_sents.drop_duplicates(subset='text',inplace=True)

#Salva o dataset
df_sents.to_csv('sents.csv')
```

Tokenização de Sentenças e Deduplicação

De posse desse conjunto de dados de sentenças, resta estabelecer a classe de cada uma delas e essa tarefa foi feita de diversas formas, algumas vezes automaticamente, outras manualmente. Por isso, não é discutida a implementação dessa parte do código. Entretanto, cabe citar que os dados do Catho foram utilizados da seguinte forma: atributo Descrição antes da primeira quebra de linha - classe Responsabilidades, atributo Descrição após a primeira quebra de linha - classe Requisitos, atributo Benefícios - classe Benefícios, Informações Adicionais - classe Outros, Idiomas - classe Requisitos. O restante dos atributos foi utilizado em experimentos mas não se mostrou relevante para as análises por se apresentarem de forma menos textual, como o atributo Local

que normalmente contém o nome de uma cidade ou o atributo Regime de Contratação que normalmente contém um entre dois valores: CLT e Autônomo.

5.3 Pré-processamento e Construção de Atributos

Este trabalho, por simplicidade, considera essas duas etapas juntas. Isso se deve ao fato de as implementações utilizadas permitirem essa união de operações de forma ótima. Conforme apresentado no Capítulo 4, dois tipos de atributos foram utilizados: léxicos e semânticos. Ambos tem suas implementações descritas nas subseções a seguir.

5.3.1 Remoção de *Stop Words* e Stemização

Mantendo o padrão iniciado na segmentação de sentenças, este trabalho utiliza as *stop words* do pacote***NLTK***. Além disso, conforme discutido no Capítulo 4, a melhor heurística de stemização em língua portuguesa é a RSLP e ela também está implementada no pacote ***NLTK***. O código abaixo especifica uma função de análise, composta pela aplicação da heurística RSLP, e uma lista de *stop words* para serem utilizadas pelas funções que controem atributos.

```
from nltk.corpus import stopwords
from nltk.stem import RSLPStemmer
from sklearn.feature_extraction.text import CountVectorizer


stopw = stopwords.words('portuguese')
stemmer = RSLPStemmer()
analyzer = CountVectorizer().build_analyzer()

def stemizacao:
    return (stemmer.stem(w) for w in analyzer(doc))
```

Stemização e Lista de *Stop Words*

5.3.2 Atributos Léxicos

As técnicas consideradas para construção de atributos léxicos foram apresentadas no Capítulo 4 e foram implementadas utilizando o pacote ***Scikit-Learn*** (172), o principal pacote de Python para aprendizado de máquina. No código a seguir estão implementadas as representações: binária, TF e TF.IDF. Foram considerados dois cenários de pré-processamento que estão

representados abaixo por A e B, que consistem em: A) sem remoção de *stop words* e sem stemização e B) com remoção de *stop words* e com stemização, ambos com a tokenização padrão provida pelas classes ***CountVectorizer()*** e ***TfidfVectorizer()***.

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import Binarizer
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd

#Le conjunto de dados de sentencas
df_sents = pd.read_csv('df_sents.csv')

#Representacao TF - Cenario A
tf = CountVectorizer()
rep_tfA = tf.fit_transform(df_sents.text)

#Representacao Binaria - Cenario A
bin = Binarizer()
rep_binA = bin.fit_transform(rep_tfA)

#Representacao TFIDF - Cenario A
tfidf = TfidfVectorizer()
rep_tfidfA = tfidf.fit_transform(df_sents.text)

#Representacao TF - Cenario B
tf = CountVectorizer(analyzer = stemizacao, stop_words = stopw)
rep_tfB = tf.fit_transform(df_sents.text)

#Representacao Binaria - Cenario B
bin = Binarizer()
rep_binB = bin.fit_transform(rep_tf)

#Representacao TFIDF - Cenario B
tfidf = TfidfVectorizer(analyzer = stemizacao, stop_words =
    stopw)
rep_tfidfB = tfidf.fit_transform(df_sents.text)
```

Construção de Atributos Léxicos

5.3.3 Atributos Semânticos

As técnicas de *Word Embedding* consideradas para a construção de atributos semânticos foram apresentadas no Capítulo 4 e foram implementadas utilizando o pacote ***Gensim***. No código a seguir estão implementadas as

técnicas *Word2Vec* e *FastText*. A tokenização, para esse caso, teve que ser explicitamente executada e, para isso, utilizou-se a função ***word_tokenize()*** do pacote ***NLTK***.

```
from gensim.models import Word2Vec
from gensim.models import FastText
from nltk import word_tokenize
import pandas as pd


#Le o conjunto de dados
ds_sents = pd.read('ds_sents.csv')
ds_sents['token'] = ds_sents.text.apply(word_tokenize)

#Cria e treina 100 epocas de Word2Vec
#Skip-gram com Amostragem Negativa
#Vetores de tamanho 100
word2vec = Word2Vec(ds_sents.token, sg=1, size=100, iter=100)

#Analisa os resultados
analisaResultados(word2vec)

#Treina mais 10 epocas o modelo Word2Vec
word2vec.train(ds_sents.token, total_examples = len(ds_sents.
    token), epochs = 10)

#Analisa os resultados
analisaResultados(word2vec)

#Cria e treina 100 epocas de FastText
#Skip-gram com Amostragem Negativa
#Vetores de tamanho 100
fasttext = FastText(ds_sents.token, sg=1, size=100, iter=100)

#Analisa os resultados
analisaResultados(fasttext)

#Treina mais 10 epocas o modelo FastText
fasttext.build_vocab(ds_sents.token, update = True)
fasttext.train(ds_sents.token, total_examples = len(ds_sents.
    token), epochs = 10)

#Analisa os resultados
analisaResultados(fasttext)
```

Treinamento de Modelos de *Word Embedding*

A função genérica ***analisaResultados()*** representa uma análise quali-

tativa do treinamento do modelo de acordo com as propriedades de seus vetores, como verificar a consistência das palavras semelhantes. Alguns exemplos disso são dados no Capítulo 6.

Depois de treinados os modelos, as sentenças do conjunto de dados são representadas através da média dos vetores de suas sentenças e essa é a entrada para os algoritmos de aprendizado (92).

5.4 Treinamento de Algoritmos de Aprendizado e Métricas

Essa subseção apresenta a implementação dos algoritmos clássicos de aprendizado considerados neste trabalho: Naïve Bayes, Regressão Logística, Floresta Aleatória e Máquina de Suporte de Vetores (SVM). Mais uma vez, foi utilizado o pacote ***Scikit-Learn*** (172) com o intuito de facilitar essa tarefa. Para treinar, validar e testar os algoritmos em condições diferentes, o conjunto de dados foi dividido aleatoriamente em três partes, treinamento, validação e teste, na seguinte proporção: 60% - 20% - 20%. Isso é importante, pois permite escolher a melhor combinação algoritmo-condição sem viesar a estimativa da métrica em dados fora do conjunto de treinamento, ou seja, sem comprometer a capacidade de generalização do algoritmo.

Considerando ***df_trn*** como o conjunto de dados de treinamento e ***df_val*** como o conjunto de dados de validação, ambos com os atributos *text*, que contém as sentenças, e *labels*, que contém as classes das sentenças, tem-se no código abaixo uma ilustração da implementação do treinamento e da avaliação da acurácia (***acc***) desses algoritmos. A função genérica ***encontraAtributos()*** representa qualquer uma das técnicas de construção de atributos discutida anteriormente neste trabalho.

```
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC

#Encontra os atributos para o conj de treino
Xtrn = EncontraAtributos(df_trn.text)
ytrn = df_trn.labels
#Encontra os atributos para o conjunto de validacao
Xval = EncontraAtributos(df_val.text)
yval = df_val.labels

#Naive Bayes
```

```
alg = MultinomialNB()
alg.fit(Xtrn)

#Regressao Logistica
alg = LogisticRegression()
alg.fit(Xtrn)

#Floresta Aleatoria
alg = RandomForestClassifier()
alg.fit(Xtrn)

#Maquina de Vetores de Suporte (SVM)
alg = LinearSVC()
alg.fit(Xtrn)

#Avaliacao de acuracia para qualquer um dos algoritmos acima
acc_trn = accuracy_score(ytrn, alg.predict(Xtrn))]
acc_val = accuracy_score(ytrn, alg.predict(Xval))]
```

Treinamento e Avaliação de Algoritmos Clássicos de Aprendizado

5.5 Segmentação de Texto

Para analisar a qualidade da segmentação de texto realizada neste trabalho, foi utilizado o pacote ***NLTK*** e suas funções ***pk()*** e ***windowdiff()*** para calcular as métricas já discutidas no Capítulo 4.

6
Resultados

Esse capítulo apresenta características do conjunto de dados utilizados e os resultados dos experimentos em busca do melhor classificador de sentenças de vagas de emprego considerando as quatro classes já apresentadas: Responsabilidades, Requisitos, Benefícios e Outros. Com uma abordagem empírica, este trabalho entende o melhor classificador como a combinação de pré-processamento, construção de atributos e algoritmo de aprendizado que produz o melhor resultado em acurácia de validação levando em conta o sobreajuste/subajuste do modelo.

O classificador de sentenças é um método proposto por este trabalho para tratar de forma prática a segmentação semântica de vagas de emprego. Por isso, após determinado o melhor classificador, este trabalho experimenta ele na tarefa de segmentação semântica das vagas de emprego.

6.1
Características do Conjunto de Dados

Este trabalho utiliza dois tipos de conjunto de dados com finalidades diferentes: um com gabarito gerado automaticamente e outro com gabarito gerado manualmente. O primeiro deles, chamado de *vagas-auto*, foi coletado inteiramente do Catho e foi utilizado para treino, validação e teste durante o processo de busca do melhor classificador. Enquanto que o segundo deles, chamado de *vagas-man*, foi coletado do Catho, do VAGAS.com.br e do LinkedIn e foi utilizado para testes com a finalidade de mostrar uma evidência de generalização do classificador e para tentar entender um pouco do efeito da atribuição automática de classes feita no *vagas-auto* na classificação.

Tabela 6.1: Descrição dos Dados no Conjunto *data-auto*

Vagas			**Sentenças**				
83714			558220				
Trn	**Val**	**Tst**	**Únicas**				**Duplicadas**
			361277				196943
			C1	**C2**	**C3**	**C0**	
50237	16740	16737	197792	126129	7415	29941	

Tabela 6.2: Descrição dos Dados no Conjunto *data-man*

Vagas			**Sentenças**				
1884			35243				
			Únicas				**Duplicadas**
C	**V**	**L**	10978				
			C1	**C2**	**C3**	**C0**	24265
429	680	775	5117	4268	444	1149	

As informações gerais do conjunto de dados *vagas-auto* estão representadas na Tabela 6.1. Nela, é possível perceber que o conjunto é composto por 83714 vagas de emprego que foram divididas na proporção 60/20/20 em treino, validação e teste, respectivamente. Essas vagas, por sua vez, conforme apresentado no Capítulo 5, foram decompostas em 558220 sentenças, das quais 196943 eram duplicadas e tiveram que ser descartadas para não introduzir dependência entre os conjuntos de treino, validação e teste. Entre as sentenças únicas, 55% delas pertencem a classe C1 (Responsabilidades), 35% a classe C2 (Requisitos), 2% a classe C3 (Benefícios) e 8% a classe C0 (Outros). É possível perceber que as classes são bem desbalanceadas entre si e isso é importante para avaliação dos algoritmos, conforme visto no Capítulo 4 quando se discutiu as métricas.

Por outro lado, as informações gerais do conjunto de dados *vagas-man* estão representadas na Tabela 6.2. Nela, é possível perceber que o conjunto é composto por 1884 vagas de empregos, das quais 429 são do Catho (C), 680 são do VAGAS.com.br (V) e 775 são do LinkedIn (L). A quantidade de vagas, nesse caso, foi restrita, por causa da necessidade de atribuir o gabarito manualmente. Prosseguindo com a análise, as vagas foram decompostas em 35243 sentenças, das quais 19978 eram duplicadas e tiveram que ser descartadas para não viesar a acurácia avaliada. Entre as sentenças únicas, 47% delas pertencem a classe C1 (Responsabilidades), 39% a classe C2 (Requisitos), 4% a classe C3 (Benefícios) e 10% a classe C0 (Outros). Mais uma vez, é possível perceber que as classes são bem desbalanceadas entre si.

Discutidas as informações gerais dos dois conjuntos de dados, resta caracterizar um pouco melhor a vaga de emprego buscando entender o tamanho do seu texto em quantidade de sentenças, o tamanho de suas sentenças em quantidade de palavras e a sua riqueza léxica medida através da diversidade léxica, que é a razão entre palavras únicas sobre o total de palavras. Essas análises estatísticas foram feitas com os conjuntos de dados completos, antes de serem deduplicados, com a finalidade de caracterizar a vaga antes do processamento em seu estado natural. Na Tabela 6.3, se encontram representadas as medidas citadas acima, separadas por *site* de vaga de emprego.

Tabela 6.3: Análise Estatística do Texto de Vagas de Emprego

	Catho	**VAGAS.com.br**	**LinkedIn**
Sentenças/vaga	6.7 ± 4.0	10.1 ± 6.8	31.9 ± 13.2
Palavras/sentença	10.6 ± 10.3	12.2 ± 15.4	10.1 ± 11.7
Palavras Únicas	50928	6267	11025
Total de Palavras	5958668	83811	250683
Diversidade Léxica	0.0085	0.0747	0.0439

Analisando a Tabela 6.3 é possível perceber que a quantidade de sentenças varia bastante entre os *sites*, de 6.7 a 31.9 sentenças em média. Entretanto, cada sentença tem por volta de 10 palavras em média, o que em conjunto com a quantidade de sentenças leva a conclusão de que os textos de vagas de emprego são curtos, tendo tamanho equivalente a um parágrafo de um texto longo. O grande desvio padrão na quantidade de palavras por sentença, que é uma medida estritamente positiva, indica a presença de uma assimetria positiva na distribuição de probabilidade dos dados. Essa assimetria positiva é uma evidência de que vagas de emprego são apenas eventualmente mais longas que um parágrafo. Por último, mas não menos importante, pode ser percebido que o texto de vaga de emprego tem baixa diversidade léxica o que leva a concluir que o texto da vaga de emprego é simples e repetitivo. A variação entre a diversidade léxica para os diferentes *sites* analisados é causada pelas diferentes quantidades de vagas em cada um, que variam de milhares a dezenas de milhares dependendo do *site*.

6.2 Classificação de Texto

Nessa seção são apresentados os experimentos que buscam a melhor combinação entre pré-processamento, construção de atributos e algoritmo de aprendizado. Neles são consideradas todas as técnicas de construção de atributos, tanto léxicos quanto semânticos, implementadas no Capítulo 5. São considerados também todos os algoritmos de aprendizado implementados no Capítulo 5: Naïve Bayes, Regressão Logística Multinomial, Máquinas de Suporte de Vetores e Florestas Aleatórias.

6.2.1 Atributos Léxicos

Este trabalho utiliza as seguintes técnicas de construção de atributos léxicos: representação binária, TF e TF.IDF. Em associação com elas, são utilizados os dois cenários de pré-processamento dos dados textuais já citados no capítulo anterior: A) sem remoção de *stop words* e sem stemização, B) com

Tabela 6.4: Acurácia em Treino e Validação dos Algoritmo de Aprendizado com Diversas Técnicas de Construção de Atributos Léxicos (Cenário A)

	Binário		**TF**		**TF.IDF**	
	Trn	**Val**	**Trn**	**Val**	**Trn**	**Val**
Naïve Bayes	94.09%	93.34%	94.06%	93.27%	92.71%	91.69%
Regressão Logística	96.61%	95.43%	95.71%	95.14%	96.11%	95.35%
Floresta Aleatória	99.60%	94.55%	99.50%	94.07%	99.50%	94.45%
SVM	97.36%	95.01%	97.31%	94.87%	97.11%	95.33%

Tabela 6.5: Acurácia em Treino e Validação dos Algoritmo de Aprendizado com Diversas Técnicas de Construção de Atributos Léxicos (Cenário B)

	Binário		**TF**		**TF.IDF**	
	Trn	**Val**	**Trn**	**Val**	**Trn**	**Val**
Naïve Bayes	93.34%	92.69%	93.28%	92.63%	91.80%	90.86%
Regressão Logística	95.88%	94.96%	95.19%	94.77%	95.62%	94.96%
Floresta Aleatória	99.50%	94.14%	99.51%	94.14%	99.49%	93.99%
SVM	96.24%	94.56%	96.19%	94.49%	96.21%	94.85%

remoção de *stop words* e com stemização. Além disso, todos os algoritmos de aprendizado foram utilizados com seus hiperparâmetros padrões.

Os resultados desses experimentos para o cenário A estão na Tabela 6.4 e para o cenário B na Tabela 6.5. Analisando esses resultados, alguns pontos chamam a atenção. O primeiro deles é que a acurácia dos algoritmos está na mesma faixa do estado da arte para a tarefa, entre 93.43%-99.38%, conforme pode ser visto na Tabela 3.2. Isso surpreende porque somente foram usados algoritmos clássicos sem otimização de hiperparâmetros. Tomando esse resultado como base, foi percebido que investir em algoritmos mais complexos, como Redes Neurais (tradicionais, convolutivas ou recorrentes), era desnecessário e, portanto, este trabalho se manteve no clássico.

Um segundo ponto de destaque é que o cenário B foi pior que o cenário A em todos os casos, alinhado com a tendência de que pré-processamento excessivo dos dados iniciais pode levar a perda de informação. No caso do conjunto de dados deste trabalho, isso é ainda mais crítico, uma vez que como sentenças são curtas, contendo entre duas e vinte palavras, a remoção de sufixos e *stop words* pode simplificar demais a sentença.

Por fim, resta apresentar uma breve análise sobre os resultados de cada um dos algoritmos e entender qual o melhor deles para a tarefa de classificação no conjunto de dados *data-auto*.

O Naïve Bayes apresenta um bom resultado para a tarefa considerada (por volta de 93% de acurácia no conjunto de validação), entretanto, quando comparado com os outros ele é o pior. Apesar disso, sempre vale a pena analisar

sua performance, porque ele é o algoritmo mais simples e mais rápido de treinar entre esses quatro considerados.

A Floresta Aleatória apresenta um resultado excelente no conjunto de treinamento (acurácia de 99.5%) mas medíocre no conjunto de validação (por volta de 94%) quando comparado com os outros. Isso é um indício de sobre-ajuste do modelo, o que é comum em modelos do tipo *bagging* como a Floresta Aleatória. Nessas circunstâncias, caberia uma otimização de hiperparâmetros do modelo para tentar regulariza-lo, evitando o sobre-ajuste, todavia, dado que os outros modelos foram melhores, foi decidido não investir nessa tarefa.

Resta analisar os resultados da Regressão Logística e da Máquina de Vetores de Suporte (SVM). Ambos os modelos tiveram bons resultados em treino e validação, entretanto a Regressão Logística com representação binária se sobressai com uma acurácia no conjunto de validação de 95.43% contra 95.33% do modelo SVM com TF.IDF. Portanto, Regressão Logística com representação binária no cenário A é a melhor combinação considerando apenas atributos léxicos.

6.2.2 Dependência e Probabilidade Condicional

Durante a subseção anterior, os classificadores foram avaliados considerando cada sentença como independente. Na prática, isso não é observado, uma vez que as sentenças seguem determinados padrões dentro de uma vaga. Por exemplo, é possível perceber que muitas das vagas do conjunto de dados começam com sentenças classificadas como Responsabilidades e que há alguma continuidade entre as classes, não havendo frequentemente sentenças de classes iguais intercaladas por sentenças de classes diferentes.

Sem querer buscar um conjunto de regras para a especificação dessa dependência, modelou-se a vaga de emprego como um processo estocástico, no qual a classe da sentença atual depende probabilisticamente da classe da sentença anterior, conforme ilustrado na Figura 6.2. Em cada arco entre classes está a probabilidade p_{ij} que é a probabilidade da sentença atual ser da classe j dado que a sentença anterior foi atribuída a classe i.

Analisando o conjunto de treino com todas as sentenças, chega-se ao conjunto de probabilidades apresentado na Tabela 6.6. Nela é possível perceber claramente alguns padrões:

- Todas as vagas do conjunto de treinamento começam com sentenças classificadas como Responsabilidades;
- O bloco de sentenças classificadas como Responsabilidade é contínuo, pois nenhuma classe consegue retornar a essa classe;

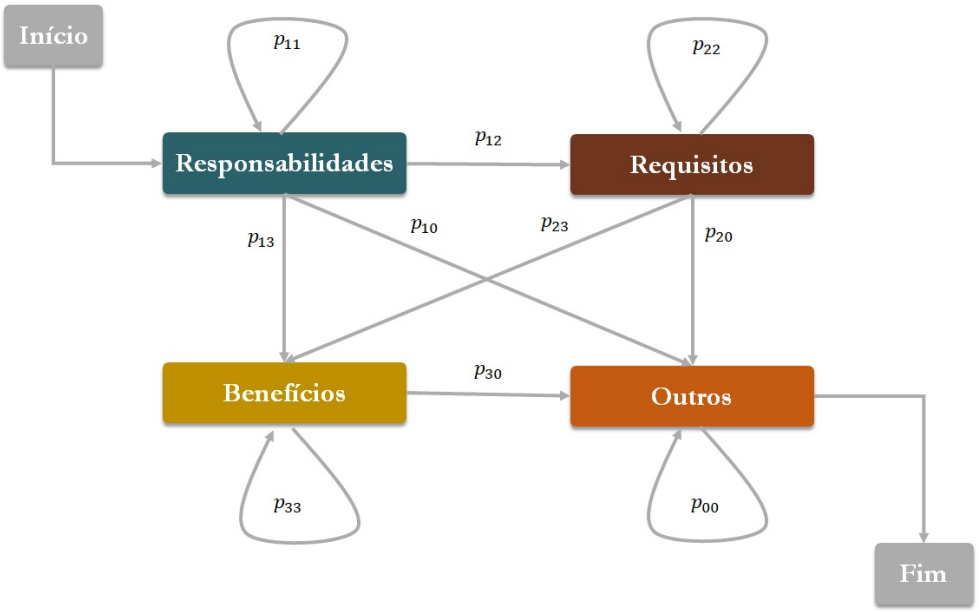

Figura 6.1: Representação da Vaga de Emprego como um Processo Estocástico

Tabela 6.6: Probabilidades Condicionais entre Classes

	Resp.	**Requisitos**	**Benefícios**	**Outros**
Resp.	63.31%	36.69%	0	0
Requisitos	0	64.17%	29.46%	6.37%
Benefícios	0	15.90%	1.67%	82.43%
Outros	0	0	0	1

– A sentença atual tem uma probabilidade por volta de 64% de manter a classe da sentença anterior, caso ela seja classificada como Responsabilidades e Requisitos.

Como o foco deste trabalho não é enumerar extensivamente essas regras, resolveu-se utilizar essa matriz de probabilidades para complementar o vetor de probabilidades de classificação resultado dos classificadores Naïve Bayes e Regressão Logística. Operacionalmente, a classe passou a ser determinada como a classe de máxima probabilidade no vetor de probabilidades calculado como a média entre o vetor de resultado do classificador e o vetor de probabilidades condicionais. Com isso, esperava-se que se introduzisse uma inércia para mudança de classe, com a expectativa de melhorar a classificação tentando reduzir a quantidade de blocos alternados da mesma classe.

Os resultados de acurácia para os cenários A e B já considerados, variando métodos de construção de atributos e algoritmos de aprendizado são apresentados nas Tabelas 6.7 e 6.8. Analisando os resultados, é possível perceber que, em alguns casos, como Naïve Bayes no Cenário A com atributos binários, houve uma melhoria marginal na acurácia como esperado. Entretanto,

Tabela 6.7: Acurácia em Treino e Validação dos Algoritmo de Aprendizado com Diversas Técnicas de Construção de Atributos Léxicos Acrescidos das Probabilidades Condicionais das Transições entre Classes (Cenário A)

	Binário		**TF**		**TF.IDF**	
	Trn	**Val**	**Trn**	**Val**	**Trn**	**Val**
Naïve Bayes	94.84%	94.50%	94.81%	94.45%	92.47%	91.80%
Regressão Logística	95.13%	94.21%	94.33%	93.85%	94.97%	94.41%

Tabela 6.8: Acurácia em Treino e Validação dos Algoritmo de Aprendizado com Diversas Técnicas de Construção de Atributos Léxicos Acrescidos das Probabilidades Condicionais das Transições entre Classes (Cenário B)

	Binário		**TF**		**TF.IDF**	
	Trn	**Val**	**Trn**	**Val**	**Trn**	**Val**
Naïve Bayes	94.13%	93.84%	94.10%	93.79%	90.31%	89.92%
Regressão Logística	93.92%	93.15%	93.74%	93.35%	94.48%	94.01%

nenhuma dessas melhorias foi suficiente para superar o melhor resultado obtido usando somente os atributos léxicos (95.43%).

6.2.3 Atributos Semânticos

Para a construção de atributos semânticos, foram utilizadas duas técnicas de *Word Embedding*: *Word2Vec* e *FastText*. Para a técnica *Word2Vec*, foram treinados os modelos CBOW e SGNS com seus hiperparâmetros padrões, enquanto que para a técnica *FastText* foi treinado apenas o modelo CBOW. Para cada uma dessas configurações de experimento, foram avaliados variações entre dois tamanhos de representação vetorial (100 e 300), conforme apresentado nas Tabelas 6.9 e 6.10. Diferentemente do procedimento adotado na subseção anterior, somente foi considerado um cenário de pré-processamento, sem remoção de *stop words* e sem stemização.

São considerados apenas três dos quatro algoritmos clássicos implementados: Regressão Logística, Máquina de Vetores de Suporte e Floresta Aleatória. O algoritmo Naïve Bayes foi excluído por impossibilidade de treinamento, uma vez que ele exige que o vetor de atributos seja positivo e essas representações semânticas contém valores positivos e negativos nos vetores.

Os modelos foram treinados por no máximo 120 épocas e sua qualidade ao longo do treinamento era avaliada através da consistência de palavras semelhantes e da coerência de algumas operações algébricas com os vetores.

Quanto a consistência de palavras semelhantes, eram verificadas as palavras mais semelhantes a *mestrado*, por exemplo. Nos modelos *Word2Vec*, foi considerado como um bom resultado quando a representação retornava

Tabela 6.9: Acurácia em Treino e Validação dos Algoritmo de Aprendizado com Diversas Técnicas de Construção de Atributos Semânticos (Parte 1)

	W2V SGNS 100		**W2V SGNS 300**	
	Trn	**Val**	**Trn**	**Val**
Regressão Logística	89.01%	87.61%	89.32%	87.98%
Floresta Aleatória	99.40%	91.64%	99.40%	90.56%
SVM	93.07%	92.34%	93.60%	92.89%

Tabela 6.10: Acurácia em Treino e Validação dos Algoritmo de Aprendizado com Diversas Técnicas de Construção de Atributos Semânticos (Parte 2)

	FT CBOW 100		**W2V CBOW 300**	
	Trn	**Val**	**Trn**	**Val**
Regressão Logística	91.68%	91.16%	93.90%	93.44%
Floresta Aleatória	99.32%	89.16%	99.40%	91.49%
SVM	92.71%	92.23%	94.81%	94.24%

palavras como *doutorado*, *graduação*, *especialização*, uma vez que a semântica inferida por esse tipo de modelo diz respeito ao contexto das palavras. Por outro lado, nos modelos FastText, foi considerado como bom resultado quando a representação retornava palavras como *doutorado*, *mestra*, *semestralmente* e *especialização*, uma vez que a semântica inferida por esse tipo de modelo diz respeito ao contexto e a derivações morfológicas.

Quanto as operações algébricas vetoriais, algumas operações foram utilizadas para verificar coerência dos modelos, como: 1) $x_{medicina} - x_{médico} + x_{engenheiro}$; 2) $x_{criar} - x_{marketing} + x_{engenharia}$, 3) $x_{analisar} - x_{analista} + x_{desenvolvedor}$ e 4) $x_{enfermagem} - x_{enfermeiro} + x_{nutricionista}$. Para todos os modelos, foi considerado como bom resultado quando o resultado de pelo menos três das quatro operações acima faziam sentido. As operações 1) e 3) são simples e todos os modelos considerados convergidos conseguiram chegar nas respostas corretas $x_{engenharia}$ e $x_{desenvolver}$ respectivamente. Entretanto, modelos Word2Vec conseguem encontrar uma resposta aceitável para a operação 2) ($x_{construir}$), que depende de uma relação semântica baseada em contexto, mas não conseguem encontrar uma resposta aceitável para a operação 4) que depende de uma relação semântica baseada em derivações morfológicas. Os modelos FastText tem o comportamento contrário, por aprenderem uma semântica mista baseada em contexto e derivações morfológicas, e conseguem encontrar uma resposta aceitável para a operação 4) ($x_{nutrição}$).

Os resultados desses experimentos se encontram nas Tabelas 6.9 e 6.10. Todas as combinações consideradas tem resultados piores que suas respectivas versões com atributos léxicos. Isso pode ter acontecido por vários motivos: os textos das vagas de emprego não eram em quantidade suficiente ou não foram

treinados por épocas suficientes para inferir uma semântica ou o tamanho dos vetores semânticos estava inadequado. Independente disso, foi evidenciado que as relações semânticas eram coerentes e consistentes para todos esses modelos durante o treinamento. Em todo caso, a melhor combinação considerando somente os atributos semânticos atinge acurácia de validação de 94.24% e formada por SVM com *Word2Vec* modelo CBOW e vetores de tamanho 300.

6.2.4 Conjunto de Dados de Teste

A partir dos experimentos realizados neste trabalho, foi possível determinar a melhor combinação entre os pré-processamentos, as técnicas de construção de atributos e os algoritmos de aprendizado considerados. Essa combinação é dada por Regressão Logística com representação binária no Cenário A (sem remoção de *stop words* e sem stemização).

Antes de prosseguir com esse arranjo para avaliar a qualidade da segmentação derivada desse classificador, é importante estimar a capacidade de generalização do algoritmo e isso pode ser feito através de duas avaliações: acurácia no conjunto de teste de *'data-auto'*, para generalizar o resultado para sentenças novas vindas de vagas do mesmo *site*, e acurácia no conjunto *'data-man'*, para generalizar o resultado para sentenças vindas de vagas de outros *sites*.

Os resultados desses testes se encontram na Tabela 6.11. Nela pode ser observado que do treino para o teste e do teste para os conjuntos de sentenças com gabarito manual há uma degradação natural da acurácia por dois motivos: a) alguns dos dados são diferentes em vocabulário e formato, como exemplo, tem-se o LinkedIn com texto mais longos sobre as vagas e b) o gabarito foi elaborado manualmente o que possibilita uma classificação mais detalhada de sentenças anteriormente classificadas como Outros. Apesar disso, a acurácia está muito boa, por volta de 88%, dado que o treino foi feito usando dados de apenas um dos *sites* e com gabarito gerado automaticamente, sem supervisão.

Analisando as métricas Precisão e *Recall*, é possível perceber que a dificuldade na generalização reside justamente nas classes C0 e C3. Elas sofrem de problemas distintos: a classe C0 (Outros) não tem uma característica específica, abrangendo tudo que não puder ser classificado em nenhuma das outras classes, e a classe C3 (Benefícios) é muito específica no conjunto de treinamento, normalmente se restringindo a sentenças enumerando *tíquete*, *vale*, *refeição*, *alimentação*, mas nos outros conjuntos é mais abrangente, envolvendo benefícios indiretos como *ser contratado como PJ* ou *existe refeitório no local de trabalho*. Essas classes, apesar de importantes, não são as mais importantes

para induzir atributos semânticos para vagas de emprego conforme representado na Figura 2.2. As classes C1 e C2 são as mais relevantes e essas estão com métricas consistentes, indicando que o classificador consegue generalizar o comportamento delas a partir dos dados utilizados para treinamento.

6.3 Segmentação de Texto

Resta avaliar o classificador encontrado na tarefa de Segmentação de Texto, especificamente considerando a semântica de cada segmento. Para avaliá-lo foram usadas as vagas presentes no conjunto de teste de *data-auto*. Cada vaga foi então dividida em sentenças, suas sentenças foram classificadas e foram gerados dois vetores de resultados: um com a classificação de todas as suas sentenças e outro com a informação sobre início de cada segmento. Esses vetores estão ilustrados na Figura **??** e usando eles foram calculadas as métricas Precisão por classe, *Recall* por classe, P_k e *WindowDiff*.

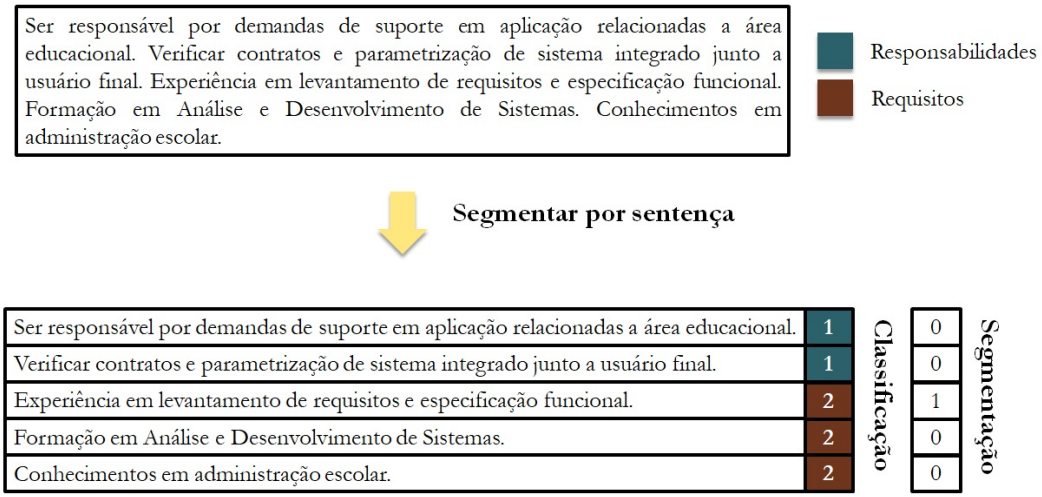

Figura 6.2: Representação do Uso de Classificador para Segmentação de Texto

Tabela 6.11: Acurácia, Precisão e *Recall* em Teste do Melhor Algoritmo de Aprendizado

		Acc	**Prec**				***Recall***			
			C0	**C1**	**C2**	**C3**	**C0**	**C1**	**C2**	**C3**
data-auto	**treino**	96.61%	96.01%	96.75%	96.34%	99.59%	91.37%	98.18%	95.50%	95.71%
	teste	95.58%	94.27%	95.86%	95.28%	98.73%	88.93%	97.44%	94.23%	92.91%
data-man	**Catho**	88.14%	31.80%	92.61%	96.90%	99.40%	60.81%	98.08%	83.89%	69.33%
	LinkedIn	91.14%	81.08%	90.01%	95.56%	85.71%	70.40%	98.21%	93.24%	18.95%
	VAGAS.com.br	88.60%	62.65%	89.82%	95.05%	100%	70.51%	96.95%	90.73%	9.91%

Tabela 6.12: *WindowDiff*, P_k, Precisão e *Recall* em Teste do Melhor Classificador aplicado a Segmentação

data-auto			**Precisão(%)**	***Recall (%)***
WindowDiff (%)	4.78 ± 13.97	**C0**	97.08 ±14.41	94.10 ± 21.06
		C1	94.47 ± 15.00	98.06 ± 10.36
Pk(%)	3.67 ± 11.05	**C2**	97.85 ± 9.35	97.47 ± 10.91
		C3	99.91 ± 2.47	90.49 ± 29.15

Os resultados da aplicação do melhor classificador para segmentação semântica das vagas do conjunto de teste encontram-se na Tabela 6.12. Nela é possível perceber que o classificador, após a segmentação em sentenças, é muito competente em segmentar a descrição da vaga de emprego, alcançando valores das métricas P_k e *WindowDiff* comparáveis ao estado da arte na tarefa. Chama atenção o grande desvio padrão dessas métricas, mas ele indica uma assimetria positiva nas suas distribuições, o que é uma evidência de que apenas eventualmente os valores das métricas serão piores que os encontrados.

7
Conclusões e Trabalhos Futuros

Uma das principais contribuições deste trabalho foi a elaboração de um fluxo de trabalho para coleta, processamento e análise dos dados de vagas de emprego, que se inicia com a criação de uma base de dados utilizando *crawlers* e se encerra com a aplicação de algoritmos de aprendizado de máquina para diversas tarefas de NLP. Esse fluxo deriva diretamente do processo de busca de resposta para a questão principal da pesquisa, citada no Capítulo 1.

Pensando especificamente na coleta de dados dos diversos *sites* de vagas de emprego, este trabalho propõe uma estrutura geral de *crawler*, que foi aplicada para pelo menos três *sites* (Catho, LinkedIn e VAGAS.com.br) com bastante facilidade de adaptação para cada cenário.

Prosseguindo, este trabalho analisa como a literatura modela vagas de emprego. Com base nas taxonomias e ontologias estudadas, propõe-se uma semântica simples, em que a vaga tem quatro seções (Responsabilidades, Requisitos, Benefícios e Outros). Essas classes por si só, trazem um grande ganho no entendimento automático das vagas de emprego.

Para segmentar a vaga em cada uma seções citadas acima, evitou-se o uso de técnicas de Segmentação de Texto, por causa da natureza do texto da vaga de emprego: curtos, simples e repetitivos. Em vez disso, este trabalho repensa a segmentação de texto e a decompõe em duas novas tarefas seriadas: segmentação em sentenças e classificação de texto.

Apesar do texto da vaga de emprego se distanciar da boa gramática, este trabalho insistiu no uso de técnicas famosas de construção de atributos léxicos e semânticos para representar esses textos que não seguem a norma culta. O resultado é que apesar de parecerem simples ou limitadas, essas técnicas podem ser suficientes para permitir o treino de modelos cujos resultados são comparáveis ao estado da arte das tarefas consideradas. No caso, foi obtida uma acurácia de 95.58% com os dados do Catho.

A partir desse classificador, este trabalho mostrou evidências de que o aprendizado por ele adquirido é generalizável para dados de vagas de emprego vindos de outros *sites*. Por exemplo, foi obtida uma acurácia de 88.60% com os dados do VAGAS.com.br e 91.14% com os dados do LinkedIn.

Ainda com o classificador, foi verificada a eficiência dele na tarefa de

segmentação semântica, mostrando que além de um bom classificador, ele é um bom segmentador semântico. As métricas P_k e *WindowDiff* obtidas foram de 3.66% e 4.78%.

Além disso, este trabalho teve uma contribuição implícita de mostrar que mesmo algoritmos de aprendizado clássicos e tradicionais como a Regressão Logística e o Naïve Bayes podem alcançar resultados comparáveis o estado da arte da tarefa de Classificação de Texto e, portanto, sempre devem ser experimentados.

Como trabalhos futuros, propõe-se:

- Desenvolver as próximas etapas do fluxo de trabalho representado na Figura 2.2: Extração de Atributos das Sentenças Classificadas e Representação Vetorial Semântica;
- Explorar a classe Outros e buscar a presença consistente de outras classes não rotuladas;
- Explorar mais os modelos de construção de atributos semânticos de forma a melhorar seus resultados;
- Explorar mais a dependência e probabilidade condicional como uma forma de melhorar a classificação;
- Utilizar redes neurais profundas para construção de atributos automaticamente, encapsulando, na Figura 2.2, desde a etapa do Segmentador até a Extração de Atributos 2.

Referências bibliográficas

[1] NÜTZI, M.; TREZZINI, B.; MEDICI, L.; SCHWEGLER, U.. **Job matching: An interdisciplinary scoping study with implications for vocational rehabilitation counseling**. Rehabilitation Psychology, 62(1):45–68, 2017.

[2] GREENBERG, H. M.. **Job matching to hire motivated employees**. National Precast Concrete Association, 2010. Acesso em: Abril de 2019.

[3] FURTMUELLER, E.; WILDEROM, C.; TATE, M.. **Managing recruitment and selection in the digital age: e-hrm and resumes**. Human Systems Management, 30(4):243–259, 2011.

[4] KLINGNER, R. M.; NALBANDIAN, J.; LLORENS, J.. **Public Personnel Management: contexts and strategies**. Routledge, New York, 6th edition, 2010.

[5] AHMED, N.; KHAN, S.; LATIF, K.. **Job description ontology**. In: 14TH INTERNATIONAL CONFERENCE ON FRONTIERS OF INFORMATION TECHNOLOGY (FIT), p. 217–222, Islamabad, Pakistan, 2016.

[6] CHOLLET, F.. **Deep Learning with Python**. Manning Publications, USA, 1st edition, 2018.

[7] HISLOP, D.. **Knowledge Management in Organizations: A Critical Introduction**. OUP Oxford, 2013.

[8] ŞAHIN, A.; SONG, J.; TOPA, G.; VIOLANTE, G. L.. **Mismatch unemployment**. The American Economic Review, 104(11):3529–3564, 2014.

[9] MANROOP, L.; RICHARDSON, J.. **Job search: A multidisciplinary review and research agenda**. International Journal of Management Reviews, 18(2):206—227, 2015.

[10] PRESIDENT, U. S.. **Manpower Report of the President**. U.S. Government Printing Office, 1970.

[11] EDUWORKS. **Eduworks network**. Organisation for Economic Co-operation and Development, 2016. Acesso em: Abril de 2019.

[12] CATHO. **Pesquisa dos profissionais brasileiros**. Catho, 2014. Acesso em: Abril de 2019.

[13] VAGAS.COM.BR. **Sobre vagas.com.br**. VAGAS.com.br, 2019. Acesso em: Abril de 2019.

[14] FONSECA, M.. **O cenário de rh**. Revista EXAME, 2018. Acesso em: Abril de 2019.

[15] LINKEDIN. **Sobre o linkedin**. LinkedIn, 2019. Acesso em: Abril de 2019.

[16] FONSECA, A.. **Como as hr techs estão aproveitando um setor carente em inovação e tecnologia para se desenvolver**. Projeto DRAFT, 2019. Acesso em: Abril de 2019.

[17] HEATH, T.; BIZER, C. . **Linked data: Evolving the web into a global data space**. Synthesis lectures on the semantic web: theory and technology, 1(1):1—136, 2011.

[18] VAGAS.COM.BR. **Mapa de carreiras**. VAGAS.com.br, 2019. Acesso em: Abril de 2019.

[19] CATHO. **Guia de profissões**. Catho, 2019. Acesso em: Abril de 2019.

[20] SANTOS, M. B.. **Beyond skill mismatch. why there are so many unfilled vacancies and simultaneously high unemployment rates?** R-LEGO Revista Lusófona de Economia e Gestão das Organizacoes, p. 9–27, 2016.

[21] CATHO. **Busca de vaga de emprego**. Catho, 2019. Acesso em: Abril de 2019.

[22] KOSHOREK, O.; COHEN, A.; MOR, N.; ROTMAN, M. ; BERANT, J.. **Text segmentation as a supervised learning task**. In: PROCEEDINGS OF THE 2018 CONFERENCE OF THE NORTH AMERICAN CHAPTER OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS: HUMAN LANGUAGE TECHNOLOGIES, VOLUME 2 (SHORT PAPERS), p. 469–473, New Orleans, Louisiana, 2018. Association for Computational Linguistics.

[23] STOKES, N.; CARTHY, J.; SMEATON, A. F.. **Select: A lexical cohesion based news story segmentation system**. AI Commun., 17(1):3–12, 2004.

[24] KOZIMA, H.. **Text segmentation based on similarity between words**. In: PROCEEDINGS OF THE 31ST ANNUAL MEETING ON ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, ACL '93, p. 286–288, Stroudsburg, PA, USA, 1993. Association for Computational Linguistics.

[25] HEARST, M. A.. **Text tiling: Segmenting text into multi-paragraph subtopic passages**. Computational Linguistics, 23(1):33–64, 1997.

[26] CHOI, F. Y. Y.. **Advances in domain independent linear text segmentation**. In: PROCEEDINGS OF THE 1ST NORTH AMERICAN CHAPTER OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS CONFERENCE, NAACL 2000, p. 26–33, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[27] MALIOUTOV, I.; BARZILAY, R.. **Minimum cut model for spoken lecture segmentation**. In: PROCEEDINGS OF THE 21ST INTERNATIONAL CONFERENCE ON COMPUTATIONAL LINGUISTICS AND 44TH ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, p. 25–32, Sydney, Australia, 2006. Association for Computational Linguistics.

[28] MISRA, H.; YVON, F.; CAPPÉ, O. ; JOSE, J.. **Text segmentation: A topic modeling perspective**. Information Processing & Management, 47(4):528–544, 2011.

[29] JURAFSKY, D.;MARTIN, J. H.. **Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition**. Prentice Hall, 2 edition, 2008.

[30] INMON, W.H. AND NESAVICH, A.. **Tapping into Unstructured Data: Integrating Unstructured Data and Textual Analytics into Business Intelligence**. Pearson Education, 2007.

[31] GUO, S., A. F. . H. T.. **Résumatcher: A personalized résumé-job matching system**. Expert Systems with Applications, 60(4):169–182, 2016.

[32] JACINTHO, E. M.; GONZALEZ, J. M.. **Oferta de emprego: habilidades necessárias para arquivistas em empresas no brasil**. Informação & Informação, 24(1):424–441, 2019.

[33] FANK, D. R. B.; WERNKE, R. ; ZANIN, A.. **Funções do controller no brasil e na argentina: Comparativo com base em anúncios de sites de empregos**. Management Control Review, 2(2):2–17, 2018.

[34] DO VALE, J. W. S. P.. **Competências dos gerentes de projetos: revisão de literatura, análise de oportunidades de emprego e estudos de casos**. Dissertação de mestrado, Departamento de Engenharia de Produção, Escola Politécnica, São Paulo, 2015.

[35] DBPEDIA. **About**. DBpedia, 2019. Acesso em: Abril de 2019.

[36] ERTUĞRUL, D. C.; ELÇI, A.. **An ontology-based information extraction approach for résumés**. In: LECTURE NOTES IN COMPUTER SCIENCE, volumen 7719, p. 165–179, 2012.

[37] SINGH, A.; CATHERINE, R.; VISWESWARIAH, K.; CHENTHAMARAKSHAN, V. ; KAMBHATLA, N.. **Prospect: A system for screening candidates for recruitment**. In: PROCEEDINGS OF THE 19TH ACM CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT, p. 659–668, 2010.

[38] SÁNCHEZ, D.; BATET, M.; ISERN, D. ; VALLS, A.. **Ontology-based semantic similarity: A new feature-based approach**. Expert Systems with Applications, 39:7718–7728, 03 2012.

[39] LU, Y.; EL HELOU, S. ; GILLET, D.. **A recommender system for job seeking and recruiting website**. In: PROCEEDINGS OF THE 22ND INTERNATIONAL CONFERENCE ON WORLD WIDE WEB, WWW '13 Companion, p. 963–966, New York, NY, USA, 2013. ACM.

[40] YI, X.; ALLAN, J. ; CROFT, W.. **Matching resumes and jobs based on relevance models**. p. 809–810, 01 2007.

[41] DENG, L.; LIU, Y.. **Deep Learning in Natural Language Processing**. Springer, Singapore, 1st edition, 2018.

[42] KESSLER, R.; TORRES-MORENO, J. M. ; EL-BÈZE, M.. **E-gen: Automatic job offer processing system for human resources**. In: Gelbukh, A.; Kuri Morales, Á. F., editors, MICAI 2007: ADVANCES IN ARTIFICIAL INTELLIGENCE, p. 985–995, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[43] BEKKERMAN, R.; GAVISH, M.. **High-precision phrase-based document classification on a modern scale**. In: PROCEEDINGS OF THE

17TH ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, KDD '11, p. 231–239, New York, NY, USA, 2011. ACM.

[44] WOWCZKO, I.. **Skills and vacancy analysis with data mining techniques**. Informatics, 2:31–49, 11 2015.

[45] JAVED, F.; MCNAIR, M.; JACOB, F. ; ZHAO, M.. **Towards a job title classification system**. CoRR, abs/1606.00917, 2016.

[46] YU, K.; GUAN, G. ; ZHOU, M.. **Resume information extraction with cascaded hybrid model**. In: PROCEEDINGS OF THE 43RD ANNUAL MEETING ON ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, ACL '05, p. 499–506, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[47] LI, J.; SUN, A. ; JOTY, S.. **Segbot: A generic neural text segmentation model with pointer network**. In: PROCEEDINGS OF THE 27TH INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE AND THE 23RD EUROPEAN CONFERENCE ON ARTIFICIAL INTELLIGENCE, IJCAI-ECAI-2018, Stockholm, Sweden, July 2018.

[48] HEARST, M. A.. **Text tiling: Segmenting text into multi-paragraph subtopic passages**. Computational Linguistics, 23(1):33–64, 1997.

[49] HEARST, M. A.. **Multi-paragraph segmentation of expository text**. In: PROCEEDINGS OF THE 32ND ANNUAL MEETING ON ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, ACL '94, p. 9–16, Stroudsburg, PA, USA, 1994. Association for Computational Linguistics.

[50] REYNAR, J. C.. **Topic Segmentation: Algorithms and Applications**. PhD thesis, Philadelphia, PA, USA, 1998. AAI9829978.

[51] EISENSTEIN, J.; BARZILAY, R.. **Bayesian unsupervised topic segmentation**. In: PROCEEDINGS OF THE CONFERENCE ON EMPIRICAL METHODS IN NATURAL LANGUAGE PROCESSING, EMNLP '08, p. 334–343, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[52] BEEFERMAN, D.; BERGER, A. ; LAFFERTY, J.. **Statistical models for text segmentation**. Machine Learning, 34(1):177–210, Feb 1999.

[53] UTIYAMA, M.; ISAHARA, H.. **A statistical model for domain-independent text segmentation**. In: PROCEEDINGS OF THE 39TH ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, p. 499–506, Toulouse, France, July 2001. Association for Computational Linguistics.

[54] KAZANTSEVA, A.; SZPAKOWICZ, S.. **Linear text segmentation using affinity propagation**. In: PROCEEDINGS OF THE CONFERENCE ON EMPIRICAL METHODS IN NATURAL LANGUAGE PROCESSING, EMNLP '11, p. 284–293, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[55] SAKAHARA, M.; OKADA, S. ; NITTA, K.. **Domain-independent unsupervised text segmentation for data management**. In: 2014 IEEE INTERNATIONAL CONFERENCE ON DATA MINING WORKSHOP, p. 481–487, Dec 2014.

[56] PURVER, M.; GRIFFITHS, T. L.; KÖRDING, K. P. ; TENENBAUM, J. B.. **Unsupervised topic modelling for multi-party spoken discourse**. In: PROCEEDINGS OF THE 21ST INTERNATIONAL CONFERENCE ON COMPUTATIONAL LINGUISTICS AND THE 44TH ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, ACL-44, p. 17–24, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[57] MISRA, H.; YVON, F.; JOSE, J. M. ; CAPPE, O.. **Text segmentation via topic modeling: An analytical study**. In: PROCEEDINGS OF THE 18TH ACM CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT, CIKM '09, p. 1553–1556, New York, NY, USA, 2009. ACM.

[58] RIEDL, M.; BIEMANN, C.. **TopicTiling: A text segmentation algorithm based on LDA**. In: PROCEEDINGS OF ACL 2012 STUDENT RESEARCH WORKSHOP, p. 37–42, Jeju Island, Korea, July 2012. Association for Computational Linguistics.

[59] GROSZ, B.; HIRSCHBERG, J.. **Some intonational characteristics of discourse structure**. 03 1997.

[60] TUR, G.; HAKKANI-TUR, D.; STOLCKE, A. ; SHRIBERG, E.. **Integrating prosodic and lexical cues for automatic topic segmentation**. CoRR, cs.CL/0105037, 03 2001.

[61] REYNAR, J. C.. **Statistical models for topic segmentation**. In: PROCEEDINGS OF THE 37TH ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS ON COMPUTATIONAL LINGUISTICS, ACL '99, p. 357–364, Stroudsburg, PA, USA, 1999. Association for Computational Linguistics.

[62] MOCHIZUKI, H.; HONDA, T. ; OKUMURA, M.. **Text segmentation with multiple surface linguistic cues**. In: COLING-ACL, 1998.

[63] LI, J.; SUN, A. ; JOTY, S.. **Segbot: A generic neural text segmentation model with pointer network**. In: PROCEEDINGS OF THE 27TH INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE AND THE 23RD EUROPEAN CONFERENCE ON ARTIFICIAL INTELLIGENCE, IJCAI-ECAI-2018, p. xx – xx, Stockholm, Sweden, July 2018.

[64] BADJATIYA, P.; KURISINKEL, L. J.; GUPTA, M. ; VARMA, V.. **Attention-based neural text segmentation**. In: ECIR, 2018.

[65] PEVZNER, L.; HEARST, M. A.. **A critique and improvement of an evaluation metric for text segmentation**. Computational Linguistics, 28(1):19–36, 2002.

[66] SCHENKER, A.; BUNKE, H.; LAST, M. ; KANDEL, A.. **Graph-Theoretic Techniques for Web Content Mining**. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2005.

[67] MIHALCEA, R. F.; RADEV, D. R.. **Graph-based Natural Language Processing and Information Retrieval**. Cambridge University Press, New York, NY, USA, 1st edition, 2011.

[68] CHANG, M.; POON, C.. **Using phrases as features in email classification**. Journal of Systems and Software, 82:1036–1045, 06 2009.

[69] FIGUEIREDO, F.; ROCHA, L.; COUTO, T.; SALLES, T.; GONÇALVES, M. ; MEIRA JR, W.. **Word co-occurrence features for text classification**. Inf. Syst., 36:843–858, 07 2011.

[70] LEE, L. H.; ISA, D.; CHOO, W. ; CHUE, W.. **High relevance keyword extraction facility for bayesian text classification on different domains of varying characteristic**. Expert Systems with Applications: An International Journal, 39:1147–1155, 01 2012.

[71] ONAN, A.; KORUKOĞLU, S. ; BULUT, H.. **Ensemble of keyword extraction methods and classifiers in text classification**. Expert Syst. Appl., 57(C):232–247, Sept. 2016.

[72] XIE, F.; WU, X. ; ZHU, X.. **Efficient sequential pattern mining with wildcards for keyphrase extraction**. Knowledge-Based Systems, 115, 10 2016.

[73] LI, C. H.; YANG, J. C. ; PARK, S. C.. **Text categorization algorithms using semantic approaches, corpus-based thesaurus and wordnet**. Expert Syst. Appl., 39(1):765–772, Jan. 2012.

[74] CAGLIERO, L.; GARZA, P.. **Improving classification models with taxonomy information**. Data & Knowledge Engineering, 86:85–101, 07 2013.

[75] DE KNIJFF, J.; FRASINCAR, F. ; HOGENBOOM, F.. **Domain taxonomy learning from text: The subsumption method versus hierarchical clustering**. Data Knowl. Eng., 83:54–69, Jan. 2013.

[76] LIU, J.; HE, Y.-L.; LIM, E. ; WANG, X.-Z.. **Domain ontology graph model and its application in chinese text classification**. Neural Computing and Applications, 24, 03 2014.

[77] SÁNCHEZ-PI, N.; MARTÍ, L. ; GARCIA, A. C. B.. **Improving ontology-based text classification: An occupational health and security application**. Journal of Applied Logic, 17:48–58, 09 2015.

[78] KANG, Y. B.; DELIR HAGHIGHI, P. ; BURSTEIN, F.. **Taxofinder: A graph-based approach for taxonomy learning**. IEEE Transactions on Knowledge and Data Engineering, p. 1–1, 01 2015.

[79] SALEH, A.; ALRAHMAWY, M. ; ABULWAFA, A.. **A semantic based web page classification strategy using multi-layered domain ontology**. World Wide Web, 20:1–55, 10 2016.

[80] WU, Z.; ZHU, H.; LI, G.; CUI, Z.; HUANG, H.; LI, J.; CHEN, E. ; XU, G.. **An efficient wikipedia semantic matching approach to text document classification**. Inf. Sci., 393(C):15–28, July 2017.

[81] BANDHAKAVI, A. S.; WIRATUNGA, N.; P, D. ; MASSIE, S.. **Lexicon based feature extraction for emotion text classification**. Pattern Recognition Letters, 93, 12 2016.

[82] MANEK, A.; SHENOY, P.; MOHAN, M. ; K R, V.. **Aspect term extraction for sentiment analysis in large movie reviews using gini index feature selection method and svm classifier**. World Wide Web, 20, 02 2016.

[83] AGATHANGELOU, P.; KATAKIS, I.; KOUTOULAKIS, I.; KOKKORAS, F. ; GUNOPULOS, D.. **Learning patterns for discovering domain-oriented opinion words**. Knowl. Inf. Syst., 55(1):45–77, Apr. 2018.

[84] BARONI, M.; DINU, G. ; KRUSZEWSKI, G.. **Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors**. In: PROCEEDINGS OF THE 52ND ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS (VOLUME 1: LONG PAPERS), p. 238–247, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[85] MIKOLOV, T.; YIH, W.-T. ; ZWEIG, G.. **Linguistic regularities in continuous space word representations**. In: PROCEEDINGS OF THE 2013 CONFERENCE OF THE NORTH AMERICAN CHAPTER OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS: HUMAN LANGUAGE TECHNOLOGIES, p. 746–751, Atlanta, Georgia, June 2013. Association for Computational Linguistics.

[86] WANG, P.; XU, J.; XU, B.; LIU, C.; ZHANG, H.; WANG, F. ; HAO, H.. **Semantic clustering and convolutional neural network for short text categorization**. In: PROCEEDINGS OF THE 53RD ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS AND THE 7TH INTERNATIONAL JOINT CONFERENCE ON NATURAL LANGUAGE PROCESSING (VOLUME 2: SHORT PAPERS), p. 352–357, Beijing, China, July 2015. Association for Computational Linguistics.

[87] CHATURVEDI, I.; ONG, Y.; TSANG, I.; WELSCH, R. ; CAMBRIA, E.. **Learning word dependencies in text by means of a deep recurrent belief network**. Knowledge-Based Systems, 108, 07 2016.

[88] TOMMASEL, A.; GODOY, D.. **Short-text feature construction and selection in social media data: a survey**. Artificial Intelligence Review, 11 2016.

[89] ENRIQUEZ, F.; TROYANO, J. ; LÓPEZ-SOLAZ, T.. **An approach to the use of word embeddings in an opinion classification task**. Expert Systems with Applications, 66:1–6, 12 2016.

[90] PENNINGTON, J.; SOCHER, R. ; MANNING, C.. **Glove: Global vectors for word representation**. In: PROCEEDINGS OF THE 2014 CONFERENCE ON EMPIRICAL METHODS IN NATURAL LANGUAGE PROCESSING (EMNLP), p. 1532–1543, Doha, Qatar, Oct. 2014. Association for Computational Linguistics.

[91] BOJANOWSKI, P.; GRAVE, E.; JOULIN, A. ; MIKOLOV, T.. **Enriching word vectors with subword information**. Transactions of the Association for Computational Linguistics, 5:135–146, Dec 2017.

[92] JOULIN, A.; GRAVE, E.; BOJANOWSKI, P. ; MIKOLOV, T.. **Bag of tricks for efficient text classification**. Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers, 2017.

[93] HOWARD, J.; RUDER, S.. **Universal language model fine-tuning for text classification**. Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2018.

[94] DEVLIN, J.; CHANG, M.-W.; LEE, K. ; TOUTANOVA, K.. **Bert: Pre-training of deep bidirectional transformers for language understanding**, 2018.

[95] PETERS, M. E.; NEUMANN, M.; IYYER, M.; GARDNER, M.; CLARK, C.; LEE, K. ; ZETTLEMOYER, L.. **Deep contextualized word representations**. In: PROC. OF NAACL, 2018.

[96] YANG, Z.; DAI, Z.; YANG, Y.; CARBONELL, J.; SALAKHUTDINOV, R. ; LE, Q. V.. **Xlnet: Generalized autoregressive pretraining for language understanding**, 2019.

[97] SAHA, S.; EKBAL, A.. **Combining multiple classifiers using vote based classifier ensemble technique for named entity recognition**. Data Knowl. Eng., 85:15–39, May 2013.

[98] KRÁL, P.. **Named entities as new features for czech document classification**. In: PROCEEDINGS OF THE 15TH INTERNATIONAL CONFERENCE ON COMPUTATIONAL LINGUISTICS AND INTELLIGENT TEXT PROCESSING - VOLUME 8404, CICLing 2014, p. 417–427, Berlin, Heidelberg, 2014. Springer-Verlag.

[99] GUI, Y.; GAO, Z.; LI, R. ; YANG, X.. **Hierarchical text classification for news articles based-on named entities**. p. 318–329, 12 2012.

[100] ZUO, Y.; ZHAO, J. ; XU, K.. **Word network topic model: a simple but general solution for short and imbalanced texts**. Knowledge and Information Systems, 48(2):379–398, Sep 2015.

[101] ZHANG, H.; ZHONG, G.. **Improving short text classification by learning vector representations of both words and hidden topics**. Knowledge-Based Systems, 102, 03 2016.

[102] QIN, Z.; CONG, Y. ; WAN, T.. **Topic modeling of chinese language beyond a bag-of-words**. Computer Speech & Language, 40, 04 2016.

[103] PAVLINEK, M.; PODGORELEC, V.. **Text classification method based on self-training and lda topic models**. Expert Syst. Appl., 80(C):83–93, Sept. 2017.

[104] DUIN, R.; LOOG, M.; PEKALSKA, E. ; TAX, D.. **Feature-Based Dissimilarity Space Classification**, p. 46–55. 01 1970.

[105] PINHEIRO, R. H.; CAVALCANTI, G. D. ; TSANG, I. R.. **Combining dissimilarity spaces for text categorization**. Inf. Sci., 406(C):87–101, Sept. 2017.

[106] WANG, P.; HU, J.; ZENG, H.-J. ; CHEN, Z.. **Using wikipedia knowledge to improve text classification**. Knowl. Inf. Syst., 19:265–281, 06 2009.

[107] KO, Y.; PARK, J. ; SEO, J.. **Improving text categorization using the importance of sentences**. Information Processing & Management, 40:65–79, 01 2004.

[108] RAO, Y.; LI, Q.; WU, Q.; XIE, H.; WANG, F. L. ; WANG, T.. **A multi-relational term scheme for first story detection**. Neurocomputing, 254:42 – 52, 2017. Recent Advances in Semantic Computing and Personalization.

[109] SABBAH, T.; SELAMAT, A.; SELAMAT, M. H.; AL-ANZI, F.; HERRERA-VIEDMA, E.; KREJCAR, O. ; FUJITA, H.. **Modified frequency-based term weighting schemes for text classification**. Applied Soft Computing, 58, 05 2017.

[110] ZHANG, L.; JIANG, L.; LI, C. ; KONG, G.. **Two feature weighting approaches for naive bayes text classifiers**. Knowledge-Based Systems, 100:137–144, 03 2016.

[111] CHEN, K.; ZHANG, Z.; LONG, J. ; ZHANG, H.. **Turning from tf-idf to tf-igm for term weighting in text classification**. Expert Systems with Applications, 66:245 – 260, 2016.

[112] KIM, H. K.; KIM, M.. **Model-induced term-weighting schemes for text classification**. Applied Intelligence, 45(1):30–43, July 2016.

[113] JIANG, L.; LI, C.; WANG, S. ; ZHANG, L.. **Deep feature weighting for naive bayes and its application to text classification**. Eng. Appl. Artif. Intell., 52(C):26–39, June 2016.

[114] ABDEL FATTAH, M.. **New term weighting schemes with combination of multiple classifiers for sentiment analysis**. Neurocomput., 167(C):434–442, Nov. 2015.

[115] KE, W.. **Information-theoretic term weighting schemes for document clustering**. p. 1–10, 07 2013.

[116] ESCALANTE, H. J.; GARCÍA-LIMÓN, M.; MORALES-REYES, A.; GRAFF, M.; MONTES, M. ; MORALES, E.. **Term-weighting learning via genetic programming for text classification**. Knowledge-Based Systems, 83, 10 2014.

[117] REN, F.; SOHRAB, M.. **Class-indexing-based term weighting for automatic text classification**. Information Sciences, 236:109–125, 07 2013.

[118] LUO, Q.; CHEN, E. ; XIONG, H.. **A semantic term weighting scheme for text categorization**. Expert Syst. Appl., 38(10):12708–12716, Sept. 2011.

[119] BELLOTTI, T.; NOURETDINOV, I.; YANG, M. ; GAMMERMAN, A.. **Chapter 6 - feature selection**. In: Balasubramanian, V. N.; Ho, S.-S. ; Vovk, V., editors, CONFORMAL PREDICTION FOR RELIABLE MACHINE LEARNING, p. 115 – 130. Morgan Kaufmann, Boston, 2014.

[120] GUYON, I.; ELISSEEFF, A.. **An introduction to variable and feature selection**. J. Mach. Learn. Res., 3:1157–1182, Mar. 2003.

[121] LI, Y.; LI, T. ; LIU, H.. **Recent advances in feature selection and its applications**. Knowledge and Information Systems, 53, 05 2017.

[122] ZOU, H.; HASTIE, T.. **Regularization and variable selection via the elastic net (vol b 67, pg 301, 2005)**. Journal of the Royal Statistical Society Series B, 67:768–768, 02 2005.

[123] MANNING, C. D.; RAGHAVAN, P. ; SCHÜTZE, H.. **Introduction to Information Retrieval**. Cambridge University Press, New York, NY, USA, 2008.

[124] BOUMA, G.. **Normalized (pointwise) mutual information in collocation extraction**. Proceedings of the Biennial GSCL Conference 2009, 01 2009.

[125] LI, S.; XIA, R.; ZONG, C. ; HUANG, C.-R.. **A framework of feature selection methods for text categorization**. In: PROCEEDINGS OF THE JOINT CONFERENCE OF THE 47TH ANNUAL MEETING OF THE ACL AND THE 4TH INTERNATIONAL JOINT CONFERENCE ON NATURAL LANGUAGE PROCESSING OF THE AFNLP, p. 692–700, Suntec, Singapore, Aug. 2009. Association for Computational Linguistics.

[126] VERGARA, J. R.; ESTÉVEZ, P. A.. **A review of feature selection methods based on mutual information**. Neural Computing and Applications, 24(1):175–186, Mar 2013.

[127] ZHOU, Y.; JIN, R. ; HOI, S.. **Exclusive lasso for multitask feature selection**. Journal of Machine Learning Research - JMLR, 9:988–995, 01 2010.

[128] LABANI, M.; MORADI, P.; AHMADIZAR, F. ; JALILI, M.. **A novel multivariate filter method for feature selection in text classification problems**. Engineering Applications of Artificial Intelligence, 70:25 – 37, 2018.

[129] JALILVAND, A.; SALIM, N.. **Feature unionization: A novel approach for dimension reduction**. Applied Soft Computing, 52:1253 – 1261, 2017.

[130] ŠAJGALÍK, M.; BARLA, M. ; BIELIKOVA, M.. **Searching for discriminative words in multidimensional continuous feature space**. Computer Speech & Language, 53, 10 2017.

[131] REHMAN, A.; JAVED, K. ; BABRI, H.. **Feature selection based on a normalized difference measure for text classification**. Information Processing & Management, 53:473–489, 03 2017.

[132] AGNIHOTRI, D.. **Variable global feature selection scheme for automatic classification of text documents**. Expert Systems with Applications, 81:268–281, 03 2017.

[133] TUTKAN, M.; GANIZ, M. C. ; AKYOKUŞ, S.. **Helmholtz principle based supervised and unsupervised feature selection methods for text mining**. Inf. Process. Manage., 52(5):885–910, Sept. 2016.

[134] GHAREB, A.; ABU BAKAR, A. ; HAMDAN, A.. **Hybrid feature selection based on enhanced genetic algorithm for text categorization**. Expert Systems with Applications, 49, 12 2015.

[135] UYSAL, A.. **An improved global feature selection scheme for text classification**. Expert Systems with Applications, 43, 09 2015.

[136] BURGES, C.. **Dimension reduction: A guided tour**. Foundations and Trends in Machine Learning, 2, 01 2010.

[137] CUNNINGHAM, J. P.; GHAHRAMANI, Z.. **Linear dimensionality reduction: Survey, insights, and generalizations**. Journal of Machine Learning Research, 16(89):2859–2900, 2015.

[138] HASTIE, T.; TIBSHIRANI, R. ; FRIEDMAN, J.. **The Elements of Statistical Learning: Data Mining, Inference, and Prediction**. Springer series in statistics. Springer, 2009.

[139] CHANG, X.; NIE, F.; YANG, Y.; ZHANG, C. ; HUANG, H.. **Convex sparse pca for unsupervised feature learning**. ACM Trans. Knowl. Discov. Data, 11(1):3:1–3:16, July 2016.

[140] GUAN, H.; XIAO, B.; ZHOU, J.; GUO, M. ; YANG, T.. **Fast dimension reduction for document classification based on imprecise spectrum analysis**. volumen 222, p. 1753–1756, 01 2010.

[141] VAN DER MAATEN, L.; HINTON, G.. **Visualizing high-dimensional data using t-sne**. Journal of Machine Learning Research, 9(nov):2579–2605, 2008. Pagination: 27.

[142] VAN DER MAATEN, L.. **Accelerating t-sne using tree-based algorithms**. J. Mach. Learn. Res., 15(1):3221–3245, Jan. 2014.

[143] KONTOSTATHIS, A.; POTTENGER, W. M.. **A framework for understanding latent semantic indexing (lsi) performance**. Information Processing & Management, 42(1):56 – 73, 2006. Formal Methods for Information Retrieval.

[144] SCHÖLKOPF, B.; SMOLA, A. ; MÜLLER, K.-R.. **Nonlinear component analysis as a kernel eigenvalue problem**. Neural Computation, 10:1299–1319, 07 1998.

[145] BINGHAM, E.; MANNILA, H.. **Random projection in dimensionality reduction: Applications to image and text data**. In: PROCEEDINGS OF THE SEVENTH ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, KDD '01, p. 245–250, New York, NY, USA, 2001. ACM.

[146] HOSMER, D.; LEMESHOW, S.. **Applied Logistic Regressio**, volumen 85. 10 2004.

[147] HARRELL, JR., F. E.. **Regression Modeling Strategies**. Springer-Verlag, Berlin, Heidelberg, 2006.

[148] CHEN, W.; XIE, X.; WANG, J.; PRADHAN, B.; HONG, H.; BUI, D. T.; DUAN, Z. ; MA, J.. **A comparative study of logistic model tree, random forest, and classification and regression tree models for spatial prediction of landslide susceptibility**. CATENA, 151:147 – 160, 2017.

[149] DOU, J.; YAMAGISHI, H.; ZHU, Z.; YUNUS, A. P. ; CHEN, C.. **TXT-tool 1.081-6.1 A Comparative Study of the Binary Logistic Regression (BLR) and Artificial Neural Network (ANN) Models for GIS-Based Spatial Predicting Landslides at a Regional Scale**, p. 139–151. 01 2018.

[150] HAN, E.-H.; KARYPIS, G.. **Centroid-based document classification: Analysis and experimental results**. Lecture Notes in Computer Science, 1910:424–431, 01 2000.

[151] MANEVITZ, L. M.; YOUSEF, M.. **One-class svms for document classification**. J. Mach. Learn. Res., 2:139–154, Mar. 2002.

[152] XU, B.; GUO, X.; YE, Y. ; CHENG, J.. **An improved random forest classifier for text categorization**. Journal of Computers, 7, 12 2012.

[153] LI, L.; WEINBERG, C. R.; DARDEN, T. A. ; PEDERSEN, L. G.. **Gene selection for sample classification based on gene expression data: study of sensitivity to choice of parameters of the ga/knn method**. Bioinformatics, 17 12:1131–42, 2001.

[154] LECUN, Y.; BENGIO, Y. ; HINTON, G.. **Deep learning**. Nature, 521:436–44, 05 2015.

[155] ZHANG, X.; ZHAO, J. ; LECUN, Y.. **Character-level convolutional networks for text classification**, 2015.

[156] JOHNSON, R.; ZHANG, T.. **Supervised and semi-supervised text categorization using lstm for region embeddings**, 2016.

[157] ZHOU, P.; QI, Z.; ZHENG, S.; XU, J.; BAO, H. ; XU, B.. **Text classification improved by integrating bidirectional LSTM with two-dimensional max pooling**. In: PROCEEDINGS OF COLING 2016, THE 26TH INTERNATIONAL CONFERENCE ON COMPUTATIONAL LINGUISTICS: TECHNICAL PAPERS, p. 3485–3495, Osaka, Japan, Dec. 2016. The COLING 2016 Organizing Committee.

[158] JOHNSON, R.; ZHANG, T.. **Deep pyramid convolutional neural networks for text categorization**. In: PROCEEDINGS OF THE 55TH ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS (VOLUME 1: LONG PAPERS), p. 562–570, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[159] CONNEAU, A.; SCHWENK, H.; BARRAULT, L. ; LECUN, Y.. **Very deep convolutional networks for text classification**. Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers, 2017.

[160] CER, D.; YANG, Y.; YI KONG, S.; HUA, N.; LIMTIACO, N.; JOHN, R. S.; CONSTANT, N.; GUAJARDO-CESPEDES, M.; YUAN, S.; TAR, C.; SUNG, Y.-H.; STROPE, B. ; KURZWEIL, R.. **Universal sentence encoder**, 2018.

[161] VANDERLOOY, S.; HÜLLERMEIER, E.. **A critical analysis of variants of the auc**. Machine Learning, 72(3):247–262, Sep 2008.

[162] SOKOLOVA, M.; LAPALME, G.. **A systematic analysis of performance measures for classification tasks**. Information Processing & Management, 45(4):427 – 437, 2009.

[163] ALI, R.; LEE, S. ; CHUNG, T. C.. **Accurate multi-criteria decision making methodology for recommending machine learning algorithm**. Expert Systems with Applications, 71:257 – 278, 2017.

[164] ANANDARAJAN, M.; HILL, C. ; NOLAN, T.. **Practical Text Analytics: Maximizing the Value of Text Data**. Advances in Analytics and Data Science. Springer International Publishing, 2018.

[165] AGGARWAL, C. C.. **Machine Learning for Text**. Springer Publishing Company, Incorporated, 1st edition, 2018.

[166] PORTER, M. F.. **Readings in information retrieval**. chapter An Algorithm for Suffix Stripping, p. 313–316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[167] ORENGO, V. M.; HUYCK, C. R.. **A stemming algorithm for the portuguese language**. Proceedings Eighth Symposium on String Processing and Information Retrieval, p. 186–193, 2001.

[168] MIKOLOV, T.; CHEN, K.; CORRADO, G. ; DEAN, J.. **Efficient estimation of word representations in vector space**, 2013.

[169] LIU, B.. **Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data**. Data-centric systems and applications. Springer, 2007.

[170] MARON, M. E.. **Automatic indexing: An experimental inquiry**. J. ACM, 8(3):404–417, July 1961.

[171] SCIKIT-LEARN. **Naive bayes**. Naive Bayes, 2019. Acesso em: Julho de 2019.

[172] PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M. ; DUCHESNAY, E.. **Scikit-learn: Machine learning in Python**. Journal of Machine Learning Research, 12:2825–2830, 2011.

[173] CRAMER, J.. **Logit Models from Economics and Other Fields**. Cambridge University Press, 2003.

[174] BISHOP, C. M.. **Pattern Recognition and Machine Learning (Information Science and Statistics)**. Springer-Verlag, Berlin, Heidelberg, 2006.