UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

Ygor Henrique de Paula Barros Baêta

Recuperação de Erros em Derivação de Gramáticas Livres de Contexto Aplicado à Correção de Erros Ortográficos e Gramaticais

São João Del Rei

2021

UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

Ygor Henrique de Paula Barros Baêta

Recuperação de Erros em Derivação de Gramáticas Livres de Contexto Aplicado à Correção de Erros Ortográficos e Gramaticais

Dissertação apresentada como requisito ao título de mestre no Programa de Pós Graduação em Ciência da Computação da UFSJ.

Orientador: Carolina Ribeiro Xavier

Universidade Federal de São João del-Rei — UFSJ

Bacharelado em Ciência da Computação

São João Del Rei

2021

Ficha catalográfica elaborada pela Divisão de Biblioteca (DIBIB)
e Núcleo de Tecnologia da Informação (NTINF) da UFSJ,
com os dados fornecidos pelo(a) autor(a)

B142r

Baêta, Ygor Henrique de Paula.
Recuperação de Erros em Derivação de Gramáticas Livres de Contexto Aplicado à Correção de Erros Ortográficos e Gramaticais / Ygor Henrique de Paula Baêta ; orientadora Carolina Ribeiro Xavier; coorientador Alexandre Bittencourt Pigozzo. -- São João del-Rei, 2020.
52 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciência da Computação) -- Universidade Federal de São João del-Rei, 2020.

1. Processamento de Linguagens Naturais. 2. Marcação de Função Gramatical. 3. Derivação de Gramáticas Livres de Contexto. 4. Correção de Erros. I. Xavier, Carolina Ribeiro , orient. II. Pigozzo, Alexandre Bittencourt, co-orient. III. Título.

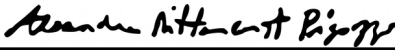
Ygor Henrique de Paula Barros Baêtas

Recuperação de Erros em Derivação de Gramáticas Livres de Contexto Aplicado à Correção de Erros Ortográ cos e Gramaticais

Dissertação apresentada para a Banca Examinadora da Universidade Federal de São João del Rei, como exigência parcial ao título de Mestre em Ciência da Computação.

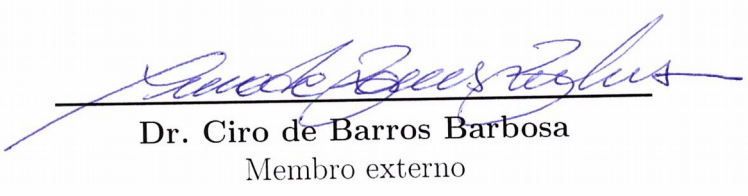
Trabalho aprovado. São João Del Rei, 10 de Dezembro de 2020:

Dra. Carolina Ribeiro Xavier
Orientadora



Dr. Alexandre Bittencourt Pigozzo
Coorientador

Dr. Vinícius Humberto Serapilha Durelli
Membro interno



Dr. Ciro de Barros Barbosa
Membro externo

São João Del Rei
2020

Dedico este trabalho à minha família,
aquela que nasci em e aquela que me tornei parte.

Agradecimentos

Não tenho como agradecer todos que fizeram parte da jornada que me trouxe até esse ponto, afinal o primeiro passo foi dado há quase 10 anos atrás, mas mesmo assim agradeço à todos aqueles que são parte desse caminho, de forma direta ou indireta.

Tudo começa com minha família. Agradeço meu irmão por sempre estar ao meu lado, me ajudando mesmo quando inicialmente ele não saiba o que fazer. Agradeço também meus pais pelo incentivo e apoio incondicional.

Agradeço então aos amigos, antigos e novos. Em especial agradeço minha namorada, que mesmo com seus próprios problemas ainda se preocupa em me ajudar a ter paz na correria de cada semestre. Agradeço aos meus amigos do ensino médio, que mesmo com a distância ainda sei que posso contar com a ajuda deles. Ao meu grupo de amigos mais próximos, agradeço por serem como uma segunda família longe de casa, sempre me ajudando em novas aventuras. Agradeço também a uma amiga em especial, que mesmo do outro lado do mundo, me apoiou em momentos difíceis.

Então agradeço aos professores. Primeiramente a minha orientadora, agradeço por me aturar, sei que não foi uma tarefa fácil. Ao meu orientador de TCC e Co-orientador, muito obrigado por acreditar em mim, mesmo quando minhas ideias parecem impossíveis. Agradeço mais um professor, que me ensinou a sempre tentar ser o melhor dos melhores.

Finalmente agradeço aqueles que acreditam na ciência, os professores, mestres, doutores e técnicos desta instituição, pois sem vocês não alcançaria este ponto.

"Os homens sempre se consideraram mais inteligentes que os golfinhos, porque haviam criado tanta coisa - a roda, Nova Iorque, as guerras, etc – enquanto os golfinhos só sabiam nadar e se divertir. Porém, os golfinhos, por sua vez, sempre se acharam muito mais inteligentes que os homens exatamente pelos mesmos motivos."

(Douglas Adams)

Resumo

Processamento de Linguagem Natural (*NLP* ou *Natural Language Processing* em inglês) é uma área interdisciplinar que estuda o processamento computacional de fala natural. É uma área em crescimento devido a popularidade de *chatbots* e Interfaces Vocais. Qualquer aplicação de NLP é construída sobre um fluxo básico de passos de tokenização, *stemming*, marcação de função gramatical (*Part-of-Speech tagging* ou *POS-tagging* em inglês), remoção de *Stop-Words*, análise de dependências, reconhecimento de entidade nomeadas e co-referenciamento, e embora essas etapas já estejam muito bem desenvolvidas para textos em inglês, o mesmo não pode ser dito para todos os idiomas. Este trabalho investiga o uso de técnicas desenvolvidas para correção de erros em compiladores para melhorar o resultado do *POS-tagging* em frases com problemas de classificação. Os resultados apresentados na pesquisa indicam que o processo desenvolvido pode ser utilizado de maneira eficaz em alguns contextos.

Palavras-chaves: Processamento de Linguagens Naturais, Marcação de Função Gramatical, Correção de Erros, Derivação de Gramáticas Livres de Contexto

Abstract

Natural Language Processing (NLP) is an interdisciplinary field that studies computational processing of natural speech. It is a growing area due to the popularity of Chat and Voice Interfaces. NLP applications are built upon a basic flow of steps: tokenization, stemming, part-of-speech (POS) tagging, stop-word removal, dependency analysis, named entity recognition and co-referencing, and while those steps already work really well on English texts, the same cannot be said for every language. This thesis investigates the use of techniques developed to correct errors in compilers to improve the result of POS-tagging in phrases with classification problems. The results presented in the research indicate that the developed process can be used effectively in some contexts.

Keywords: Natural Language Processing, POS tagging, Error Correction, Top-Down Parsing, Lexical Analysis

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AG	Algoritmo Genético
AST	Árvore de Sintaxe Abstrata
BLEU	*Bilingual Evaluation Understudy*
BNF	*Backus-Naur Form*
GLC	Gramática Livre de Contexto
NLP	*Natural Language Processing*, Processamento de Linguagens Naturais em inglês

Sumário

1 Introdução

Idiomas são um sistema complexo em constante evolução, é uma força motriz da tecnologia e estão sempre mudando para se adaptarem ao meio em que nos comunicamos. Por exemplo, Ogham evoluiu para ser escrito no canto de uma pedra (MacNeill, 1908). A imagem de uma língua em evolução normalmente é associada aos tempos antigos com proto-línguas e a criação de alfabetos como Hangul (KIM-RENAUD, 1997), mas idiomas modernos estão evoluindo e mudando por causa de políticas, uso popular, e outras razões. Um exemplo dessa evolução é o acordo ortográfico de 2008 da língua portuguesa, pensado para reduzir a diferença entre suas variantes (ZúQUETE, 2008).

Desde a popularização da Internet, idiomas aceleraram sua evolução, com novos termos sendo criados, aumento do uso de abreviações para conformar com limites de caracteres e o crescente catálogo de emojis, contando atualmente com cerca de 3000 símbolos (HERRING, 2008). A influência desse "Internetês" e seus neologismos e pictogramas é indiscutível, considerando, por exemplo que em 2015, a palavra do ano do dicionário Oxford foi 😂 (*Face With Tears of Joy*) (STEINMETZ, 2015). Essa evolução trouxe um interesse em campos de estudo de linguagens e suas interações como o Processamento de Linguagens Naturais.

O Processamento de Linguagem Natural (NLP ou *Natural Language Processing* em inglês) é um campo interdisciplinar de Computação e Linguística que estuda técnicas para representar a comunicação natural dos seres humanos de maneira computável (BATES, 1995). Nas últimas décadas, suas aplicações tem aumentado em número e importância (CAMBRIA; WHITE, 2014). Esse aumento pode ser explicado pelo aumento da interação entre humanos e inteligências artificiais, a adoção de *chatbots* ou o uso mais constante de interfaces por comando de voz e a popularização de sistemas de uso geral por usuários leigos (ZHANG, 2018).

Aplicações dessa área são diversas, como, análise de sentimentos (ROCHA et al., 2015), ferramentas capazes de entender linguagens naturais e realizar comandos computacionais (SILVA; LIMA, 2013), tradução computadorizada (GACHOT; LANGE; YANG, 1998), aprendizado de conceitos baseado em corpus (RADFORD et al., 2019), entre outras.

De acordo com a abordagem clássica de NLP o processo pode ser subdividido em passos similiares a como nós humanos compreendemos textos. A ideia básica, como ilustrado na Figura 1, é compreender a sintaxe, estrutura e corretude do texto; pragmática, contexto e coesão; e finalmente a semântica, sentido. Computacionalmente esses passos

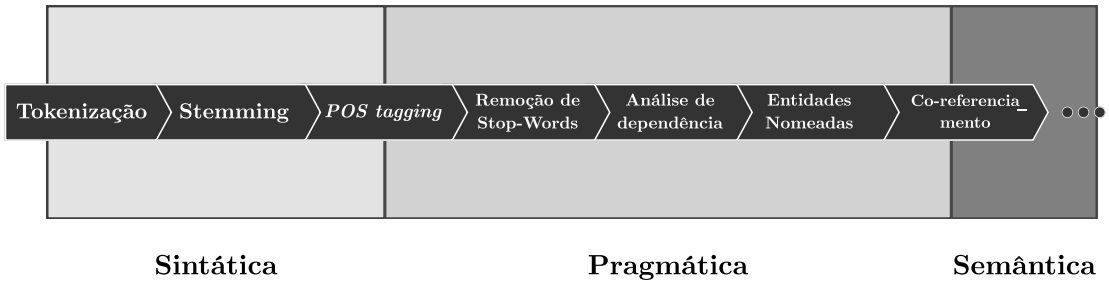


Figura 1 – Estrutura básica dos passos de NLP

são uma combinação dos processos de tokenização, *stemming*, marcação de classes gramaticais (*POS-tagging*), remoção de *Stop-Words*, análise de dependência, reconhecimento de entidade nomeadas e co-referenciamento (INDURKHYA; DAMERAU, 2010).

A abordagem clássica funciona muito bem na análise de estruturas formais de muitos idiomas, porém, os problemas para este tipo de processamento surgem nos dialetos informais, como o próprio Internetês, onde as regras claras do idioma são substituídas por versões mais relaxadas, fazendo com que a maioria das aplicações sofram quando a máquina deixa de compreender o que lhe foi pedido (BRN.AI, 2019).

1.1 NLP Globalmente

Como é um idioma global e a maioria dos esforços de pesquisa são voltados para o Inglês, NLP no idioma é um problema quase resolvido, com algoritmos com até 97% de acurácia (MANNING, 2011). Sendo o principal idioma da Internet existem diversos corpus de texto anotados e *frameworks* bem testados para os passos básicos de NLP (LORIA, 2018; BIRD; KLEIN; LOPER, 2009).

Esta acurácia cai drasticamente quando lidamos com textos em outros idiomas, principalmente aqueles que divergem drasticamente do Inglês. Essa queda é explicada pela falta de ferramentas que lidem com suas estruturas, alfabetos ou presença de características como aglutinação ou inflexão em excesso, como o caso do Croata (LJUBEšIć; DANIJELA, 2013) Árabe (FARGHALY; SHAALAN, 2009) ou Alemão (BAEZA-YATES, 2004)

Em português, por exemplo, os desafios principais têm relação com sua estrutura complexa, característica de línguas Latinas (CURY et al., 2002) e aos seu grande número de dialetos e variações de escrita, que podem confundir alguns algoritmos e dificultar os processos de *stemming* e marcação de funções gramaticais (GöRSKI; COELHO, 2009).

1.1.1 Português e Suas Complexidades

O Português é o idioma falado em Portugal, no Brasil e em algumas partes da África. É da família romance ocidental que evoluiu do latim (CPLP, 2019) e usa o alfabeto latino padrão com a adição de **Ç**. É um idioma com uma gramática complexa (BECHARA, 2015) e muitas variantes regionais.

As principais diferenças ortográficas e gramaticais são entre as variantes brasileira e europeia. Apesar do recente acordo ortográfico (ZúQUETE, 2008), essas variantes contam com muitas diferenças, principalmente quando utilizadas de maneira informal, causando confusão até entre os falantes. O português Brasileiro muitas vezes é considerado um idioma próprio com suas particularidades (CARREIRO; DIAS, 2015), porém ainda possui estrutura e dihttps://www.overleaf.com/project/5fe35cec59085f82e05aa31ccionário quase idênticos ao português europeu.

A construção de palavras em português segue uma estrutura silábica pouco definida devido às diversas influências externas que o idioma absorveu ao longo do tempo (CORREIA; LEMOS, 2005), e o uso de acentos para demonstrar fonemas tem origem discutida entre linguistas (NETTO, 2007).

Do ponto de vista sintático, a linguagem possui uma estrutura hierárquica para a criação de frases (BECHARA, 2015). Uma frase é a base sintática da linguagem e pode ser uma oração, ou um período contendo muitas orações. A organização estrutural das palavras segue basicamente a ordem Sujeito → Verbo → Objeto (SVO).

1.1.2 NLP em Português

No cenário de Processamento de Linguagens Naturais para nosso idioma temos atualmente diversos estudos focados nos pequenos problemas específicos (NUNE, 2008), mas ainda é largamente baseado nos mesmos algoritmos de inglês com correções e adaptações.

Esforços coletivos estão sendo feitos para mitigar essa falta de ferramentas especializadas (MUNIZ; NUNES; LAPORTE, 2005; RODRIGUES; OLIVEIRA; GOMES, 2018; RANCHHOD et al., 2004), mas com a criação de algoritmos específicos também há o surgimento de problemas específicos ao idioma, como a identificação e a colocação de pronomes como o caso das próclises, ênclises e mesóclises, dificuldades com gírias, analogias, entre outros que dificultam a maioria das tarefas (SANTOS; BARREIRO, 2004; HARTMANN et al., 2017; BARREIRO; RANCHHOD, 2005).

1.2 Objetivos

Pensando na importância de aplicações utilizando linguagem natural e nos problemas existentes para seu processamento, este trabalho tem como objetivo investigar o uso de algoritmos já existentes para a correção de erros em derivações de Gramáticas Livres de Contexto (ou GLCs) aplicados à um dos passos do processo de NLP clássico.

Dada uma frase em português brasileiro com palavras ou símbolos desconhecidos, a proposta deste trabalho é identificá-los corretamente de acordo com sua parte da fala. Isso será feito aplicando um analisador *Top-Down* (de cima para baixo) conhecido como Packrat (FORD, 2002), modificado para correção de erros. Esse analisador realiza o processo de *POS tagging* à medida que faz o *parsing*.

O analisador foi construído para aceitar a derivação de qualquer GLC e configurado para a língua portuguesa através de uma gramática que também foi desenvolvida durante o trabalho. A gramática descreve formalmente a construção sintática de frases em português e está estruturada de acordo com as suas funções gramaticais.

1.2.1 O Meta-compilador

O analisador desenvolvido durante o trabalho é uma aplicação de um trabalho anterior (BAETA, 2018), que pode ser usado como um compilador de uso geral para fazer protótipos de linguagens de programação. Algumas funcionalidades foram adicionadas para suportar a geração de uma Árvore de Sintaxe Abstrata (AST) com nós funcionais e para impor regras e restrições semânticas.

O projeto desta ferramenta pode ser dividido em três etapas. Primeiro foi definida uma notação capaz de descrever a gramática e suas produções, terminais, não terminais, produção inicial, caracteres ignorados, além de diferentes opções para os compiladores. A segunda tarefa foi implementar o analisador em si que é capaz de receber como entrada um arquivo escrito na linguagem definida e além de validá-lo e gerar uma representação intermediária executável, também é capaz de corrigir pequenos erros de derivação. Finalmente, foi definido um modelo de nós para a AST que pode ser estendido para adicionar novas funcionalidades. Além disso, foram implementados alguns nós básicos de modo a tornar mais simples o desenvolvimento de novos nós.

Em especial, o objetivo da notação desenvolvida é a facilidade de leitura e escrita para desenvolvimento rápido de gramáticas, tornando o analisador uma ferramenta de prototipagem rápida. Para isso, foi escolhido como base de desenvolvimento a notação na forma de Backus-Naur (BACKUS, 1963) que serve como base para muitos documentos de descrição de linguagens.

2 Conceitos Básicos

O trabalho desenvolvido se apoia no conhecimento de áreas da computação e em alguns conhecimentos básicos do idioma Português e esse capítulo pretende apresentar os conceitos relacionados. Na Seção 2.1 é abordado o assunto das gramáticas usadas e suas notações, a Seção 2.2 apresenta os algoritmos de derivação aplicados, na Seção 2.3 são apresentados os conceitos de processamento natural de linguagens clássico e estocástico, base para este trabalho, e finalmente na Seção 2.4 é feita uma revisão do conhecimento necessário da língua portuguesa.

2.1 Gramáticas e Notações

Uma gramática é um conjunto de regras que define a construção de uma linguagem e tem como base áreas da matemática relacionadas às linguagens formais e à computação teórica (MENEZES, 1998). As regras de uma gramática podem ser entendidas como um conjunto de diretrizes para reescrever sucessivamente um símbolo, ou conjunto de símbolos.

O estudo de gramáticas levou a vários formalismos e classificações, sendo a principal delas proposta por Noam Chomsky em 1956 (CHOMSKY, 1956), na qual ele organiza as linguagens em quatro níveis hierárquicos, esta hierquia ficou conhecida como hierarquia de Chomsky. Na Figura 2, observa-se os níveis definidos por Noam Chomsky. Do nível mais interno para o mais externo, aumenta-se a capacidade de expressão das gramáticas. Cada nível também possui formalismos com os quais ela pode ser descrita, validada e gerada (MENEZES, 1998).

2.1.1 Expressões Regulares

Expressões regulares (*Regex* ou *Regular Expression* em inglês), são um tipo de formalismo matemático usado em estudos de teoria de linguagens. Expressões regulares são normalmente empregadas na busca de sub-cadeias de caracteres em um texto. O conceito surgiu nos anos 50 quando Kleene formalizou matematicamente a descrição de uma linguagem regular (THE. . . , 2009).

Expressões regulares são usadas em diversas aplicações de busca que definem diferentes sintaxes e padrões, que, apesar de similares, possuem diferenças sutis (FRIEDL, 2002). Em sua grande parte essas sintaxes acabam por definir funcionalidades que tornam as linguagens dessas expressões em linguagens não regulares, apesar do nome.

Figura 2 – Hierarquia de Chomsky

Uma expressão regular é composta por uma combinação de caracteres simples e caracteres especiais. Os caracteres especiais são usados para denotar repetições, classes de caractere ou sub-expressões. Um exemplo dessas expressões é um *Regex* que identifica a maioria dos endereços de e-mail válidos (FRIEDL, 2002).

$$/[0-9A-Z][0-9A-Z.]\{0,63\}@[A-Z][0-9A-Z]^*(.[A-Z]^+)^+/i$$

Neste exemplo podemos observar a maioria dos elementos de um *Regex*, primeiro vemos que a expressão é dividida na própria expressão delimitada por / e em seguida suas opções, neste caso i pede ao interpretador do *Regex* que ignore a diferença entre minusculas e maiúsculas.

Dentro da expressão observamos os especificadores de quantidade $^+$, * e $\{m, n\}$ que indicam respectivamente que a expressão anterior ao especificador deve ser repetida uma ou mais, zero ou mais, ou entre m e n vezes respectivamente, essas expressões podem ser caracteres, classes ou sub-expressões. Caracteres simples como **@** representam eles mesmos. Classes são identificadas por estarem entre colchetes por exemplo uma classe $[IJkl]$ que representa qualquer um dos caracteres entre os colchetes, **I** maiúsculo, **J** maiúsculo, **k** minusculo ou **l** minusculo. Classes também podem conter séries, indicadas por $-$ como na classe do exemplo $[0-9A-Z.]$ que pode representar qualquer letra, número ou o caractere **.** (ponto). Sub-expressões seguem as mesmas regras de uma expressão e são representadas

por parenteses como em $(.[A-Z]^{+})$.

2.1.2 Gramáticas Livres de Contexto

Entre as classes definidas por Chomsky, a classe de Linguagens Livres de Contexto possui diversas aplicações no contexto da computação, e também engloba a construção de várias das linguagens naturais. Um dos formalismos desse tipo de linguagem são as Gramáticas Livres de Contexto (GLC) (CHOMSKY, 1956). A aplicação sucessiva das produções partindo do símbolo inicial gera qualquer palavra pertencente à linguagem gerada pela GLC e por isso é chamada formalização construtiva (MENEZES, 1998).

Uma GLC é geralmente representada por uma 4-tupla $G = (V, \Sigma, P, S_0)$ sendo V um conjunto de não-terminais, Σ um conjunto de terminais, P é um conjunto de produções, ou funções do tipo $p : V \rightarrow (\Sigma \cup V)^{\star}$, e, $S_0 \in V$ é o simbolo inicial da gramática. O conjunto de produções é normalmente representado por um conjunto de regras na forma $A \rightarrow \beta$ mas outras notações foram desenvolvidas por conveniência ou legibilidade (MENEZES, 1998).

2.1.3 Notação BNF

A descrição de uma linguagem de programação deve ser feita de maneira clara e precisa. Para este fim, em 1958, John Backus propôs uma metalinguagem de fórmulas metalinguísticas para descrever o ALGOL, sua nova linguagem de programação (BACKUS, 1963). Desde então, a forma de Backus-Naur (BNF) vem sendo a forma majoritária que é usada para descrever linguagens de programação.

A BNF é uma notação de produções de uma GLC, composta por diversas regras escritas na forma: produção ::= expressão onde produção representa um não-terminal da gramática e expressão é composta por terminais e não terminais. Cada uma dessas regras indica que a expressão à direita é derivada a partir do não terminal à esquerda.

A expressão de uma BNF usa nomes entre ⟨ e ⟩ para identificar um não terminal e cadeias de caracteres entre aspas duplas para representar cadeias literais de terminais.

A notação BNF também usa o caractere | para indicar expressões alternativas. Podemos observar essas regras, por exemplo, na gramática descrita através da BNF que define operações matemáticas incluindo as quatro operações básicas (+, -, *, /) e o uso de parênteses:

$G(V, \Sigma, P, \text{expressão})$ onde:
$V = \{\text{expressão}, \text{fator}, \text{parcela}, \text{número}\}$
$\Sigma = \{+, -, \div, \times, (,), 0, 1, ..., 9\}$
P é representado por:

$$
\begin{aligned}
\langle\text{expressão}\rangle &:= \langle\text{parcela}\rangle\text{“}+\text{”}\langle\text{expressão}\rangle \\
&\mid \langle\text{parcela}\rangle\text{“}-\text{”}\langle\text{expressão}\rangle \\
&\mid \langle\text{parcela}\rangle \\
\\
\langle\text{parcela}\rangle &:= \langle\text{fator}\rangle\text{“}\times\text{”}\langle\text{parcela}\rangle \\
&\mid \langle\text{fator}\rangle\text{“}\div\text{”}\langle\text{parcela}\rangle \\
&\mid \langle\text{fator}\rangle \\
\\
\langle\text{fator}\rangle &:= \text{“(”}\langle\text{expressão}\rangle\text{“)”} \\
&\mid \langle\text{numero}\rangle \\
\langle\text{número}\rangle &:= \langle\text{número}\rangle\langle\text{número}\rangle \\
&\mid \text{‘0’} \mid \text{‘1’} \mid \text{‘2’} \mid \text{‘3’} \mid \text{‘4’} \mid \text{‘5’} \mid \text{‘6’} \mid \text{‘7’} \mid \text{‘8’} \mid \text{‘9’}
\end{aligned}
$$

Nesse exemplo podemos interpretar a regra $\langle\text{fator}\rangle :=$ “(”$\langle$expressão$\rangle$“)” $\mid \langle$numero$\rangle$ como: “um fator é composto de uma expressão entre parênteses ou por um número"um número que por sua vez é definido por outra regra. Uma extensão comum e útil da BNF é o uso de expressões regulares no lugar de terminais, com essa extensão a regra que representa os números poderia ser reescrita como $\langle\text{número}\rangle := /[0-9]+/$

2.2 Compiladores e Análises

Um compilador é um software que traduz um programa escrito em uma linguagem, chamada de fonte, à outra linguagem, conhecida como destino. Para completar esta tarefa, o compilador é dividido em três etapas (SETHI; ULLMAN; LAM, 2008; AHO; ULLMAN, 1999).

Primeiramente, o compilador executa a etapa de análise onde realiza três tipos

de análise do código-fonte: análise léxica, análise sintática e análise semântica. Ao fim da etapa de análise, é gerada uma representação intermediária do arquivo fonte que pode então ser passada ao próximo passo. O segundo passo é conhecido como síntese e é responsável por gerar o código na linguagem de destino. O passo final é um passo de otimização onde o compilador modifica o código gerado com objetivo de torná-lo melhor em algum aspecto, normalmente tamanho e eficiência (GRUNE et al., 2016).

O compilador desenvolvido trabalha principalmente no estágio de análise. A análise pode ser divida em três processos menores: Análise Léxica, Análise Sintática e Análise Semântica.

2.2.1 Análise Léxica

A primeira das análises de um compilador, a análise léxica, tem como objetivo traduzir as cadeias de caracteres que compõe o código-fonte para uma sequência de *tokens*, onde cada *token* representa uma unidade com significado, por exemplo, identificadores ou operadores. Essa análise é feita para que os próximos passos do compilador possam ignorar o processamento de cadeias de caracteres, assim simplificando o processo.

O passo de análise léxica é estudado e aplicado em diversas áreas incluindo o processamento de linguagem natural, onde é usado no processo de tokenização como apresentado na Seção 2.3. Por este motivo, é um processo com algoritmos conhecidos e eficientes.

Normalmente, em um compilador, essa análise é implementada usando um autômato finito determinístico ou através da interpretação de expressões regulares que representam as definições léxicas da linguagem.

2.2.2 Análise Sintática

Após a análise léxica, o compilador realiza a análise sintática, com objetivo de validar as regras de construção da linguagem. Regras descritas utilizando alguma notação formal para descrição de gramáticas como, por exemplo, a notação BNF discutida na Seção 2.1.3.

O analisador recebe a cadeia de *tokens* gerada anteriormente e verifica se a cadeia pode ser gerada pela gramática da linguagem fonte. Nesta etapa, é esperado que sejam apontados erros de sintaxe, ou seja, erros na estrutura ou gramática utilizadas no código-fonte.

Para isso é gerada uma árvore de derivação, que representa os passos ou a sequência de derivações necessárias para se chegar à cadeia de *tokens* específica da entrada a partir da produção base da gramática, que é a produção do símbolo inicial da gramática.

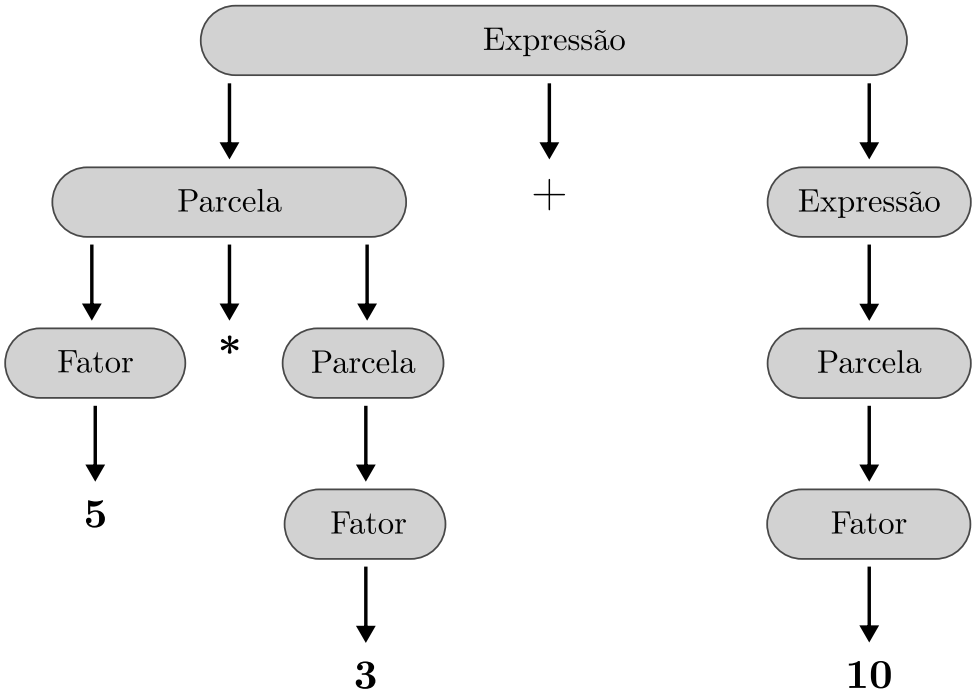


Figura 3 – Árvore de derivação da expressão 5 * 3 + 10

A Figura 3 mostra um exemplo de árvore de derivação das expressão 5 * 3 + 10 utilizando a gramática apresentada na Seção 2.1, note que a ordem das operações é mantida devido à sua inclusão na construção da gramática.

A árvore de derivação pode ser gerada utilizando métodos descendentes ou ascendentes. Métodos ascendentes geram a árvore de derivação aglutinando os terminais à medida que estes aparecem na cadeia de *tokens*, usando uma pilha ou alguma outra estrutura similar. Métodos descendentes começam a análise sintática pelo símbolo inicial da gramática e aplicam uma sequência de regras da gramática para tentar derivar a cadeia de entrada. Existem dois métodos principais de análise descendente: o método baseado em tabela e o método de descida recursiva.

O método baseado em tabela constrói uma tabela de análise com base nos conjuntos *First* (conjuntos de iniciadores) e *Follow* (conjuntos de seguidores) da gramática e utiliza uma pilha para controlar o processo de derivação. Já o método de descida recursiva define, para cada não-terminal, uma função que implementa as regras sintáticas do não-terminal. As funções são chamadas recursivamente, daí o nome do método, a partir da função que representa o símbolo inicial da gramática.

Existe uma variação do método de descida recursiva conhecida como método de *Packrat*, que otimiza a principal deficiência do método original, os recálculos excessivos devido ao *backtracking* . O método *Packrat* grava na memória derivações parciais sendo capaz de se recuperar de uma falha com maior velocidade no processo de derivação, em

troca do aumento no uso de memória durante a execução.

2.2.3 Análise Semântica

A última análise, conhecida como análise semântica, tem como objetivo atribuir significado às construções e expressões da linguagem. Essa fase é responsável por gerar o código intermediário que será passado ao passo de síntese. Normalmente é realizada juntamente com a análise sintática, e ajuda a fazer verificações dependentes de contexto como, por exemplo, tipagem e *null values*, valores vazios passados em funções que não os esperam.

Em geral, o resultado final da análise semântica é um código intermediário que normalmente está representado como uma árvore de sintaxe abstrata (*abstract syntax tree* ou AST em inglês) ou um código de três endereços.

Uma árvore de sintaxe abstrata é uma representação em árvore das construções da linguagem, ou seja, é uma representação onde cada comando ou expressão de uma linguagem é um nó da árvore, e a árvore completa representa todo o código fonte.

A principal vantagem da AST em relação à árvore de derivação é que a AST representa de forma mais concisa todas as construções presentes no código fonte de entrada. A AST não representa os passos da derivação e também não representa caracteres de pontuação e de estruturação do código, isso permite que a estrutura seja simplificada, ocupando menos espaço e sendo mais eficiente quando percorrida. A representação da AST também costuma ser mais semântica, gravando nos próprios nós qual tipo de operação ou comando este representa.

Um exemplo dessas vantagens é a comparação entre a árvore de derivação e a AST de uma expressão. Onde na AST cada expressão é representada como uma subárvore que tem o operador da expressão como nó pai e os nós filhos são os operandos do lado esquerdo e do lado direito, e estes operandos do lado esquerdo e do lado direito podem ser outras expressões como pode ser visto na Figura 4, que representa a AST da mesma derivação da Figura 3.

2.2.4 Recuperação de Erros

Durante o processo de compilação, um *parser* possui duas alternativas ao encontrar erros, ou falhar em sua análise ou tentar corrigir o erro e prosseguir com a sua análise (AHO; ULLMAN, 1999). Em sua grande maioria, os compiladores são programados para realizar uma combinação de ambos, falhar com uma mensagem descritiva ao programador e ignorar qualquer comando até que este se encontre em um estado onde é possível prosseguir, isto permite que outros erros sejam encontrados em partes que estão à frente

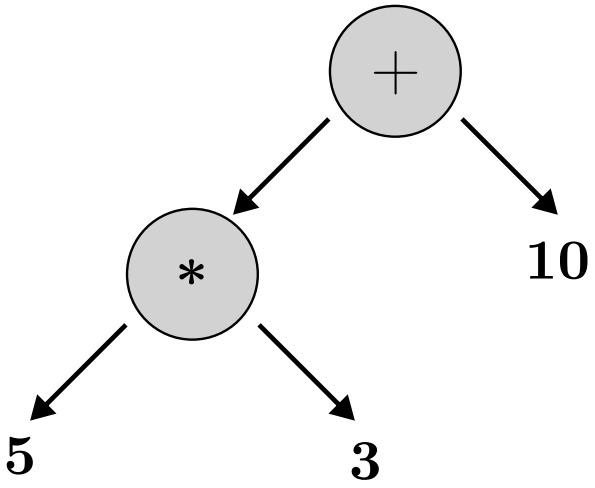


Figura 4 – AST da expressão 5 * 3 + 10

do erro atual (GRUNE et al., 2016).

A recuperação de erros de um compilador é um processo computacionalmente custoso que deve ser implementado com bastante cuidado, de maneira a evitar laços infinitos durante a análise, porém pode ser realizada para alguns casos específicos. Um dos casos mais comuns é a adição de terminadores como **;** ou quebra de linha no ponto atual da derivação, essa correção permite que o compilador possa prosseguir com o processamento da próxima expressão.

2.3 Processamento de Linguagens Naturais

Como apresentado no Capitulo 1, o NLP é um campo interdisciplinar que em sua forma clássica pode ser dividida em passos, que são cumulativos e usam o resultado do passo anterior como entrada e sua saída é usada como entrada pelo passo seguinte. A Figura 5 representa o fluxo geral do processo (INDURKHYA; DAMERAU, 2010).

A tokenização é o processo de dividir a entrada de texto em elementos contidos conhecidos como *tokens*. Esses elementos normalmente são as palavras da frase. A tokenização é um processo muito semelhante à análise léxica descrita na Seção 2.2.1. Em seguida o segundo passo, *stemming*, é responsável por normalizar as palavras reduzindo-as às suas raízes morfológicas, seus radicais, em geral descartando flexões e conjugações, a não ser que estas sejam explicitamente necessárias. O *POS Tagging* tem como objetivo a marcação e classificação das palavras em suas classes gramáticais, como essa etapa é o foco deste trabalho será melhor discutida na Seção 2.3.1.

Os próximos passos iniciam o processamento pragmático com a remoção de *stop words*, que é usada para reduzir a carga das próximas etapas, eliminando palavras repetitivas mas com pouca importância ao contexto. E finalmente a análise de dependências, o

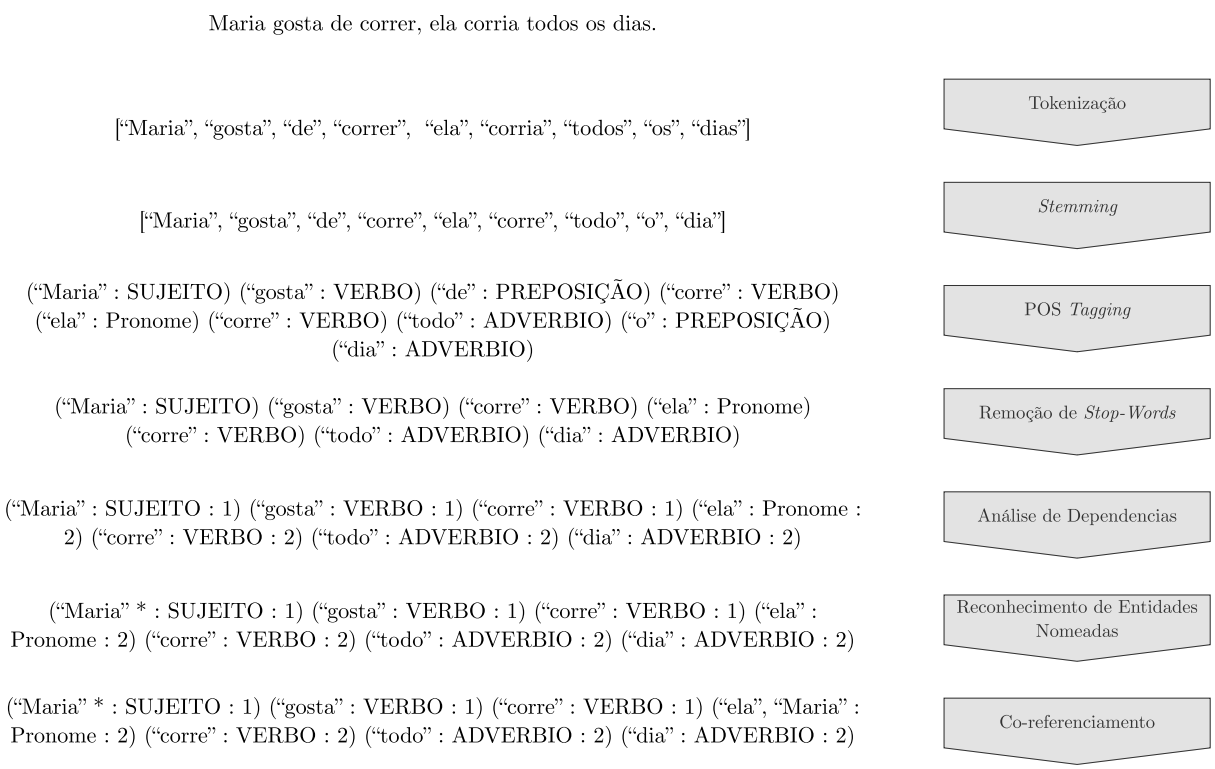


Figura 5 – Fluxo Geral de NLP

reconhecimento de entidades nomeadas e o co-referenciamento lidam com a relação entre duas ou mais sentenças, suas relações e as relações com pessoas, lugares ou conceitos do mundo real (KRULEE, 1991).

2.3.1 POS Tagging

O processo de POS Tagging consiste na categorização das palavras com base em sua função gramatical. Este processo é uma parte fundamental da análise da linguagem natural computacionalmente, pois aproxima a compreensão da máquina à como humanos classificam e dividem cognitivamente as frases (KRULEE, 1991; INDURKHYA; DAMERAU, 2010).

Em português, as classes de palavras são: substantivo, verbo, adjetivo, pronome, artigo, numeral, preposição, conjunção, interjeição e advérbio (BECHARA, 2015). Essas classes serão discutidas em maior detalhe na Seção 2.4. Um *tagger* também pode adicionar informações extras a cada classe, como conjugação de verbos ou sua inflexão, caso estes não tenham sido descartados nos passos anteriores.

Atualmente, existem várias técnicas de marcação utilizando métodos de diferentes áreas da computação (ACLWEB, 2019), sendo as principais baseadas em redes Bayesianas

através do modelo de Markov, redes neurais, máquinas de vetores de suporte e gramáticas definidas por especialistas. Tais técnicas surgiram da união do aprendizado de máquina com o NLP.

Cada uma das técnicas tem suas vantagens e desvantagens (ZHANG, 2018), por exemplo, as baseadas em gramática são simples de implementar e eficientes, mas caras para criar inicialmente, devido à necessidade de um especialista para criação da gramática. Já as baseadas em modelos de Markov precisam ser treinadas em uma base pré-anotada e portanto são sujeitas a *overfitting*, mas podem ser reusadas para diversos idiomas estruturalmente próximos.

2.3.2 Aplicações de NLP

O processo de NLP possui diversas aplicações, classificação de textos, sumarização, texto preditivos, entre outras. Para verificação da validade do trabalho serão utilizadas duas aplicações simples, análise de sentimentos e tradução de máquina.

A análise de sentimentos é uma aplicação de NLP com objetivo de identificar e quantificar um texto como positivo, negativo ou neutro (FELDMAN, 2013). Análise de sentimentos possui diversas aplicações (ALAMOODI et al., 2020; KAUFFMANN et al., 2020) e recentemente se baseia principalmente em modelos de aprendizado de máquinas.

Já a Tradução de maquina é o processo de realizar uma tradução de um texto para outro idioma através de algoritmos. O método mais simples de tradução envolve o uso de um dicionário para a substituição das palavras, e, apesar dessa técnica ser eficiente acaba apresentando acurácia muito baixa.

2.4 A Língua Portuguesa

Usada em 10 países por cerca de 270 milhões de pessoas na Europa, Africa, América do sul e Ásia, é o idioma oficial de Portugal, Brasil, Cabo Verde, Guiné-Bissau, Moçambique, Angola e São Tomé e Príncipe, também é encontrada em Macau, Timor Leste e Guiné Equatorial. Português é o 5ª idioma mais falado do mundo, fica em 3ª lugar no ocidente e em 1º no hemisfério sul (CPLP, 2019).

O português possui muitas variantes faladas, e somente no Brasil conta com 16 dialetos reconhecidos (GöRSKI; COELHO, 2009; CARREIRO; DIAS, 2015). Já a forma escrita possui apenas duas variantes formais, a escrita fora do Brasil e a escrita no Brasil, essas diferenças foram reduzidas no mais recente acordo ortográfico e existe um esforço dos países lusófonos de unificar o idioma formal (ZúQUETE, 2008).

A língua portuguesa é derivada do Latim e compartilha várias similaridades com

idiomas de mesma origem, é um idioma sintético e flexivo com dois gêneros e números gramaticais aplicados nos substantivos, adjetivos, artigos e pronomes (BECHARA, 2015).

Em português, uma sentença é uma ou mais palavras com sentido completo (BECHARA, 2015). Essas sentenças são subdividas em orações estruturadas ao redor de um verbo, uma oração pode ou não possuir sentido individualmente. A estruturação das frases segue em geral a ordem Sujeito-Verbo-Objeto ou SVO, e a construção das frases pode ser feita usando a ordem direta, mais comum, ou ordem indireta, usada em algumas obras mais rebuscadas.

As palavras do idioma são classificadas em 10 classes gramaticais (BECHARA, 2015). Numerais indicam posição ou quantidade, podendo ser cardinais, ordinais, multiplicativos fracionários e coletivos. Interjeições exprimem emoções ou sentimentos, já conjunções são usadas para unir duas orações de mesmo valor gramatical.

Substantivos, são palavras que se referem à objetos, fenômenos, animais, pessoas e coisas em geral, substantivos geralmente são classificados em comum ou próprios. Adjetivos, Pronomes e Artigos são relacionados à substantivos, com Adjetivos caracterizando e especificando os substantivos, Pronomes usados para substitui-los, quando necessário, e artigos usados em precedência à eles.

Verbos indicam ações, estados ou fenômenos da natureza, verbos são conjugados em relação ao modo e tempo verbal, existem 11 principais formas de conjugação verbal divididas em três modos, indicativo, subjuntivo e imperativo. Verbos podem ser regulares ou irregulares e quando regulares são conjugados de acordo com sua terminação. Advérbios modificam e caracterizam os verbos especificando tempo, modo, intensidade entre outros, da mesma maneira que Adjetivos o fazem para Substantivos.

2.5 Algoritmos Genéticos

Algoritmos genéticos (ou AGs) são uma classe de algoritmos baseados no processo de seleção natural, baseada em três funções, seleção, cruzamento e mutação que imitam os organismos e sua evolução (WANG, 2003).

Um algoritmo genético pode ser adaptado para diversas problemas de otimização, e por isso vem sendo utilizado extensamente na literatura, sua única restrição é que para que um problema possa ser resolvido ele deve ser representado como um conjunto de valores. Essa representação normalmente é chamada de genótipo.

Os três passos de um algoritmo genético são repetidos até que se alcance uma uma condição de parada, normalmente baseada em uma função objetivo. A estrutura básica de um AG segue sempre o mesmo modelo com poucas alterações, esse algoritmo pode ser

observado no Algoritmo 1 (SHIFFMAN, 2012)

Algoritmo 1: Modelo de Algoritmo Genético

Entrada: População inicial: P_0, Condição de Parada: x

$P \leftarrow P_0$;
enquanto $\texttt{Max}(f_o(i) \forall i \in P) < x$ **faça**
    $P' \leftarrow \texttt{Seleção}(P, f_o)$;
    $P' \leftarrow \texttt{Cruzamento}(P')$;
    $P' \leftarrow \texttt{Mutação}(P')$;
    $P \leftarrow P'$;

3 Trabalhos Relacionados

A técnica de *POS tagging* utilizada é baseada na construção de uma gramática correta da língua portuguesa. Para essa construção foi utilizado um Algoritmo Genético baseado no artigo descrito na Seção 3.1. O artigo da Seção 3.2 foi utilizado como técnica para comparação da eficácia da marcação. Finalmente, na Seção 3.3, são listados alguns frameworks que foram utilizados durante a implementação do algoritmo.

3.1 Indução de Gramáticas

O trabalho *Sequential Structuring Element for CFG Induction Using Genetic Algorithm* de Choubey e Kharat (CHOUBEY; KHARAT, 2010; CHOUBEY; PANDEY; KHARAT, 2011) investiga um método de indução de Gramáticas através do uso de algoritmos genéticos. Esse trabalho explora técnicas para melhorar os resultados em relação à velocidade e à acurácia das GLCs encontradas. O trabalho alcança uma técnica com convergência aceitável para aplicação a partir de um mapeamento direto entre os cromossomos e os terminais da gramática e a utilização de uma função objetivo com parâmetros ponderados de maneira a dar prioridade a uma gramática concisa mas correta.

O funcionamento geral de um algoritmo genético é baseado na implementação correta do mapeamento entre o genoma e o objetivo e na escolha de uma boa função objetivo. O genoma escolhido é uma sequência de bits com tamanho fixo que é decodificada para uma *string* contínua de símbolos terminais e não terminais. Essa *string* de símbolos então é divida em regras onde são aplicados algoritmos padrão de gramáticas para remoção de regras inúteis, inalcançáveis e recursão.

A função objetivo escolhida foi $f_o = (W_p * N_p) - (W_n * N_n) + (W_r - N_r)$ onde $N_p, N_n e N_r$ são respectivamente o número de testes positivos, o número de testes negativos e o número de regras da gramática. W_r foi balanceado para o tamanho esperado da gramática.

O método investigado pelos autores foi essencial para o funcionamento final do trabalho desenvolvido nesta dissertação. A indução foi utilizada para a geração de uma GLC capaz de descrever o idioma a partir de uma versão inicial construída manualmente, como será discutido no Capítulo 4.

3.2 Outro algoritmo de *POS tagging*

O artigo *HunPos: an Open Source Trigram Tagger* por Hálcsy, Kornai e Oravecz descreve um marcador baseado na técnica que apresenta os melhores resultados para essa tarefa, a mesma técnica do *TnT* (HALáCSY; KORNAI; ORAVECZ, 2007). Os pesquisadores focaram na criação de uma versão código livre da ferramenta já que o *TnT* apesar de muito utilizado não distribui seu código fonte.

A técnica básica é a utilização de modelos probabilísticos com a frequência da ocorrência de tuplas de até três palavras. Esses modelos são utilizados como preditores para a próxima palavra a ser marcada em uma sequência.

A comparação com ferramentas disponíveis já existentes cria um patamar ao qual qualquer nova técnica pode tentar se posicionar. O artigo e ferramenta são bases para a comparação da eficácia do trabalho desenvolvido e apesar de utilizarem técnicas completamente diferentes apresentaram resultados similares, como visto na Seção 5.1.1 onde os trabalhos foram comparados em relação à sua acurácia.

3.3 Frameworks de NLP

Durante o desenvolvimento do trabalho foram utilizados dois frameworks existentes de Processamento de Linguagem Natural, *Node-natural* (UMBEL; ELLIS; MULL, 2011) e *NLTK* (BIRD; KLEIN; LOPER, 2009). A utilização de dois frameworks foi necessária uma vez que o trabalho foi desenvolvido em duas linguagens de programação *Node.js* e *Python*.

A biblioteca *NLTK* é um projeto de código livre que existe desde 2000 e foi utilizada para todo o pré-processamento. As ferramentas de tradução e análise de sentimento da biblioteca *NLTK* foram utilizadas como testes do método desenvolvido neste trabalho.

Já o *Node-natural* foi utilizado como tratamento da entrada para a ferramenta de correção que já havia sido desenvolvida anteriormente em *Node.js*. Todo processo incluindo os testes poderiam ter sido feitos utilizando apenas esta biblioteca mas a implementação utilizando *NLTK* acabou se tornando a implementação mais simples.

3.3.1 Ferramentas de Correção Comerciais

A evolução dos processos de NLP e dos corretores de texto trouxe consigo a criação de ferramentas comerciais de correção gramatical. Essas ferramentas trazem consigo funções avançadas como correção de estilo de escrita (GRAMARLY, 2020), sugestão de modificações para melhoria da qualidade do texto (SAPLING, 2020) e até geração de relatórios sobre a escrita (Pro Writting Aid, 2020).

4 Metodologia

O desenvolvimento do trabalho foi dividido em duas partes, sendo a primeira o interpretador e segunda a gramática, cada uma dessas partes funciona individualmente e colaboram para a obtenção da correção de erros. O corretor é baseado em algoritmos de derivação e sua construção é discutida na Seção 4.2, a gramática foi gerada a partir de um algoritmo genético como discutido na Seção 4.3. A correção foi aplicada como passo de dois outros algoritmos de NLP, análise de sentimentos e tradução de texto. Os testes permitiram a análise da eficácia de um processo de *POS tagging* mais correto.

Foram utilizados dois corpus para a realização dos testes, o primeiro da copa do mundo 2014, um conjunto de dados previamente disponível (MANSSOUR; SILVEIRA, 2015), e outro coletado a partir de hashtags referentes às eleições de 2018 (BANHESSE; NADAI, 2018). Dentre todas as correções realizadas durante o tagging, a identificação e tratamento de emojis foi realizada e analisada como foco principal. Esse foco de correção foi feito ao observar que eles são obstáculos para vários métodos tradicionais de NLP, porém todos os erros descritos na Seção 4.1 foram tratados.

Para verificar a eficacia dos algoritmos, foi feita uma comparação do resultado dos algoritmos sem o processo de correção e com o processo de correção. Essa correção foi feita com a substituição do emoji por uma palavra equivalente como descrito na Seção 4.4.

4.1 Definição de Um Erro de Derivação

Para entender a técnica e suas aplicações, primeiro precisamos entender o que é um erro no processo de derivação. Para esse trabalho um erro é qualquer palavra que foi marcada incorretamente. Erros podem ocorrer a partir de **três** processos diferentes baseados nos erros considerados por Levenshtein (Levenshtein, 1966). Todos esses erros foram representados a partir de modificações na gramática original $G(V, T, P, S)$.

Na **inserção** uma palavra foi adicionada à frase onde, de acordo com as regras formais, não deveria existir. A inserção foi representada pela adição de uma nova regra $Al \rightarrow Z\alpha$, para cada regra do tipo $A \rightarrow \alpha \mid \alpha \in (V \cup T)*$, α é uma cadeia de terminais e não terminais, e uma nova regra $Z \rightarrow a$ na gramática para cada terminal a. Um exemplo de erros deste tipo é a repetição acidental de palavras, por exemplo na frase "A casa é **é** azul"o segundo verbo **é** foi adicionado incorretamente.

O segundo tipo de erro é a **remoção**, que ocorre quando uma palavra foi omitida

causando um vácuo, sua representação nas regras de produção pode ser feita com a adição de uma regra $A \rightarrow \alpha$ para cada regra do tipo $A \rightarrow B\alpha, B \in T$. Alternativamente, podem ser adicionadas regras do tipo $A \rightarrow \epsilon$ para cada não-terminal A, o algoritmo implementado utiliza esta segunda técnica. Esse tipo de erro é o mais complexo de verificar na língua portuguesa uma vez que várias classes de palavras são opcionais na construção de uma oração.

Finalmente, a **substituição** ocorre quando uma palavra usada não faz parte do dicionário ou foi usada de uma maneira que não é formalmente aceita, por exemplo, como uma gíria. Para cada regra do tipo $A \rightarrow \alpha A\beta | \alpha, \beta \in (V \cup T) * A \in T$ onde α e β são uma cadeia de 0 ou mais terminais e não terminais e A são terminais é criada uma nova regra do tipo $A \rightarrow \alpha B\beta \forall B \in T$. Idealmente, a gramática se encontra em uma forma que garante que $|T|$ é o menor possível, porém este não é o caso e isso acarreta em um grande número de novas regras.

A partir dessas definições, foram desenvolvidas as ferramentas capazes de ler, interpretar e modificar uma GLC, capaz de apontar e corrigir esses tipos de erro.

4.2 Construção do Algoritmo

O Algoritmo de correção foi implementado como uma extensão de um meta-compilador, construído anteriormente como meu trabalho de conclusão da graduação, capaz de interpretar qualquer GLC. A ferramenta foi implementada em *JavaScript* e tem a capacidade de realizar os passos de derivação de uma gramática a partir de uma definição em formato similar à **BNF** utilizando uma versão modificada do algoritmo de *packrat*.

O fluxo geral da ferramenta descrito na Figura 6 pode ser dividido em três passos: 1) *Parsing* da gramática; 2) Construção do interpretador; e 3) Interpretação. O primeiro passo recebe como entrada um arquivo *.grmmr* contendo a definição de uma gramática escrita no formato desenvolvido e utiliza essas regras para, em conjunto com scripts que definem nós, realizar a construção de um interpretador no segundo passo. Por fim, o interpretador é executado para interpretar o programa.

Os nós são definidos em uma classe de *JavaScript* que pode ser estendida ou sobrescrita para permitir qualquer tipo de operação "nativa" à linguagem desenvolvida, assim como qualquer tipo de análise semântica que seja necessária. O algoritmo de construção também aplica otimizações na gramática original, removendo recursão à esquerda direta e indireta e construindo os conjuntos *First* e *Follow* de cada não terminal de modo a agilizar o processo final.

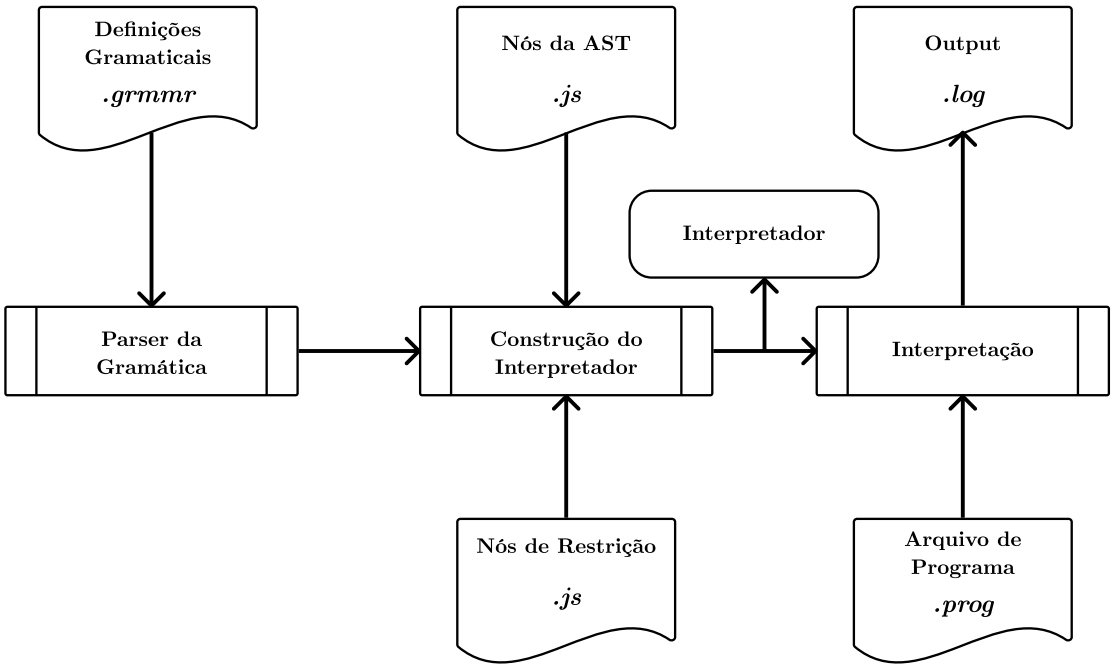
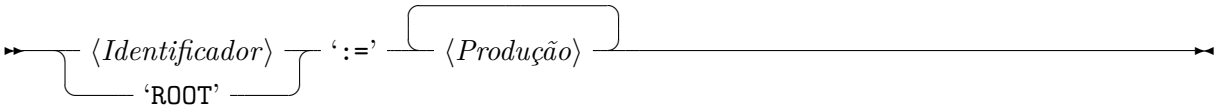


Figura 6 – Fluxograma do Funcionamento do Compilador Desenvolvido

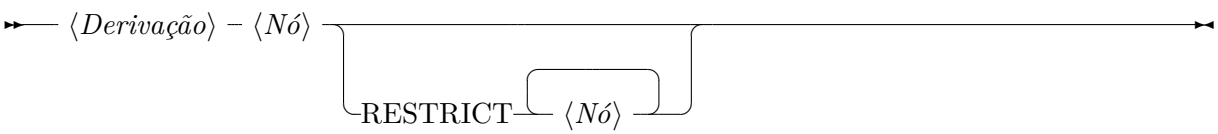
4.2.1 A Notação Desenvolvida

A notação de descrição da gramática segue um padrão parecido com a notação BNF. Cada linguagem é descrita por uma sequência de regras, uma por linha do arquivo, que indicam uma ou mais produções e, para cada uma, o nó da AST equivalente à essa produção. Cada regra pode ser definida como:

Regra

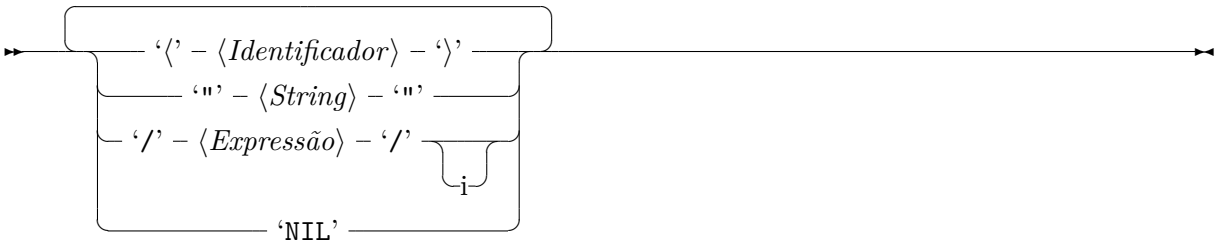


Produção



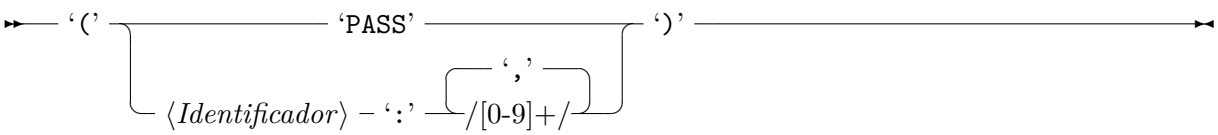
Os identificadores são compostos de caracteres entre A e Z minúsculos ou maiúsculos, números e o caractere *underline* (_). As cadeias de produção em si são definidas por uma sequência de terminais e não terminais, definidos pelas regras:

Derivação



Onde expressão é qualquer expressão regular válida e *string* representa qualquer cadeia de caracteres sem aspas. Os nós de equivalência são compostos do nome de um nó e a lista de parâmetros deste. A lista de parâmetros é uma lista de índices que indica quais produções são filhas deste nó. Finalmente, um nó equivalente ou de restrição é descrito como:

Nó



Para completar a gramática foram definidas seis palavras chaves. As palavras **ROOT**, **NIL**, **PASS** e **RESTRICT** são usadas para representar, respectivamente, a produção inicial da gramática, a produção vazia, uma produção que deve ser ignorada na construção da AST e um nó que deve seguir uma restrição semântica. Duas outras **IGNORE** e **USE** são especiais e indicam caracteres a serem ignorados pelo analisador e a importação de outros arquivos como subparte da gramática, respectivamente.

A palavra chave **USE** pode ser usada para definir regras externas à serem reutilizadas, por exemplo podemos definir a interpretação de um tipo numérico em um arquivo e utilizá-lo em outro, este exemplo pode ser visto nas duas gramáticas abaixo.

Numeric.grmr

$$
\begin{aligned}
\textbf{ROOT} &:= < int > && (\textbf{PASS}) && \textit{; Um número é inteiro...}\\
&\quad | < float > && (\textbf{PASS}) && \textit{; ...ou decimal}\\
\\
\textbf{int} &:= /[1-9][0-9]*/ && (value:1)\\
&\quad | ''0'' && (value:1)\\
\\
\textbf{float} &:= < int > \ ''.''\ < int > && (valueF:1,3) && \textit{; 3.14}\\
&\quad | ''.''\ < int > && (value0F:2) && \textit{; .14}
\end{aligned}
$$

No exemplo, vemos que o nó **ROOT** pode ser usado como um agregador de diversos tipos de nós diferentes através do uso de **PASS**. Esse nó serve de ponto de entrada caso outros arquivos o importem.

Math.grmr

$$
\begin{aligned}
\textbf{num} &:= \textbf{USE}\ Numeric.grmr && \textit{; Importar inteiros}\\
\\
\textbf{ROOT} &:= < fat > \ ''*''\ < ROOT > && (MulOp:1,3)\\
&\quad | < fat > \ ''/''\ < ROOT > && (DivOp:1,3)\,\textbf{RESTRICT}\,(ne0:3)\\
&\quad | \textbf{NIL}\\
\\
\textbf{fat} &:= < parc > \ ''+''\ < fat > && (AddOp:1,3)\\
&\quad | < parc > \ ''-''\ < fat > && (SubOp:1,3)\\
&\quad | \textbf{NIL}\\
\\
\textbf{parc} &:= ''('' < ROOT > '')'' && (NoOp:2)\\
&\quad | < num > && (\textbf{PASS})
\end{aligned}
$$

4.2.2 Modificações e Otimizações na Gramática

Após analisada por um processo similar ao de um compilador clássico, a gramática é armazenada em memória e algumas otimizações são aplicadas. A primeira otimização é eliminação de qualquer produção inútil, essa remoção é feita com o algoritmo *flood-fill* na gramática, descrito no Algoritmo 2. A eliminação de produções inúteis atinge qualquer produção que nunca é derivada a partir da raiz assim como qualquer produção que nunca atinge um terminal, a eliminação de produções não atingíveis comprime o tamanho da gramática em memória enquanto a eliminação de produções que não atingem um terminal é essencial para que o algoritmo de derivação não caia em um laço infinito.

Algoritmo 2: Algoritmo para eliminação de produções inúteis

Entrada: Gramática: G

$P' \leftarrow \{x \mid x \in \text{G.produções} \wedge \text{x.destino} \cap \text{G.terminais} \neq \emptyset\}$;
enquanto P' *foi modificado* **faça**
    não_terminais $\leftarrow \{x.origem \; \forall x \in P'\}$;
    validos $\leftarrow$ não_terminais $\cup$ G.terminais ;
    $P' \leftarrow \{x \mid x \in \text{G.produções} \wedge \text{x.destino} \cap \text{validos} \neq \emptyset\}$;

G.produções $\leftarrow P'$;

$P' \leftarrow \{x \mid x \in \text{G.produções} \wedge \text{x.origem} = \text{G.simbolo\_inicial}\}$;
enquanto P' *foi modificado* **faça**
    para $p \in P'$ **faça**
        $P' \leftarrow P' \cup \{x \mid x \in \text{G.produções} \wedge \text{x.origem} \in \text{p.destino}\}$;

G.produções $\leftarrow P'$;

Outra modificação essencial para o funcionamento do algoritmo de derivação é a eliminação de recursão à esquerda, no caso do algoritmo implementado esse tipo de recursão não é eliminado mas é marcada, assim como recursão à direita, para que seja tratada durante a execução da derivação. É importante notar que a identificação da recursão se limita principalmente à recursão direta, e não trata todos os casos de recursão indireta.

Outra marcação especial são os casos de produções cujo resultado nunca é utilizado para construção de outro nó, nestes casos o meta-compilador nunca precisa salvar estes resultados, reduzindo o gasto de memória e tempo de execução. A última otimização realizada é o cálculo dos conjuntos *First* e *Follow* de maneira a possibilitar a identificação rápida de qual regra aplicar e possibilitar a confirmação rápida se a regra recém aplicada está ou não correta.

Finalmente, a gramática sofre as alterações necessárias para aceitar e corrigir os erros caso estes sejam aceitos. Dos três erros da Seção 4.1, apenas a substituição foi necessária para a aplicação escolhida devido ao foco dado no trabalho à correção de

palavras com marcações erradas.

4.2.3 Interpretação e Geração da AST

O analisador construído a partir de uma gramática é responsável pela criação de uma AST composta de nós que são carregados dinamicamente durante a execução do meta-compilador. Cada um desses nós estende uma classe de nó genérico que possui métodos e atributos necessários para a interpretação da árvore como um todo.

A classe de um nó deve implementar um construtor, que recebe como entrada uma lista ordenada dos seus filhos, e uma função que é chamada pelo seu pai para realizar a operação representada por ele. Por exemplo, um nó condicional, representando um comando *if* deve receber dois ou três filhos equivalentes à condição, bloco positivo e bloco negativo e testar a sua expressão condicional para saber se chamará a função de interpretação do filho correspondente a parte verdadeira ou do filho correspondente a parte falsa, caso exista.

As restrições também são implementadas usando a mesma interface de nó, porém implementam uma função especial que deve retornar se a construção dos filhos é válida ou não. A interface dos nós providencia algumas funcionalidades básicas como uma pilha de memória e uma tabela de símbolos global. Uma implementação bem feita de um nó não causa efeitos colaterais, mas isso não é garantido pela interface.

4.2.4 Limitações técnicas do meta-compilador

Devido aos detalhes de implementação do meta-compilador ele possui algumas limitações em relação ao tamanho da entrada, tamanho de gramática e sua velocidade de derivação. Primeiramente em uma entrada e gramática que gere uma árvore de derivação suficientemente alta o limite de recursão do *JavaScript* é atingido gerando um erro, no trabalho isso forçou o uso de textos menores como *tweets* para as aplicações teste. A segunda limitação é relativa ao número máximo de regras de produção por não terminal, que devido a escolha do uso de um carácter para indexação é limitado à 256 produções por não terminal, sofrendo uma perda de performance significativa ao atingir 128 regras.

A última limitação é relativa ao tempo de execução que devido a uma implementação sem atenção aos detalhes de otimização leva um tempo relativamente longo para o processamento de uma entrada. O tempo de execução relativo aos algoritmos existentes é de duas a três vezes pior. Esse tempo de execução ainda é aceitável executando a análise dos *tweets* em menos de um segundo por *tweets* (aproximadamente 900ms por *tweet*).

Todas essas limitações são relativas aos detalhes de implementação escolhidos de maneira a simplificar o processo, em sua grande parte com o uso de estruturas de dados

corretas e algumas otimizações de código é possível superar todas essas limitações. Por exemplo o limite da recursão poderia ser evitado utilizando técnicas de recursão de cauda, e o tempo poderia ser reduzido utilizando linguagens de programação com um melhor desempenho, ou a partir de uma otimização do tempo de acesso da estrutura do *Buffer*.

4.2.5 Aplicação à Marcação

Para o processo completo foram também utilizados alguns algoritmos básicos para *tokenização* e *tagging* inicial das entradas, utilizando a biblioteca *Node-Natural* (UMBEL; ELLIS; MULL, 2011), em especial, sua implementação de *AggressiveTokenizerPt* é utilizada como pré-processamento das frases de entrada. A Figura 7 apresenta um fluxograma geral do funcionamento da técnica.

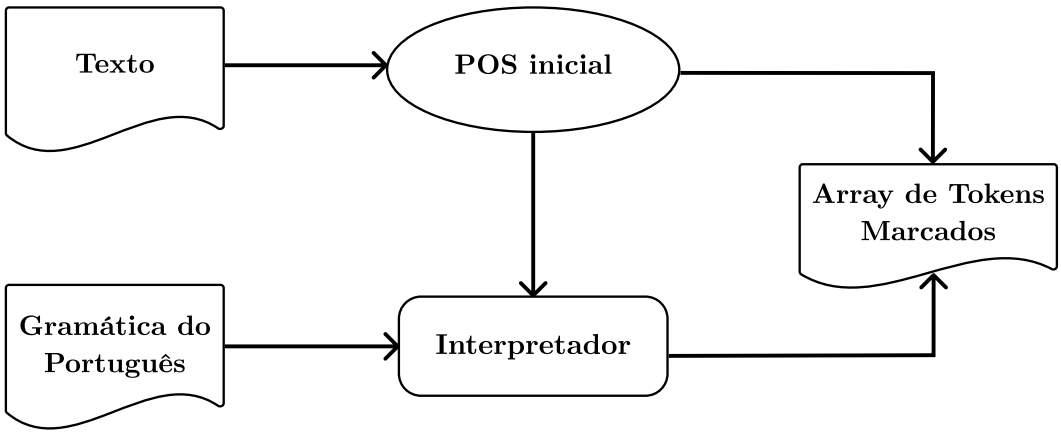


Figura 7 – Fluxograma do Funcionamento da Técnica Desenvolvida no Trabalho

No fluxograma, é possível observar que a derivação complementa o resultado obtido por um *POS tagger* mais simples, que não possua a desambiguação, de maneira a gerar um resultado mais completo. No trabalho, foi utilizado um *tagger* baseado em dicionário simples disponível através da biblioteca NLTK.

4.3 Desenvolvimento da Gramática

Para o bom funcionamento do algoritmo é necessária uma boa representação do idioma através de uma gramática, porém como discutido na literatura (SILVA, 2006) a gramática portuguesa, em especial a brasileira, apresenta algumas nuances que a tornam difícil de se tratar em aplicações computacionais.

Durante a revisão da literatura não foi encontrada uma gramática que descrevesse o português brasileiro em sua completude com o formalismo necessário para a ferramenta. Portanto, foi desenvolvida uma gramática inicial que descreve orações absolutas com a ajuda de um livro de gramática (BECHARA, 2015).

A base da gramática construída manualmente é o processo de análise de orações, e, em sua versão inicial focava em orações simples construídas por Sujeito e Predicado. A validação desta gramática, foi feita utilizado um corpus anotado do português (FONSECA; ROSA, 2013), a partir dessa validação a construção manual da gramática foi descartada rapidamente uma vez que a versão inicial descrevia a base das frases em português, porém não era capaz de identificar corretamente boa parte das construções válidas do idioma.

A gramática inicial foi utilizada em conjunto com algoritmos genéticos para indução da gramática (CHOUBEY; KHARAT, 2010). A função de *fitness* usada para essa indução foi:

$$f_o = (W_p * N_+) - (W_n * N_-) + (\frac{1}{W_r} - |G|)$$

Onde $W_p, W_n e W_r$ são pesos associados ao número de exemplos corretamente aceitos ou rejeitados, número de exemplos não corretamente classificados e ao número total de regras, a fração $\frac{1}{W_r}$ foi usada para que o peso tenha uma relação proporcional a complexidade desejada. Esta função incentiva uma gramática correta mas simples, penalizando gramáticas muito mais complexas que o esperado. A escolha dos parâmetros $W_p = 0.7$, $W_n = 1.0$ foi feita a partir de testes com populações menores, já $W_r = 0.008$ foi escolhido devido a limitações de performance da ferramenta de derivação. N_+ e N_- representam respectivamente o número de exemplos corretamente e incorretamente derivados por esta gramática, enquanto $|G|$ indica o tamanho da gramática.

O cruzamento foi feito usando cruzamento uniforme, gerando dois novos indivíduos a partir dos seus predecessores, é importante notar que a representação dos indivíduos foi feita como uma lista de símbolos que representam as regras de derivação diretamente. Este método de representação é similar a técnica discutida na Seção 3.1 apenas sem o passo de tradução. A mutação implementada é simplesmente a troca de alguns símbolos aleatoriamente.

Para acelerar a evolução do algoritmo as populações iniciais foram baseadas em partes da gramática inicial, e de execuções anteriores do algoritmo que tinham bons resultados. Várias populações foram evoluídas independentemente e ao fim a com o maior valor de *fitness* foi escolhida para a execução do algoritmo. Várias populações iniciais compostas de diferentes partes da gramática foram usadas como ponto de partida.

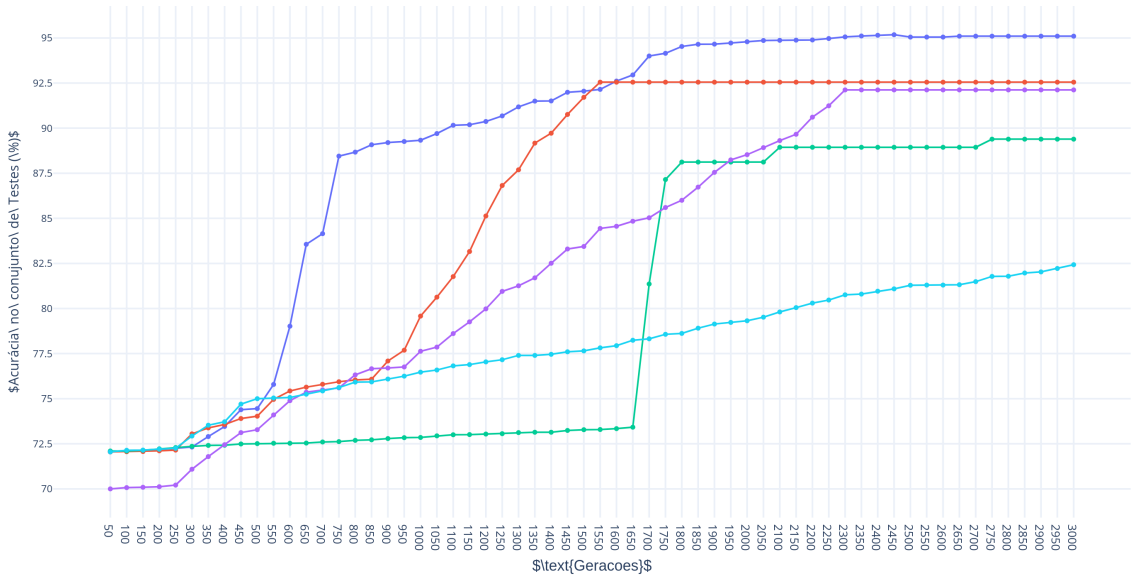


Figura 8 – Gráfico da evolução das populações

Este processo gerou uma gramática que era capaz de aceitar aproximadamente 95% da base de testes, esse valor foi aceito pois não teve melhora significativa após 500 gerações. O gráfico da Figura 8 mostra a evolução das 5 melhores populações em relação à sua acurácia de classificação no **MAC MORPHO**.

4.4 Aplicações Teste

Duas aplicações foram testadas com e sem o processo de correção de erros para verificar se houve melhorias. Em ambos testes, a correção foi focada na substituição de emojis por palavras equivalentes usando a marcação correta. O uso da palavra correta no contexto deve ser com intuito de preservar a intenção original da frase. Isso significa que foi necessário criar uma tabela correlacionando cada emoji a uma palavra em uma determinada função gramatical. Na Tabela 1, uma parte de algumas linhas dessa tabela pode ser vista.

Emoji	Substantivo	Verbo	Adjetivo	Pronome	Interjeição
😂	Risada	Rir	Rindo	-	Haha!
😭	Choro	Chorar	Triste	-	-
👦	Homem	-	-	Ele	-

Tabela 1 – Substituições de um emoji

De acordo com a tabela, se por exemplo o emoji 😂 aparecer como substantivo, ele

será substituído pela palavra Risada. Outro aspecto a ser observado é que nem todos os emojis podem aparecer em todas as funções gramaticais, por exemplo, não há ocorrência de 😂 como pronome. Nenhum emoji aparece como **Artigo**, **Conjunção** ou **Preposição** devido à natureza dessas classes gramaticais.

Essa tabela foi construída manualmente após uma execução preliminar para extração dos emojis existentes na base de dados, é importante notar que a construção dessa tabela foi feita sem o contexto nos quais os emoji apareceram, de modo a não influenciar os resultados.

A primeira aplicação foi a análise de sentimentos, onde se esperava que essa correção significasse que cada palavra fosse interpretada pela análise de sentimentos como uma palavra correspondente significativa e não ignorada pelo algoritmo. O algoritmo escolhido foi um ranqueamento simples (LOPER; BIRD, 2002), onde cada palavra recebe um *score*, palavras desconhecidas foram tratadas como neutras nesse teste.

A segunda aplicação foi feita no contexto de um tradutor simples, que depois de substituir um emoji da mesma maneira, traduz a frase normalmente e então a palavra equivalente foi substituída de volta pelo emoji, nesta tarefa houve um *input* manual quando o algoritmo não conseguia identificar qual palavra havia sido traduzida.

Para a tradução foi utilizada a metrica BLEU ou *Bilingual Evaluation Understudy*, uma pontuação calculada de acordo com a equação:

$$BP * e^{\sum_{i=1}^{N} w_n * log P_n}$$

Essa métrica estuda sequências de n palavras, chamadas de n-gramas, na equação N representa o tamanho do maior n-grama, normalmente $N = 4$, w_n representa um peso para cada um dos n-gramas analisados, normalmente $w_n = \frac{1}{N}$.

BP é um termo conhecido como penalidade de brevidade, ele penaliza traduções que são menores que o objetivo, forçando as traduções a se conformarem ao tamanho esperado. É importante que essa penalidade não priorize também traduções que são muito mais verbosas, para isso ela é limitada ao valor de um sendo definida como:

$$BP = min(e^{1-\frac{|\text{referência}|}{|\text{candidato}|}}, 1.0)$$

O termo final do calculo P_n representa uma precisão modificada, seu calculo é feito a partir da contagem da aparição das palavras em todas as referências usadas para testar a tradução, o calculo é definido como:

$$P_n = \frac{\sum_{c \in C} \sum_{N \in c} min(|n \in c|, max(|N| \in k \forall k \in C))}{\sum_{c \in C} \sum_{N \in c} |n \in c|}$$

Onde C é o conjunto de candidatos, n é o tamanho dos n-gramas e N representam os próprios n-gramas. O BLEU *score* representa uma porcentagem referente à qualidade da tradução.

4.4.1 As Bases de Dados

As bases de dados escolhidas são compostas de *tweets*, juntas as bases possuem cerca de 10.000 *tweets*, sendo que a base referente à *tweets* sobre a copa do mundo compõe a maior parte com 7.100 entradas enquanto foram coletados apenas 2.700 *tweets* sobre as eleições de 2018.

A escolha dessas bases de dados foi feita levando em conta que os tópicos a quais elas se referem são polêmicos e portanto polarizantes com uso extenso de ironia e sarcasmo, levando a resultados interessantes de um teste baseado em análise de sentimento. Um segundo fator que pesou para a escolha dessas bases é o fato de que já foram feitas análises de sentimento relativas à elas na literatura (MANSSOUR; SILVEIRA, 2015; BANHESSE; NADAI, 2018).

5 Resultados

O algoritmo construído foi testado como descrito na Seção 4.4 e os resultados foram analisados de acordo com métricas simples. No caso da Análise de Sentimentos, a pontuação foi dada de acordo com a acurácia em relação ao valor esperado. Já no caso da tradução, foi feita uma análise qualitativa e posteriormente calculado o valor da métrica BLEU em relação a dois tradutores comerciais.

5.1 Resultados da Análise de Sentimentos

A precisão do algoritmo de análise de sentimentos foi calculada antes e após a correção, descrita na seção 4, produzindo resultados mostrados na Tabela 2. A precisão representa a porcentagem de *tweets* que foram analisados corretamente. Cada uma das instâncias representa uma parte do conjunto de dados completo dividido igualmente. Todas as partes estavam sujeitas ao mesmo algoritmo e representam cerca de 2.000 *tweets*.

Os algoritmos de análise de sentimentos usam cada palavra para calcular uma pontuação que varia de $[-1.0\, a\, 1.0]$, representando um *tweet* que é 100% negativo em $-1,0$, para um tweet que é 100% positivo com uma pontuação de $1,0$. Um tweet com uma pontuação de $0,0$ significa que o sentimento é neutro. A precisão foi calculada como o erro absoluto em relação ao erro total possível, conforme indicado pela fórmula:

$$1.0 - \frac{\Delta score}{2.0}$$

Nesta fórmula, um tweet que deveria ter sido classificado como $1,0$, mas recebeu a pontuação -1,0, tem uma precisão de 0%.

Como visto, a correção melhorou os resultados em uma média de cerca de 3%. Esses resultados nos mostram que tentar pré-processar algumas informações simbólicas como emoticons, emoji ou *gírias* pode ajudar a obter melhores resultados com os algoritmos de NLP. Isso parece óbvio, mas a maioria dos processos descarta essas palavras para manter a simplicidade da implementação.

Notavelmente, os exemplos mais afetados foram aqueles em que as palavras eram positivas, mas o emoji era negativo, sendo usado como meio de sarcasmo, por exemplo, o *tweet* na figura 9 foi classificado anteriormente como positivo, mas agora é classificado corretamente como negativo, neste exemplo, quando o emoji 💩 foi classificado como uma interjeição e substituído por uma vaia que alterou o resultado.

Instância	Acc_0	Acc	ΔAcc
1	91.609%	95.363%	4.10%
2	91.642%	95.669%	4.39%
3	92.056%	92.709%	0.71%
4	91.675%	95.406%	4.07%
5	91.768%	93.223%	1.59%
6	92.375%	93.131%	0.82%
7	91.070%	91.898%	0.91%
8	91.884%	92.062%	0.19%
9	92.465%	96.982%	4.88%
10	91.759%	96.678%	5.36%
Avg	91.572%	94.312%	2.99%

Tabela 2 – Acuracia da análise de sentimentos antes (Acc_0) e pós (Acc) a correção, na base de dados da copa

Figura 9 – Exemplo de *tweet* que era classificado de forma errada

Neste mesmo exemplo, a palavra “pacas”, usada como gíria, também foi afetada pelo algoritmo, sendo previamente marcada pelo dicionário como um substantivo (o animal paca), sendo posteriormente classificada como advérbio (significando muito bom).

Figura 10 – Exemplo de tweet que era classificado de forma errada

Para esta aplicação, também foi utilizada a segunda base de *tweets*, referente às eleições de 2018. Nessa rodada de testes os *tweets* foram divididos em 5 instâncias, e os

resultados podem ser vistos na tabela 3. Nesta base o uso de emojis foi consideravelmente menor, e sempre acompanhado de uma frase que já possuía um sentimento bem definido, por esses motivos observamos uma melhoria significativamente menor quando aplicada a correção.

Instância	Acc_0	Acc	ΔAcc
1	93.74%	94.06%	0.32%
2	94.05%	94.62%	0.57%
3	92.75%	93.05%	0.30%
4	93.12%	94.13%	0.01%
5	94.06%	95.57%	1.51%
Avg	93.544%	94.286%	0.542%

Tabela 3 – Acuracia da análise de sentimentos antes (Acc_0) e pós (Acc) a correção, na base de dados de eleições

A melhoria de em média 0.542% pode ser considerada uma melhoria desprezível, principalmente quando considerando o custo necessário para consegui-la, isso mostra que o algoritmo é útil em situações especificas, isso é apoiado pela comparação com outros algoritmos similares feita na seção 5.1.1

5.1.1 Comparação com Outros Métodos de Correção de Marcação

Para comparação da eficácia da correção o algoritmo foi comparado com um algoritmo de indução de POS a partir de trigramas, grupos de três palavras em sequência (HALáCSY; KORNAI; ORAVECZ, 2007), uma vez que este tipo de algoritmo atualmente é um dos melhores para a tarefa (ACLWEB, 2019). O algoritmo foi executado nas mesmas condições ao desenvolvido, porém apenas atuou na classificação de palavras desconhecidas na base de testes.

Primeiramente os algoritmos foram comparados em relação a acurácia quanto a classificação das palavras contidas no MAC MORPHO. Neste caso, o algoritmo desenvolvido foi inferior ao baseado em trigramas com uma média de 96.46% contra 95.10%. Isso se deve à origem das palavras coletadas, uma vez que os textos da base são formais portanto possuem poucos erros de derivação.

Após esse teste, o algoritmo foi testado juntamente com a análise de sentimento, e os resultados obtidos foram comparados, na tabela abaixo:

Como pode ser observado ambos algoritmos obtiveram resultados similares, em média, mas o algoritmo desenvolvido apresentou um resultado mais consistente indicado por uma menor variância.

Instância	$Acc_{\text{gramática}}$	$Acc_{trigramas}$
1	95.363%	97.86%
2	95.669%	89.46%
3	92.709%	96.31%
4	95.406%	93.32%
5	93.223%	96.54%
6	93.131%	98.3%
7	91.898%	90.56%
8	92.062%	90.13%
9	96.982%	95.66%
10	96.678%	94.93%
Avg	94.312%	94.316%
σ^2	2.992%	8.643%

Tabela 4 – Acuracia da análise de sentimentos antes (Acc_0) e pós (Acc) a correção

5.2 Resultados da Tradução

Os melhores resultados vieram da tradução automática, onde a ordenação das palavras é um problema conhecido (BARREIRO; RANCHHOD, 2005). Ao transformar os emojis em palavras e de volta em emojis, as frases traduzidas se tornam mais compreensíveis e parecidas com as humanas do que antes da correção. Isso é perceptível principalmente em frases com a ordem sujeito-objeto-verbo, onde o verbo foi substituído pelo emoji.

Um exemplo onde essa ordem pode ser vista, é o *tweet* da figura 10, "Esse Brasil vai me 🔪 do coração", que foi traduzido anteriormente mantendo a ordem como "*This Brazil will me* 🔪 *of the heart*", pelo algoritmo original, agora é traduzido na ordem correta sujeito-verbo-objeto em inglês "*This Brazil will* 🔪 *me from the heart*".

Outro tipo de frase que apresenta uma melhoria significativa são frases com o uso da partícula possessiva do inglês "*is*" como por exemplo na frase "⚽ da Alemanha 🙄 🙄 🙄" onde a expressão "da Alemanha" deveria ser traduzida como "Germany's" mas antes do tratamento de emojis era interpretado incorretamente como "from Germany".

Para calcular a pontuação do algoritmo foram utilizadas como referência traduções feitas pelas ferramentas comerciais Google Translate e Bing Translate, através de suas APIs. Ao calcular o BLEU *score* antes e depois do processo de substituição de emojis foi observada uma melhoria de em média 2,21% (de 95,83% para 98,04%) na pontuação.

Como uma última verificação foi feita uma tradução manual de uma pequena parte dos *tweets*, 30 *tweets* sem nenhum emoji, e 80 tweets com emoji escolhidos aleatoriamente. Essa tradução foi feita por volutuários que se declararam em um nível intermediário de conhecimento em inglês. Todos os *tweets* escolhidos possuem pelo menos duas traduções

feitas por pessoas diferentes.

O score BLEU foi então recalculado de acordo com essas traduções. Isso foi feito para verificar se não existe um viés nos tradutores automáticos que também está presente na correção. O score obtido teve uma melhoria média superior à anterior de 4.1%, porém com scores levemente menores, em média 80.4% antes da correção e 83.5% depois da correção. A tradução manual foi feita por um tradutor sem contato com os testes anteriores de modo a minimizar a possibilidade de viés.

O maior aumento da média indica que, após a correção, os resultados apresentam uma versão mais próxima das frases feitas pela tradução humana, mesmo sem a modificação do algoritmo utilizado.

Já a score menor pode ser explicado por escolhas de palavras do tradutor como por exemplo no *tweet*: "Jogadores da #GER estão mais envergonhados do #7x1 que os jogadores do #BRA" que foi traduzido pelo algoritmo e os tradutores automáticos como "*#GER players so much more ashamed of #7x1 than #BRA players*" e pelo tradutor humano como "*#GER players are more embarassed of #7x1 than #BRA players*". Outro fator para a diminuição pode ser explicado por uma tradução baseada em contexto como no tweet "D E C E P Ç Ã O" que ficou idêntico no algoritmo mas foi traduzido pelo tradutor humano como "*D E L U S I O N*", este último tipo de erro poderia ser tratado ao combinar a técnica desenvolvida com outros tipos de corretor.

6 Considerações Finais

Na era da internet, é de extrema importância que sejam estudadas as variantes populares do idioma, e que estas variantes sejam consideradas na concepção de um algoritmo ou na criação de uma aplicação.

Neste trabalho, foi testada uma técnica de correção de marcação desenvolvida a partir de algoritmos existentes da área de compiladores. A técnica foi implementada a partir de um compilador desenvolvido previamente que foi modificado para trabalhar como parte do processo de NLP. Os resultados mostraram que o pré-processamento proposto pode, em algumas situações, melhorar a aplicação final. Resultados positivos também reforçam a importância do pré-processamento em aplicações de NLP.

A representação de erros de derivação através de modificações da gramática usando as regras descritas neste trabalho podem ser utilizadas para diversas aplicações onde esses tipos de erros são relevantes. O trabalho também apresentou uma oportunidade para o estudo da gramática portuguesa e serve como base para a formalização da mesma, além da obtenção de uma gramática através de algoritmos de evolução. Finalmente a ferramenta capaz de realizar as correções pode ser re-adaptada e ser utilizada como um compilador capaz de corrigir pequenos erros de sintaxe.

O trabalho pode ser continuando em diversas frentes. A melhoria da ferramenta estendida pode ser aplicada à diversos contextos onde a correção em um processo de derivação é necessária. As correções podem ser aplicadas no contexto de linguagens naturais em outros passos além da marcação ou no contexto de linguagens de programação como por exemplo em processos de verificação formal.

A da formalização da gramática também pode ser estendido e aproveitado para diversas outras aplicações no campo da linguística formal. A técnica de obtenção pode ser refinada para se obter uma gramática completa do estado atual do idioma.

Finalmente em ralação às inovações acadêmicas a técnica de tradução contextualizada de emojis pode ser considerada uma das maiores contribuições deste trabalho, uma vez que a partir dessa tradução leitores de tela e ferramentas de acessibilidade podem informar aos usuários sobre os símbolos e suas intenções.

Referências

ACLWEB. *POS Tagging (State of the art) - ACL Wiki*. 2019. Disponível em: <https://aclweb.org/aclwiki/POS_Tagging_(State_of_the_art)>. Citado 2 vezes nas páginas 26 e 46.

AHO, A. V.; ULLMAN, J. D. *Principles of Compiler Design*. [S.l.]: Narosa Publ. House, 1999. Citado 2 vezes nas páginas 21 e 24.

ALAMOODI, A. et al. Sentiment analysis and its applications in fighting covid-19 and infectious diseases: A systematic review. *Expert systems with applications*, Elsevier, p. 114155, 2020. Citado na página 27.

BACKUS, J. W. Revised report on the algorithmic language ALGOL 60. v. 5, p. 349–367, 1963. Citado 2 vezes nas páginas 17 e 20.

BAETA, Y. Uma ferramenta para a criação de linguagens de programação utilizando as definições léxico-sintáticas e semânticas. Trabalho de Conclusão de Curso. 2018. Citado na página 17.

BAEZA-YATES, R. Challenges in the interaction of information retrieval and natural language processing. p. 445–456, 2004. Citado na página 15.

BANHESSE, E. L.; NADAI, R. *Sentiment Analysis for 2018 Presidential Election*. 2018. Disponível em: <https://github.com/rdenadai/sentiment-analysis-2018-president-election>. Citado 2 vezes nas páginas 32 e 43.

BARREIRO, A.; RANCHHOD, E. Machine translation challenges for portuguese. v. 28, n. 1, p. 3–18, 2005. Citado 2 vezes nas páginas 16 e 47.

BATES, M. Models of natural language understanding. v. 92, p. 9977–9982, 1995. Citado na página 14.

BECHARA, E. *Moderna Gramática Portuguesa*. [S.l.]: Editora Nova Fronteira, 2015. Citado 4 vezes nas páginas 16, 26, 28 e 40.

BIRD, S.; KLEIN, E.; LOPER, E. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. [S.l.]: O'Reilly Media, 2009. Citado 2 vezes nas páginas 15 e 31.

BRN.AI, B. *Chatbot Report 2019: Global Trends and Analysis*. [S.l.]: Chatbots Magazine, 2019. Citado na página 15.

CAMBRIA, E.; WHITE, B. Jumping NLP curves: A review of natural language processing research. v. 9, n. 2, p. 48–57, 2014. Citado na página 14.

CARREIRO, M. N.; DIAS, E. *"O português brasileiro precisa ser reconhecido como uma nova língua. E isso é uma decisão política"*. 2015. Library Catalog: www.jornalopcao.com.br. Disponível em: <https://www.jornalopcao.com.br/entrevistas/o-portugues-brasileiro-precisa-ser-reconhecido-como-uma-nova-lingua-e-isso-e-uma-decisao-politica-> . Citado 2 vezes nas páginas 16 e 27.

CHOMSKY, N. Three models for the description of language. v. 2, n. 3, p. 113–124, 1956. Citado 2 vezes nas páginas 18 e 20.

CHOUBEY, N.; KHARAT, M. Sequential structuring element for CFG induction using genetic algorithm. v. 975, p. 8887, 2010. Citado 2 vezes nas páginas 30 e 40.

CHOUBEY, N. S.; PANDEY, H. M.; KHARAT, M. Developing genetic algorithm library using Java for CFG induction. 2011. Citado na página 30.

CORREIA, M.; LEMOS, L. S. P. de. *Inovação lexical em português*. [S.l.]: Colibri, 2005. Citado na página 16.

CPLP, C. d. P. d. L. P. *CPLP - Comunidade dos Países de Língua Portuguesa*. 2019. Disponível em: <http://www.cplp.org>. Citado 2 vezes nas páginas 16 e 27.

CURY, D. et al. Defining a lexicalized context-free grammar for a subdomain of portuguese language. p. 74–79, 2002. Citado na página 15.

FARGHALY, A.; SHAALAN, K. Arabic natural language processing. v. 8, p. 1–22, 2009. Citado na página 15.

FELDMAN, R. Techniques and applications for sentiment analysis. *Communications of the ACM*, ACM New York, NY, USA, v. 56, n. 4, p. 82–89, 2013. Citado na página 27.

FONSECA, E. R.; ROSA, J. L. G. Mac-morpho revisited: Towards robust part-of-speech tagging. In: *Proceedings of the 9th Brazilian symposium in information and human language technology*. [S.l.: s.n.], 2013. p. 98–107. Citado na página 40.

FORD, B. Packrat parsing: a practical linear-time algorithm with backtracking. 2002. Citado na página 17.

FRIEDL, J. E. F. *Mastering regular expressions*. [S.l.]: O'Reilly, 2002. Citado 2 vezes nas páginas 18 e 19.

GACHOT, D. A.; LANGE, E.; YANG, J. The systran NLP browser: an application of machine translation technology in cross-language information retrieval. v. 2, p. 105–118, 1998. Citado na página 14.

GRAMARLY. 2020. Disponível em: <https://www.grammarly.com/>. Citado na página 31.

GRUNE, D. et al. *Modern Compiler Design*. [S.l.]: Springer, 2016. Citado 2 vezes nas páginas 22 e 25.

GöRSKI, E. M.; COELHO, I. L. Variação linguística e ensino de gramática. v. 10, 2009. Citado 2 vezes nas páginas 15 e 27.

HALáCSY, P.; KORNAI, A.; ORAVECZ, C. Hunpos: an open source trigram tagger. p. 209–212, 2007. Citado 2 vezes nas páginas 31 e 46.

HARTMANN, N. et al. Portuguese word embeddings: Evaluating on word analogies and natural language tasks. 2017. Citado na página 16.

HERRING, S. C. Language and the internet. 2008. Citado na página 14.

INDURKHYA, N.; DAMERAU, F. J. *Handbook of Natural Language Processing*. [S.l.]: Taylor & Francis, 2010. Citado 3 vezes nas páginas 15, 25 e 26.

KAUFFMANN, E. et al. A framework for big data analytics in commercial social networks: A case study on sentiment analysis and fake review detection for marketing decision-making. *Industrial Marketing Management*, Elsevier, v. 90, p. 523–537, 2020. Citado na página 27.

KIM-RENAUD, Y.-K. *The Korean Alphabet: Its History and Structure*. [S.l.]: University of Hawaii Press, 1997. Citado na página 14.

KRULEE, G. K. *Computer processing of natural language*. [S.l.]: Prentice-Hall, Inc., 1991. Citado na página 26.

Levenshtein, V. I. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, v. 10, p. 707, 1966. Citado na página 32.

LJUBEŠIć, N.; DANIJELA, M. Lemmatization and morphosyntactic tagging of croatian and serbian. p. 48–57, 2013. Citado na página 15.

LOPER, E.; BIRD, S. Nltk: The natural language toolkit. In: *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*. [S.l.: s.n.], 2002. Citado na página 42.

LORIA, S. *TextBlob: Simplified Text Processing - TextBlob Documentation*. [S.l.]: TextBlob, 2018. Citado na página 15.

MacNeill, J. Notes on the distribution, history, grammar, and import of the irish ogham inscriptions. v. 27, p. 329–370, 1908. Citado na página 14.

MANNING, C. D. Part-of-speech tagging from 97% to 100%: Is it time for some linguistics? p. 171–189, 2011. Citado na página 15.

MANSSOUR, I. H.; SILVEIRA, M. S. 7x1pt: Um corpus extraído do Twitter para análise de sentimentos em língua portuguesa. 2015. Citado 2 vezes nas páginas 32 e 43.

MENEZES, P. B. *Linguagens formais e autômatos*. [S.l.]: Sagra-Dcluzzato, 1998. Citado 2 vezes nas páginas 18 e 20.

MUNIZ, M. C.; NUNES, M. D. G. V.; LAPORTE, E. Unitex-pb, a set of flexible language resources for brazilian portuguese. 2005. Citado na página 16.

NETTO, W. F. O acento na língua portuguesa. 2007. Citado na página 16.

NUNE, M. das G. V. O processamento de línguas naturais: para quê e para quem? p. 12, 2008. Citado na página 16.

Pro Writting Aid. 2020. Disponível em: <https://prowritingaid.com/>. Citado na página 31.

RADFORD, A. et al. Language models are unsupervised multitask learners. v. 1, n. 8, 2019. Citado na página 14.

RANCHHOD, E. et al. Portuguese large-scale language resources for NLP applications. 2004. Citado na página 16.

ROCHA, L. et al. SACI: Sentiment analysis by collective inspection on social media content. v. 34, p. 27–39, 2015. Citado na página 14.

RODRIGUES, R.; OLIVEIRA, H. G.; GOMES, P. NLPPort: A pipeline for portuguese NLP (short paper). In: *7th Symposium on Languages, Applications and Technologies (SLATE 2018)*. [S.l.: s.n.], 2018. p. 18:1–18:9. Citado na página 16.

SANTOS, D.; BARREIRO, A. On the problems of creating a consensual golden standard of inflected forms in. 2004. Citado na página 16.

SAPLING. 2020. Disponível em: <https://sapling.ai/>. Citado na página 31.

SETHI, R.; ULLMAN, J. D.; LAM monica S. *Compiladores: princípios, técnicas e ferramentas*. [S.l.]: Pearson Addison Wesley, 2008. Citado na página 21.

SHIFFMAN, D. The evolution of code. In: _____. [S.l.]: The Magic Book Projekt, 2012. Citado na página 29.

SILVA, B. C. D. da. O estudo lingüístico computacional da linguagem. v. 41, n. 2, p. 103–138, 2006. Citado na página 39.

SILVA, R. R.; LIMA, S. M. B. Consultas em bancos de dados utilizando linguagem natural. 2013. Citado na página 14.

STEINMETZ, K. *Oxford's 2015 Word of the Year Is This Emoji*. [S.l.]: Time Magazine, 2015. Citado na página 14.

THE Oxford handbook of computational linguistics. [S.l.]: Oxford Univ. Press, 2009. Citado na página 18.

UMBEL, C.; ELLIS, R.; MULL, R. *General natural language facilities for node*. 2011. Disponível em: <https://github.com/NaturalNode/natural>. Citado 2 vezes nas páginas 31 e 39.

WANG, S.-C. Genetic algorithm. In: *Interdisciplinary Computing in Java Programming*. [S.l.]: Springer, 2003. p. 101–116. Citado na página 28.

ZHANG, X. *The Evolution Of Natural Language Processing And Its Impact On AI*. [S.l.]: Forbes Magazine, 2018. Citado 2 vezes nas páginas 14 e 27.

ZúQUETE, J. P. Beyond reform: the orthographic accord and the future of the portuguese language: South european atlas. v. 13, n. 4, p. 495–506, 2008. Citado 3 vezes nas páginas 14, 16 e 27.