Universidade Federal do Paraná

Junior Ferri

Abordagem modular baseada em dicionário para reconhecimento de entidades nomeadas através de associação aproximada

Curitiba PR
2016

Junior Ferri

Abordagem modular baseada em dicionário para reconhecimento de entidades nomeadas através de associação aproximada

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Marcos Didonet Del Fabro.

Curitiba PR
2016

F388a Ferri, Junior
Abordagem modular baseada em dicionário para reconhecimento de entidades nomeadas através de associação aproximada / Junior Ferri. – Curitiba, 2016.
72 f. : il. ; 30 cm.

Dissertação - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática, 2016.

Orientador: Marcos Didonet Del Fabro .
Bibliografia: p. 47-50.

1. Sistemas de recuperação da informação. 2. Processamento eletrônico de dados. 3. Tecnologia da informação. 4. Fonética. 5. Conjunto de caracteres (Processamento de dados). I. Universidade Federal do Paraná. II.Didonet Del Fabro, Marcos. III. Título.

CDD: 004.2

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
Setor CIÊNCIAS EXATAS
Programa de Pós Graduação em INFORMÁTICA
Código CAPES: 40001016034P5

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **JUNIOR FERRI**, intitulada: "**ABORDAGEM MODULAR BASEADA EM DICIONÁRIO PARA RECONHECIMENTO DE ENTIDADES NOMEADAS ATRAVÉS DE ASSOCIAÇÃO INEXATA**", após terem inquirido o aluno e realizado a avaliação do trabalho, são de parecer pela sua aprovação.

Curitiba, 29 de Agosto de 2016.

Prof MARCOS DIDONET DEL FABRO
Presidente da Banca Examinadora (UFPR)

Prof FABIANO SILVA
Avaliador Interno (UFPR)

Prof HEGLER CORREA TISSOT (UFPR)

Prof EMERSON CABRERA PARAISO
Avaliador Externo (PUC/PR)

Agradecimentos

Eu gostaria de agradecer:

Primeiramente meu orientador Professor Marcos Didonet Del Fabro, pela oportunidade e pela orientação durante todo o desenvolvimento desse trabalho.

O Dr. Hegler Correa Tissot, pela colaboração e suporte desde a definição do projeto até a sua conclusão.

O Programa de Pós-gradução em Informática da Universidade Federal do Paraná por me proporcionar a possibilidade de desenvolver meus estudos.

O Departamento de Expressão Gráfica da Universidade Federal do Paraná pela compreensão e apoio.

Minha esposa pela motivação para iniciar e pelo apoio e compreensão durante todo o tempo que estive envolvido com esse trabalho.

E todos que de alguma forma me ajudaram durante todo o meu curso de mestrado.

Resumo

As técnicas de extração de informações estão sempre evoluindo para serem capazes de trabalhar com a quantidade crescente de dados disponíveis através de textos em linguagem natural e não estruturados. Destacamos a subtarefa da extração de informação conhecida como *reconhecimento de entidades nomeadas* baseado em dicionário, que realiza a identificação de sequências de caracteres que representam entidades de um determinado grupo, e o bom desempenho dessa subtarefa é fundamental para um bom processo de extração de informação. O reconhecimento de entidades nomeadas (NER) permite definir os sujeitos que são abordados pelo texto como organizações, pessoas, locais, etc. Pontos que ainda são desafios dentro da subtarefa de NER para sistemas baseados em dicionário são a presença de erros ortográficos nos textos e a existência de poucos sistemas de NER capazes de trabalhar em diferentes contextos. Esse trabalho apresenta uma abordagem para o reconhecimento de entidades nomeadas baseado em dicionário. Para trabalhar com textos que podem apresentar erros ortográficos, é utilizada uma busca por associação aproximada baseada na distância de edição entre as sequências de caracteres que representam a entrada do dicionário e as sub-partes do texto. Para promover a redução do erro entre as sequências de caracteres (SC) e facilitar a busca por associação aproximada são utilizados algoritmos de transformação. Esses algoritmos permitem a busca sobre o dicionário encontrar uma quantidade maior de entidades se comparada com as buscas utilizando as SCs originais para um mesmo valor da distância de edição aceita. As transformações também colaboram com a redução do tamanho das SCs e com a criação de mais prefixos similares, promovendo uma redução no tamanho da árvore de prefixo que indexa o dicionário. Para melhorar a precisão da nossa abordagem, disponibilizamos recursos de filtragem que fazem uso de métricas de similaridade para eliminar entidades falsas que foram retornadas da busca sobre o dicionário. Nossa abordagem também foi projetada para permitir a configuração de alguns componentes de forma a ser adaptada para diferentes casos de estudo.

Palavras-chave: Reconhecimento de entidades nomeadas, Associação Aproximada de Sequências de Caracteres, Conversão fonética.

Abstract

The information extraction techniques are always evolving to be able to work with the increasing amount of unstructured data available through texts in natural language. We highlight the information extraction subtask known as dictionary-based *named entity recognition*, which performs the identification of strings that represent entities of a particular group, and the good performance of this sub-task is critical for a good extracting information process. The named entity recognition (NER) defines the nouns that are covered by the text as organizations, people, places, etc. Some subjects that still represent chalenges in the sub-task of NER for currently systems that are dictionary-based are the presence of spelling errors in the text and the existence of few NER systems that are able to work in different contexts. This work presents an approach of a dictionary-based named entity recognition. Looking to work with texts that may have spelling errors, we use an approximate string matching search based on edit distance between the strings that represent the entries of the dictionary and the substrings of the text. To further the reduction of the error between the strings and facilitate the search using approximate matching we used transformation algorithms. These algorithms allow the search on the dictionary find a greater amount of entities if compared with the search using the original strings, for the same value of *Edit Distance*. Transformations also promote the strings size reduction and create more similar prefixes, promoting a reduction in the size of the prefix tree (trie) that indexes the dictionary. To improve the precision of our approach, we provide filtering capabilities that make use of similarity metrics to eliminate false entities that have been returned from the search on the dictionary trie. Our approach is also designed to enable the configuration of some components to be adapted to different study cases.

Keywords: Named entity recognition, Approximate string matching, Phonetic conversion.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

ASM	Approximate String Matching
DARPA	Defense Advanced Research Projects Agency
DM	Double Metaphone
ED	Edit Distance
EI	Extração de Informação
JW	Jaro-Winkler
LCS	Longes Common SubString
MUC	Message Understanding Coference
NER	Named Entity Recognition
SC	Sequência de Caracteres
SS	String_{Sim}

Capítulo 1

Introdução

A escrita foi um grande avanço no desenvolvimento das sociedades, permitindo armazenar na forma de textos um grande volume de informação através de longos períodos de tempo [QUEIROZ, 2005]. Atualmente textos são encontrados em artigos de jornais, revistas, e-mails, relatórios empresariais, páginas de internet, prontuários médicos, anotações, etc. O que caracteriza textos como uma importante fonte de informação disponível em formato não estruturado e escrito em linguagem humana. Para que a informação desses textos seja utilizada por processos computacionais, ela deve ser estruturada de forma a ganhar semântica [Moens, 2006].

O processo de Extração de Informação (EI) consiste em identificar blocos específicos de informação contidos em fontes não estruturadas como textos em linguagem natural, e então representar essa informação de uma forma estruturada bem como adicionar semântica a cada bloco de informação relevante, permitindo a utilização dessa informação por outras atividades de processamento de informações [Sarawagi, 2008]. Após ser estruturada, a informação pode ser utilizada com facilidade por outros processos computacionais que realizam pesquisas sobre essa informação, já que o processo de EI faz com que as palavras ou sentenças passem de simples sequências de caracteres (SCs) para representações de elementos e seus inter-relacionamentos [Moens, 2006].

O processo de EI é composto por um conjunto de subtarefas que aplicam tratamentos sobre os textos de forma a identificar a informação relevante com a finalidade de disponibilizar esta em uma estrutura pré-definida como tabelas ou anotações [Bird et al., 2009]. Um conjunto de subtarefas que constitui um sistema de EI definido por [Bird et al., 2009] consiste em *(1)* dividir o texto em frases, *(2)* dividir as frases em palavras, *(3)* definir o papel morfossintático de cada palavra na frase, *(4)* identificar as entidades nomeadas presentes no texto e *(5)* definir as relações entre as entidades nomeadas. Dessas subtarefas, [Hemdev, 2011] destaca o *Reconhecimento de Entidades Nomeadas* (NER, em inglês Named Entity Recognition) como a subtarefa principal.

A subtarefa NER identifica entidades nomeadas no texto. As Entidades Nomeadas são sequências de caracteres que representam elementos de um determinado grupo como organizações, pessoas, locais, etc. Essas entidades representam informações que respondem perguntas como ”Quem?” ou ”Onde?” relativas ao texto [Zhou e Su, 2002]. As abordagens de NER são classificadas em dois grupos: *Baseadas em aprendizado de máquina* e *baseadas em regras*. Nos últimos anos houve um foco maior em abordar o uso de aprendizado de máquina, mas uma desvantagem dessas abordagens é a necessidade de um conjunto de dados com as entidades anotadas manualmente para treinamento do algoritmo [Tissot et al., 2015, Chiticariu et al., 2013]. Após o algoritmo ser treinado, ele pode ser aplicado sobre outros dados do mesmo grupo de treinamento mas ainda não anotados, para identificar as entidades. As abordagens baseadas em regras não precisam de um conjunto de treinamento, as entidades são identificadas com base em

um conjunto de regras escritas por um especialista, o que demanda esforço contínuo para manter o conjunto de regras atualizado.

[Chiticariu et al., 2013] mostra que as abordagens baseadas em regras ainda são vantajosas dependendo do caso de estudo. No grupo de abordagens baseadas em regras, estão incluídas as abordagens de NER baseadas em dicionários, que utilizam uma lista de entradas que representam as entidades de interesse e procuram por ocorrências das mesmas nos textos. Porém se a entidade não estiver definida no dicionário a mesma será ignorada no texto, por isso é importante que o conjunto de entidades a ser reconhecido seja bem conhecido e fornecido por uma fonte confiável [Wimalasuriya e Dou, 2010].

A heterogeneidade na construção de textos de diferentes grupos (jornal, e-mail, transcrição de áudios, artigos científicos, páginas web, etc) fez com que a maioria das abordagens de NER fossem desenvolvidas para trabalhar com textos de um grupo específico [Maynard et al., 2002], o que dificulta a utilização dessas abordagens com textos de diferentes domínios devido a perda de desempenho [Nadeau e Sekine, 2007]. Outra característica de textos em linguagem natural é a presença de erros ortográficos que podem fazer com que informações relevantes não sejam extraídas.

Textos disponibilizados sem revisão estão sujeitos a erros ortográficos cometidos pelo autor, por esse não conhecer a ortografia correta ou por falta de atenção. Por exemplo, se um texto possuir a subparte ”Lavita Engenharia” que possui um erro, essa não será retornada quando utilizada busca exata, uma vez que o nome correto da organização é ”Lavitta Engenharia”. Esses erros são comuns em nomes próprios que muitas vezes não seguem as regras gramaticais e também em palavras que são estrangeirismos [van Berkel e De Smedt, 1988]. A presença de erros ortográficos em textos em linguagem natural são objeto de estudo a décadas, impulsionando a criação de diversas métricas que definem a similaridade léxica entre duas SCs [Gomaa e Fahmy, 2013]. Essas métricas definem a similaridade ou a distância entre duas SCs através de um valor numérico, permitindo definir quais SCs são mais similares.

Para realizar o NER utilizando uma abordagem baseada em dicionário e conseguir minimizar os efeitos dos erros, é realizada a Associação Aproximada entre Sequências de Caracteres (ASM, em inglês Approximate String Matching) permitindo identificar as entidades mesmo quando essas ocorrem com alguma variação ortográfica [Navarro, 2001]. A ASM consiste em encontrar ocorrências de uma determinada sequência de caracteres (SC) em uma SC de tamanho maior, permitindo a existência de erros em ambos. As buscas que utilizam ASM são divididas em dois grupos, *On-line* e *Off-line* [Ghodsi, 2009, Navarro et al., 2001]. O *On-line* contém as buscas que permitem indexar em uma estrutura de dados apenas o conjunto de SCs que estão sendo buscadas (entradas fornecidas no dicionário). No *Off-line* temos as buscas que permitem criar índices tanto para as SCs buscadas (entradas) quanto para as SCs nas quais as entradas são buscados (textos).

As buscas por ASM precisam considerar todas as entradas do dicionário com similaridade superior ao mínimo definido como limite em relação à SC lida do texto. Sendo assim uma busca por ASM mais complexa se comparada com as buscas que utilizam associação exata [Shang e Merrettal, 1996]. O tempo para uma busca por ASM ser realizada é maior quanto maior o erro aceito, isso se deve ao maior número de entradas do dicionário que precisam ser consideradas como possibilidades para associação [Navarro, 2001]. Para quantificar a similaridade entre as SCs muitas métricas foram propostas [Gomaa e Fahmy, 2013], sendo a mais difundida a Edit Distance (ED) [Levenshtein, 1966]. A ED calcula a distância entre duas SCs pelo número de operações básicas sobre caracteres (inserção, remoção e substituição) necessárias para transformar uma SC na outra.

Outro recurso utilizado para auxiliar na ASM é a utilização de algoritmos de transformação [Philips, 2000]. Esses algoritmos modificam ambas as SCs eliminando determinados erros. Os algoritmos de transformação podem alterar completamente a SC, onde o resultado pode ser formado por um conjunto de caracteres totalmente diferentes como em transformações fonéticas [Russell, 1918, Philips, 2000], ou apenas alterar uma subparte da SC original como a realizada pelo *Stemming* [Lovins, 1968]. Assim as SCs que representam a mesma entidade, mas que devido a variações ortográficas são diferentes lexicalmente, passam a ser mais similares na forma transformada [Arasu et al., 2009].

A motivação para esse trabalho vem do levantamento da necessidade de uma solução adaptativa que realize o reconhecimento de entidades nomeadas definidas em listas de entidades, sendo capaz de trabalhar com textos em linguagem natural de diferentes grupos e que podem apresentar erros ortográficos. Para o trabalho com textos de variados grupos, adaptabilidade e possibilidade de fácil customização são características necessárias à abordagem. Para localizar entidades que possuem erros ortográficos a utilização de ASM permite que o NER consiga identificar entidades que não são localizadas por associação exata mas que são relevantes ao caso de estudo, evitando assim que informações importantes sejam ignoradas no processo de EI.

1.1 Objetivo

O objetivo deste trabalho é uma abordagem de NER baseada em dicionário com a capacidade de reconhecer entidades nomeadas utilizando buscas por ASM, mesmo quando as entidades apresentam certa variação na sua ortografia.

A abordagem proposta deverá:

- Realizar NER em textos em linguagem natural, localizando nos textos ocorrências de SCs disponibilizadas em um dicionário. Para armazenar o dicionário é utilizada uma estrutura de trie adaptada da literatura, permitindo à estrutura atender aos requisitos do NER.
- Utilizar diferentes métodos de transformação de strings no dicionário e nos textos de entrada, permitindo localizar mais entidades sem a necessidade de aceitar similaridades menores durante a busca sobre a trie do dicionário.
- Disponibilizar uma solução adaptável que permita ao usuário realizar alterações de configuração e a utilização de diferentes técnicas para transformação de SCs e para o cálculo de similaridade, permitindo assim a aplicação em textos de diferentes grupos.

1.2 Organização

Este trabalho está organizado da seguinte forma. No capítulo 2 são apresentados os conceitos teóricos do processo de extração de informação e principalmente da sua subtarefa de reconhecimento de entidades nomeadas. Também são descritos os conceitos de similaridade entre SCs e o problema da ASM. No capítulo 3 é apresentado o trabalho realizado nesse projeto, ilustrando a arquitetura geral da nossa abordagem, depois com o detalhamento dos componentes de construção da trie e de realização da busca e filtragem das associações produzidas e concluído com os resultados dos experimentos. No capítulo 4 são apresentadas as conclusões e os trabalhos futuros.

Capítulo 2

Estado da arte

Nesse capítulo são apresentados os conceitos que servem de base para a abordagem desenvolvida nesse trabalho. A primeira seção apresenta os conceitos envolvidos no processo de extração de informação de textos e suas subtarefas, com destaque para a subtarefa de reconhecimento de entidades nomeadas (NER), seguido da descrição de algumas ferramentas de NER da literatura. Na segunda seção são abordados os conceitos da similaridade entre sequências de caracteres (SC), apresentando algumas métricas utilizadas para calcular esta similaridade. Após é descrita a ASM, que relaciona duas SCs aceitando que elas apresentem certo grau de diferença entre si. E por último é apresentada a estrutura de dados Árvore de prefixos, recurso utilizado para indexar listas do tipo dicionário.

2.1 Extração de informação

Textos são utilizados com frequência como forma de registrar e transmitir informação. A utilização de textos representou um importante passo na evolução da sociedade humana [QUEIROZ, 2005]. Se por milhares de anos e até hoje os textos são utilizados como forma de registrar informação, é consequência que exista uma enorme quantidade de informação disponível no formato de textos livres ou linguagem natural, como em jornais, revistas, páginas web, redes sociais, anotações, etc. Mas a extração de conhecimento dessas fontes é uma tarefa complexa devido a falta de estrutura na construção desses textos, eles não seguem uma estrutura que define a forma como as informações devem ser apresentadas [Sarawagi, 2008]. [Bird et al., 2009] fornece um bom exemplo para entender a diferença entre responder uma consulta utilizando fontes estruturadas e não estruturadas. Se a necessidade é saber qual companhia opera em uma cidade específica, é mais fácil responder a consulta se as informações estiverem estruturadas como na figura 2.1 do que se usarmos informações não estruturadas (texto livre) como representado pela figura 2.2. A dificuldade que computadores tem em trabalhar com dados de textos livres escritos em linguagem natural se deve ao fato de que existem diversos caminhos para expressar a mesma informação [Moens, 2006].

Empresa	Cidade
Suzano Papel e Celulose	Limeira
Volvo	Curitiba
Prodabel	Belo Horizonte

Figura 2.1: Dados estruturados.

A Suzano Papel e Celulose, por exemplo, tem uma planta só de extração de lignina em Limeira, no interior de São Paulo com foco tanto na área de bioenergia quanto no desenvolvimento de novos produtos para novos mercados. A Volvo iniciou nesta semana a produção de motores para geração de energia elétrica em sua fábrica na Cidade Industrial de Curitiba (CIC). Funcionários da Prodabel, Empresa de Informática e Informação de Belo Horizonte, fizeram uma manifestação por reajuste salarial.

Figura 2.2: Dados não estruturados.

Extração de Informação (EI) é a análise de documentos não estruturados, como textos em linguagem natural, com o objetivo de extrair fragmentos de informação relevantes. O processo de EI consiste em extrair informação de fontes não estruturadas e disponibilizar a informação obtida em formato específico e não ambíguo através de documentos estruturados, como conjuntos de entidades ou atributos de entidades, listas, tabelas, etc [Cunningham, 2005, Sarawagi, 2008]. Sistemas de EI analisam apenas as partes do texto que contém informações úteis ao usuário, evitando uma exaustiva análise profunda de todo o texto e contribuindo para acelerar e aumentar a precisão dessa análise [Maedche et al., 2002]. Após a informação estar estruturada em classes, fica mais fácil para outros algoritmos realizarem processamentos utilizando essa informação [Moens, 2006].

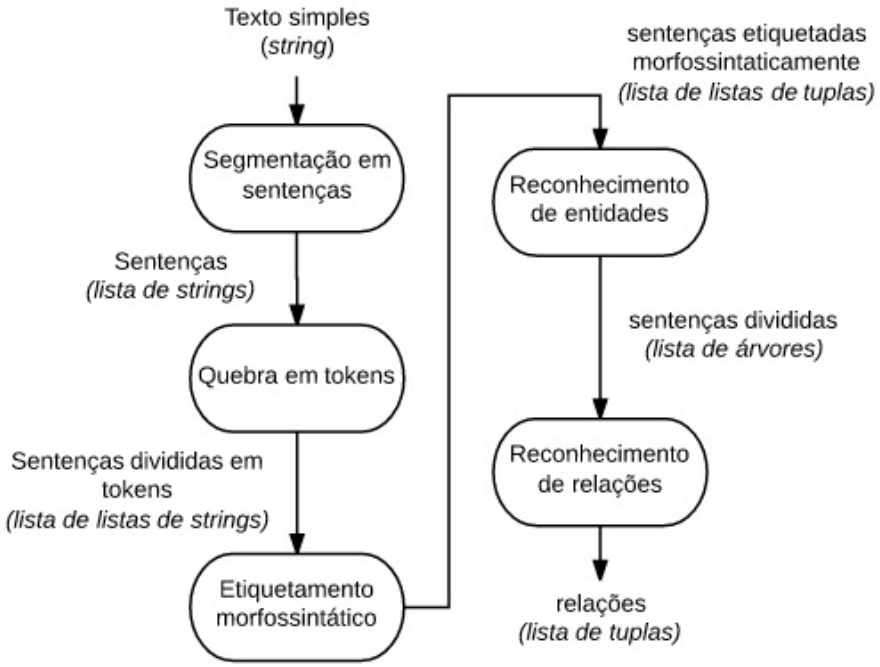

Figura 2.3: Arquitetura simples da sequência de procedimentos para um sistema de extração de informação. Traduzido de [Bird et al., 2009]

Em [Bird et al., 2009] é representada a arquitetura básica de um sistema de EI (Figura 2.3). O processo de EI submete um texto simples, sem tratamento, que foi escrito de forma livre à um conjunto de 5 procedimentos:

Texto de entrada: ”Pedro trabalha na Volvo. Ele mora em Curitiba.”

1. *Segmentação em frases*: O texto é dividido em frases. Essa fase utiliza pontuações ou outros caracteres definidos para identificar o fim das frases. Ex.: Como resultado dessa fase temos a figura 2.4, que divide a entrada em duas frases.

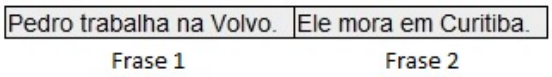

Figura 2.4: Exemplo de segmentação em frases.

2. *Quebra em tokens*: Cada frase é subdividida em palavras (tokens). Espaços em branco ou caracteres especiais como ”-” e ”&” podem ser definidos como pontos de divisão para obter os *tokens*. Ex.: A figura 2.5 ilustra o conjunto de *tokens* que é o resultado da realização dessa fase.

Figura 2.5: Exemplo de quebra em tokens.

3. *Etiquetamento morfossintático*: São aplicadas etiquetas aos *tokens* definindo o papel morfossintático (verbo, sujeito, preposição, etc) de cada *token* na frase. Ex.: Figura 2.6.

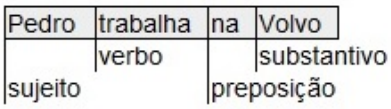

Figura 2.6: Exemplo de etiquetamento morfossintático.

4. *Reconhecimento de Entidades Nomeadas*: Localiza possíveis entidades como companhias, pessoas, locais, etc. Essa fase é ilustrada pela figura 2.7.

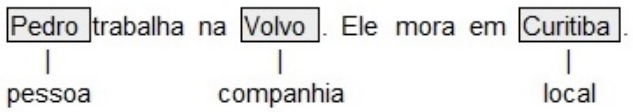

Figura 2.7: Exemplo de reconhecimento de entidades nomeadas.

5. *Reconhecimento de relações*: Define prováveis relações entre entidades encontradas no texto, retornando tuplas (entidade, relação, entidade). A figura 2.8 apresenta duas relações entre as entidades contidas no texto de exemplo.

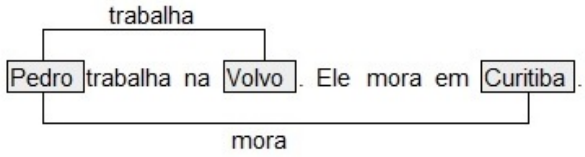

Figura 2.8: Exemplo de reconhecimento de relações.

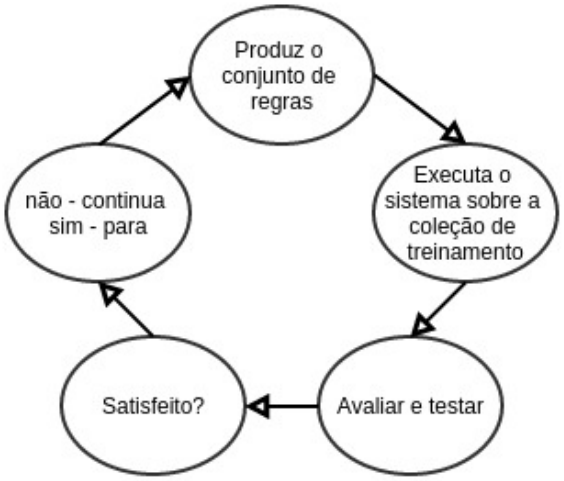

Figura 2.9: Ciclo de vida do desenvolvimento de sistema de extração de informação baseado em regras. Traduzido de [Hemdev, 2011].

Vários sistemas de EI tem sido propostos, e são classificados em dois grupos [Hemdev, 2011]: Sistemas baseados em regras e sistemas baseados em aprendizado de máquina.

O primeiro grupo, sistemas baseados em regras, engloba sistemas que trabalham sobre uma gramática criada por um programador, que normalmente está em contato com um profissional especialista no domínio abordado pelos textos e sobre os quais o sistema vai trabalhar. O engenheiro também analisa uma coleção de tamanho moderado com arquivos do domínio e usando suas habilidades cria as regras para a gramática. A figura 2.9 ilustra as etapas de cada iteração do desenvolvimento de sistemas baseados em regras. Após cada iteração as regras podem ser aprimoradas para melhorar o desempenho do sistema.

Sistemas que utilizam um arquivo de dicionário com uma lista de entradas (entidades a serem buscadas), também podem ser definidos como sistema baseado em regras, onde a regra é localizar subpartes do texto que representam às entradas informadas no dicionário. Essa será a abordagem que utilizaremos no desenvolvimento desse trabalho.

No segundo grupo, sistemas baseados em aprendizado de máquina, não é necessária a presença de um engenheiro para criar regras, um algoritmo de treinamento é executado sobre uma grande coleção de arquivos do domínio previamente anotada por um especialista do domínio [Nadeau e Sekine, 2007]. O algoritmo de treinamento retorna dados estatísticos que são utilizados quando o sistema é executado sobre novos arquivos não anotados, os dados estatísticos permitem ao sistema identificar as prováveis entidades nomeadas. Outras técnicas de aprendizado não supervisionado não fazem uso de estatísticas, mas sim de padrões presentes nos arquivos da coleção de treinamento para identificar as entidades nomeadas. [Chiticariu et al., 2013] ainda apresenta a existência de sistemas ”Híbridos”, os quais são uma composição de elementos dos dois grupos acima descritos.

2.1.1 Reconhecimento de entidades nomeadas

Reconhecimento de Entidades Nomeadas (NER) é uma subtarefa de EI que consiste na classificação de algumas SCs do texto, de forma que estas sejam anotadas identificando a palavra como sendo nomes de Pessoas, companhias, lugares, etc. [Viola, 2006]. O reconhecimento de entidades nomeadas foi criado em 1996 na sexta Conferência de Compreensão de Mensagens (MUC-6), essa conferência foi uma iniciativa da Defense Advanced Research Projects Agency (DARPA) para impulsionar pesquisas sobre a análise automática de mensagens militares com conteúdo em formato texto [Grishman e Sundheim, 1996]. Apesar de intitulada como conferência, as MUCs podem ser vistas como uma competição, pois os pesquisadores recebiam uma lista onde estava definido o domínio que deveria ser analisado, uma coleção de arquivos do domínio para os pesquisadores usarem para teste e as atividades que deveriam ser atendidas pelas ferramentas apresentadas na conferência. Na definição das atividades para o MUC-6, o reconhecimento de entidades nomeadas foi definido como uma subtarefa do processo de EI, uma vez que foi percebido que seria importante identificar ocorrências de nomes de pessoas, organizações e locais e também identificar expressões numéricas de tempo, data, moeda e porcentagem [Nadeau e Sekine, 2007].

De acordo com [Viola, 2006], NER pode ser definido da seguinte forma. "Um documento D é uma sequência de palavras *tokens* $\mathbf{x}=\langle x_1 \ldots x_n \rangle$. O objetivo do NER é extrair de D um conjunto de campos $\mathbf{F} = \{F_1 \ldots F_k\}$, onde cada campo é um par atributo-valor, $F_i = \langle a, v \rangle$ (por exemplo $F_i = \langle Cidade, Curitiba \rangle$). O valor do campo pode ser repetido para múltiplos *tokens*."

A seguir, um exemplo do funcionamento da tarefa de reconhecimento de entidades nomeadas.

"João trabalha na Volvo em Curitiba."

No exemplo acima são encontradas três entidades, João é uma pessoa, Volvo é uma companhia e Curitiba é um local. [Viola, 2006] utiliza rótulos como *nome, companhia, cidade, número de telefone, etc.*. Utilizando esse modelo, a saída para a frase acima seria a seguinte:

"<pessoa,João> trabalha na <companhia,Volvo> em <cidade,Curitiba>."

As anotações têm a finalidade de identificar e classificar as entidades nomeadas, definindo às mesmas uma semântica que pode ser facilmente utilizada por outros processos computacionais.

Para avaliar as abordagens de NER são utilizados valores de *precisão*, *cobertura* e *F-score* [Goutte e Gaussier, 2005]. A *precisão* corresponde ao percentual de entidades nomeadas reconhecidas e que estão corretas (relevantes) em relação ao total de entidades nomeadas que foram reconhecidas. A *cobertura* é definida pelo percentual que o conjunto de entidades nomeadas corretamente reconhecidas representa do total de entidades nomeadas presente nos dados analisados. Ex.: Se em um texto existem 100 entidades, e em uma busca foram retornadas 80 entidades e todas corretas, a precisão é 100% e a cobertura é 80%. Se outra busca retornar 150 entidades, sendo 100 corretas e 50 erradas, a precisão é 66,66% e a cobertura é 100%. O *F-Score* mede a efetividade do sistema de NER baseado nos valores de precisão e cobertura, sendo possível priorizar um ou outro valor. Nesse trabalho será utilizado o F1-score que é a média harmônica entre a precisão e a cobertura e calculada pela fórmula:

$F1 = 2 \cdot \frac{Preciso \cdot Cobertura}{Preciso + Cobertura}$

Para [Nadeau e Sekine, 2007], os sistemas de NER podem ser agrupados em dois conjuntos, um conjunto com os sistemas baseados em regras e o outro com sistemas de aprendizado de máquina. Os grupos são os mesmos utilizados para agrupar os sistemas de extração de informação relatados anteriormente. Isso não é uma coincidência, em [Hemdev, 2011] a subtarefa NER é definida como o "coração" do sistema de EI, e devido a essa importância, a técnica utilizada para na subtarefa NER define a classificação de todo o sistema de EI.

Os sistemas baseados em aprendizado de máquina são divididos em três grupos [Nadeau e Sekine, 2007], Aprendizado supervisionado, aprendizado semi-supervisionado e

aprendizado não supervisionado. Aprendizado supervisionado é caracterizado por sistemas treinados sobre uma base em que as entidades estão identificadas, gerando assim uma lista de entidades e utiliza características descritivas para evitar ambiguidades na classificação. Para os sistemas de aprendizado semi-supervisionado, é necessário identificar apenas alguns exemplos das entidades que se deseja na base de treinamento e então o sistema roda sobre umas frases e procura por ocorrências das entidades identificadas, e após analisa o contexto onde elas se encontram. O sistema aprende o contexto e então procura por ocorrências de outras entidades em um mesmo contexto, e então aprende novamente utilizando as novas entidades encontradas. O grupo dos sistemas de aprendizado não supervisionado tem como principal característica não necessitarem de dados anotados para treinamento. Eles utilizam agrupamento feito por funções que definem as características utilizadas para o agrupamento, e usam estatísticas sobre um grande conjunto de dados não anotados.

Ainda em [Nadeau e Sekine, 2007], os sistemas baseados em regras utilizam um conjunto de características que são tratadas por regras e dependendo da regra que selecionar a palavra, a palavra será definida como uma entidade de uma determinada classe. Alguns exemplos de características são possuir letra maiúscula, o tamanho da palavra em caracteres e a versão da palavra toda em minúsculo. Então as regras são definidas para utilizarem uma ou mais dessas características e definirem uma classificação para determinado conjunto de valores. Por exemplo, para a frase ”Pedro vai viajar amanhã” definimos a regra: se a palavra tiver letra maiúscula e tamanho maior que 3 é pessoa. As características ainda podem ser do documento, como: Frequência de ocorrência da palavra, a posição da palavra no texto, informações de metadados (ocorreu no campo ”assunto” de um e-mail).

Outros sistemas que são classificados entre os sistemas de regras utilizam listas fornecidas através de dicionário [Nadeau e Sekine, 2007]. O dicionário pode conter a própria entidade como ”Curitiba, Belo Horizonte, Porto Alegre” para uma lista utilizada para anotar cidades. Ou palavras a serem utilizadas na classificação, como títulos (DR.) ou palavras comumente utilizadas próximo a entidades como ”é uma”. A utilização de listas de entidades é a técnica que será explorada nesse trabalho e é detalhada na sequência.

2.1.2 Reconhecimento de entidades nomeadas baseado em dicionário

Uma técnica amplamente utilizada no desenvolvimento de sistemas de reconhecimento de entidades nomeadas utiliza listas conhecidas como *dicionários* [Wimalasuriya e Dou, 2010]. Essas listas contêm a relação de entidades que devem ser buscadas nos textos. Os sistemas baseados nessa técnica recebem um ou mais arquivos de dicionário que contém listas com SCs chamadas ”entradas” e que devem ser reconhecidas nos textos. Cada entrada está ligada a rótulos que são adicionados às SCs do texto que forem relacionadas com a entrada, assim cada SC do texto pode ser classificada de acordo com os rótulos que recebe. Esses sistemas reconhecem SCs individuais como estados de um país, companhias, moedas adotadas no mundo, etc.

Assim podemos definir a técnica de NER baseado em listas de entidades como: Dado um texto T e um conjunto de entradas E pertencentes a um dicionário D, sendo D e T construídos utilizando um mesmo alfabeto Z, localizar em T as ocorrências de cada elemento de E. A figura 2.10 representa a aplicação dessa técnica, através de um dicionário que contém nomes de cidades e um texto no qual estão destacas por um retângulo vermelho as entidades que foram localizadas com base na relação de entidades fornecida pelo dicionário.

Essa técnica é bastante eficaz na busca por entidades que podem ser precisamente especificadas, sendo necessária uma fonte confiável, completa e que preferencialmente não precise nenhum pré-processamento. Exemplos dessa lista são a de agências reguladoras do Brasil

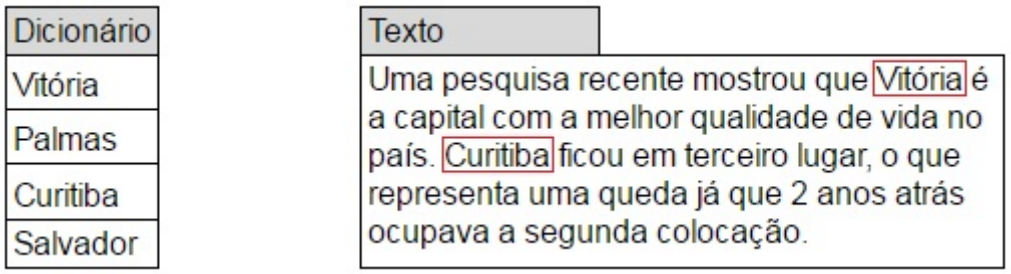

Figura 2.10: Exemplo de NER baseado em listas.

ou os nomes das cidades de um estado ou país. Uma vantagem da utilização dessa técnica de NER em relação a utilização de aprendizado de máquina é não precisar de um conjunto de dados pré-anotados para treinamento. Porém se a entidade não estiver definida na relação de entradas fornecidas pelo dicionário, ela não será identificada mesmo estando presente no texto analisado.

Outro ponto que deve ser observado para a escolha de abordagens de NER baseados em listas é a existência de erros ortográficos nas entidades presentes no texto, fazendo com que a busca por associação exata não recupere todas as entidades do texto [Navarro, 2001]. Para proporcionar aos sistemas a capacidade de trabalhar com esses erros é utilizado o conceito de associação inexata, permitindo a identificação das entidades mesmo com diferenças ortográficas entre a entrada definida no dicionário e a subparte do texto. A associação inexata e o uso de métricas para calcular a similaridade entre SCs são explicados na seção 2.2.

2.1.3 Sistemas de reconhecimento de entidades nomeadas

Nessa seção são apresentados alguns sistemas de NER encontrados na literatura. Muitos outros existem, como levantado em [Chiticariu et al., 2013] que identificou 177 artigos científicos apresentados entre 2003 e 2012 sobre extração de entidades. Os sistemas escolhidos são listados na tabela 2.1, esse pequeno conjunto de sistemas é capaz de ilustrar as diferentes abordagens que podem ser utilizadas (*Machine learning* - ML e Regras - R), as diversas áreas em que o NER pode ser aplicado e o constante desenvolvimento ano após ano. também são apresentadas informações sobre a utilização de Associação Aproximada (coluna AA) e se a abordagem faz uso de dicionário com entradas para identificar as entidades (coluna ABD). Valor ”nd” significa que a informação não está disponível. Na coluna ”Abordagem”, os valores definem se o sistema utiliza a abordagem de aprendizado de máquina (ML) ou baseada em regras (R).

Abaixo vamos descrever brevemente cada um dos sistemas citados na Tabela 2.1.

BANNER apresentado por Leaman [Leaman et al., 2008], é um sistema para reconhecimento de entidades nomeadas na área biomédica que tem código aberto e que é implementado utilizando a técnica de aprendizado de máquina. A estrutura do sistema é uma linha de 3 estágios, o primeiro estágio recebe um texto simples como entrada e é responsável por dividir as frases em *tokens* simples (palavras), no segundo estágio cada *token* é convertido em uma *rótulo* (par nome/valor) as quais são usadas pelo algoritmo de aprendizado. O conjunto de *rótulos* é então aplicado e a saída é o texto rotulado.

MUSE é um projeto apresentado por Maynard [Maynard et al., 2003] com objetivo de disponibilizar um recurso capaz de realizar, com mínima adaptação, o reconhecimento de entidades nomeadas em textos de diferentes tipos. É um sistema baseado em regras e tem como base o ANNIE, um sistema de extração de informação que por padrão acompanha o GATE [Cunningham et al., 2002]. De forma bastante simplificada, a arquitetura do sistema é uma execução sequencial de recursos de processamento sobre um conjunto de documentos. Os

Tabela 2.1: Relação de sistemas de NER.

Autor	Sistema	Ano	área de atuação	AA	ABD	Abordagem
Leaman	BANNER	2008	Biomedicina	nd	não	ML
Maynard	MUSE	2003	diversos	nd	sim	R
Rocktäschel	ChemSpot	2012	Química	nd	sim	R & ML
Hakenberg	GNAT	2002	Biomedicina	nd	não	R & ML
Boulaknadel	nd	2014	Idioma Amazighe	nd	sim	R
Li	TwiNER	2012	Twitter	nd	não	ML
Wang	nd	2009	Medicina	nd	não	ML
Ritter	T-ner	2011	Twitter	nd	não	ML
Al-Jumaily	nd	2012	Idioma Árabe	nd	sim	R
Navarro	Matchsimile	2003	Nomes próprios	sim	sim	R

recursos de processamento são baseados principalmente em algoritmos de estado finito. Os recursos de processamento executam tarefas como quebra em *tokens*, comparação com padrões de listas, resolução de ambiguidades entre entidades, etc.

ChemSpot [Rocktäschel et al., 2012] é uma abordagem híbrida desenvolvida para realizar NER no domínio biomédico, mais especificamente da química. Ela é definida como híbrida por utilizar duas técnicas para realizar a anotação do texto, aprendizado de máquina através de *conditional random field* (CRF) e associação com dicionário. Cada uma das técnicas anteriores realiza a anotação do texto de entrada independentemente uma da outra. Após, as anotações de ambas são fundidas e mais alguns tratamentos são aplicados, como selecionar a anotação vai prevalecer para as situações em que ambas anotaram a mesma entidade.

GNAT [Hakenberg et al., 2011] consiste de um conjunto de módulos para processamento de texto. A execução do GNAT pode ser dividida em 7 passos, (1) Recuperar os documentos, (2) pré-processar cada texto, (3) Realizar o NER para genes e espécies, (4) Remover falsos positivos para menções a genes, (5) Atribuir identificadores candidatos para genes, (6) Validar os identificadores e (7) Classificar os candidatos a identificadores de gene. No passo 3, o NER para genes e espécies é feito utilizando a abordagem baseada em dicionário, nesse passo um conjunto de identificadores candidatos é atribuído a cada menção de gene. Ainda para reconhecer candidatos a nomes de genes, é fornecida uma interface para o sistema BANNER (descrito acima), pode-se usar os resultados de ambos juntos ou de apenas um dos métodos. No reconhecimento de nomes de espécies, é utilizado o Linnaeus [Gerner et al., 2010].

A abordagem proposta por Boulaknadel [Boulaknadel et al., 2014], é baseada em regras e desenvolvida sobre o framework GATE, e tem por objetivo identificar e categorizar entidades nomeadas como ”pessoa”, ”companhia”, ”local”, ”data” e ”números” no idioma Amazighe (idioma utilizado pelos povos berberes, que vivem sobretudo no norte da África). Essa abordagem apresenta dois recursos principais na sua arquitetura, um procedimento de pesquisa onde os padrões buscados são fornecidos por listas (listas com entidades nomeadas conhecidas) e o segundo chamado Grammar, que é baseado em um conjunto de regras gramaticais.

TwiNER, proposto por Li [Li et al., 2012], é um sistema de NER não supervisionado que não faz uso de características linguísticas, mas sim do contexto local (postagens) e de um contexto global (World Wide Web) com objetivo de realizar NER em postagens feitas no Twitter. O sistema não classifica as entidades nomeadas (pessoa, local, companhia, etc.), apenas encontra candidatos a entidades nomeadas nas postagens. A arquitetura é composta de dois passos, o primeiro que faz a segmentação das postagens e que tem como saída um conjunto de

segmentos que são considerados candidatos a entidades nomeadas, Esses candidatos são enviados ao segundo passo, Cada candidato é rotulado com uma pontuação que indica o quanto aquele candidato é considerado uma entidade nomeada, então utilizando essa pontuação os candidatos são classificados em uma lista.

O recurso proposto por Wang [Wang e Patrick, 2009], tem o objetivo de encontrar entidades em anotações clínicas. É um sistema baseado em aprendizado de máquina que utiliza o conceito de cascata para executar três modelos de aprendizado de máquina (Conditional Random Fields, Support Vector Machine e Maximum Entropy). Os diferentes modelos são utilizados para reclassificar as entidades e se os três modelos apresentarem saídas diferentes, um reclassificador define baseado nas probabilidades de cada modelo qual será a saída do sistema.

O *T-ner*, proposto em [Ritter et al., 2011] é uma abordagem baseada em aprendizado de máquina para realizar o reconhecimento de entidades em postagens no Twitter. São realizadas três tarefas: POS tagging, Chunking e NER. As duas primeiras auxiliam na realização do NER. A tarefa de NER foi dividida em duas partes, uma que realiza a segmentação das entidades, definido se uma palavra é ou não uma entidade. A outra faz a classificação das entidades, definindo se a entidade é uma pessoa, companhia, local, etc.

Al-Jumaily propôs um sistema para realizar NER em textos escritos no idioma Árabe [Al-Jumaily et al., 2012]. O sistema faz uso de recursos de vários sistemas de processamento de linguagem natural existentes (GATE, ANERGazet, DBpedia entre outros). Os processos realizados pelo sistema são de dois tipos, *Off-line* e *online*. Os Off-line são responsáveis por criar três dicionários, um com prefixos, um dicionário morfológico e o terceiro com entidades nomeadas. Os processos Online são responsáveis por receber o texto de entrada, realizar o NER com base nos dicionários produzidos nos processos Off-line, e prouzir dois arquivos de saída, um com os *tokens* não anotados (não são entidades nomeadas) e outro no formato XML com as entidades que foram anotadas. Cada *token* extraído do texto pode ser associado a nenhum, um ou vários padrões presentes nos dicionários, se nenhum padrão dos dicionários for associado ao *token*, o *token* é gravado no arquivo de saída que contém *tokens* não anotados, se algum padrão for retornado para o *token*, verificações serão realizadas sobre esses padrões para definir se o *token* é realmente uma entidade nomeada e qual é o melhor padrão para ser associado ao *token*. Após, a saída é convertida em formato XML e gravada no outro arquivo de saída, que contém os *tokens* anotados.

Matchsimile é um recurso proposto em [Navarro et al., 2003] que utiliza uma árvore de prefixo como índice de um conjunto de entidades a serem identificadas em textos. O sistema também possui recursos para trabalhar com abreviações e diferentes formas de escrita de nomes próprios, como o segundo nome antes do primeiro. O trabalho mostra que a busca sobre a árvore é a atividade de maior custo, chegando a representar 85% do tempo total da aplicação. O sistema sofre com um problema comum a sistemas baseados em dicionário, palavras curtas e erros altos para palavras curtas, o que foi contornado ignorando palavras com 1 e 2 letras, e aplicando um erro máximo crescente de acordo com o tamanho da palavra, atingindo o ED máximo de 4 para palavras com mais de 12 letras.

Como descrito por [Maynard et al., 2002] e observado nos trabalhos acima apresentados, a maioria das abordagens de NER são desenvolvidas especificamente para estudo de um domínio. Essa característica dificulta a utilização de uma mesma abordagem para casos de estudo diferentes. Isso faz com que novas abordagens precisem ser desenvolvidas para cada conjunto de texto de um domínio novo ou com uma estrutura de escrita diferente de um grupo que já tem uma abordagem de NER em uso.

Tabela 2.2: Exemplos de valores de Edit Distance.

Palavra 1	Palavra 2	ED
paralizado	paralisado	1
cincoenta	cinquenta	2
pensar	pesar	1
xícara	chácara	3

2.2 Similaridade entre sequências de caracteres

A similaridade entre SCs é o resultado da aplicação de uma métrica de similaridade sobre duas SCs. Uma métrica para SC mede a similaridade (distância) entre duas SCs para definir correspondência aproximada entre elas ou comparação. A similaridade entre palavras pode ser dividida em dois grupos, léxica ou semântica. Similaridade léxica consiste em comparar a SC da palavra tendo como ponto de atenção a ordem dos caracteres que a compõe. Na similaridade semântica as palavras são comparadas verificando se elas têm o mesmo significado, se são usadas da mesma forma, no mesmo contexto e se uma representa variação da outra [Gomaa e Fahmy, 2013].

Nesse trabalho será utilizada a similaridade léxica para realizar a associação aproximada entre palavras. A similaridade léxica pode ser definida como, dado uma SC $s1$, uma SC $s2$ e uma métrica M, temos que a similaridade entre $s1$ e $s2$ é $M(s1, s2) = similaridade$. A similaridade apresentada pelas métricas é geralmente normalizada para valores entre 0 e 1, onde zero é nenhuma similaridade e 1 indica SCs iguais ($0 \leq similaridade \leq 1$) [Navarro et al., 2001].

Os erros léxicos foram divididos por [van Berkel e De Smedt, 1988] em dois grupos: *erros de ortografia* e *erros de escrita*. Os *erros de ortografia* representam os erros em que o autor não tem conhecimento da ortografia correta da palavra ou a esqueceu. Sendo assim a sequência escrita foi a sequência que o autor desejava, mas que não condiz com a forma correta de escrita. Esse tipo de erro é comumente encontrado em nomes próprios, palavras pouco utilizadas e palavras estrangeiras. Uma característica desse erro é que a palavra resultante (com erro) é um fonema muito similar à palavra correta (Ex: ”exceção” e (excessão)). Já os *erros de escrita* são aqueles resultantes de uma entrada incorreta na sequência de caracteres por algum motivo não relacionado ao idioma ou conhecimentos ortográficos do autor. Como pressionar uma tecla errada durante a digitação, ou seja, são ocasionados por falta de atenção. Esses erros são comuns em palavras com duas letras iguais em sequência, sendo que uma delas é omitida ou então uma letra é duplicada incorretamente.

Uma das mais difundidas métrica para similaridade entre SCs proposta é a *Levenshtein Distance* ou *Edit Distance* (ED) [Levenshtein, 1966], apesar do *Edit Distance* possuir diferentes definições, sendo que cada uma considera diferentes operações sobre as SCs comparadas, a *Levenshtein Distance* é a mais comum, sendo sua definição tida como a definição padrão de *Edit Distance* [Navarro, 2001], e será essa a métrica que utilizaremos para realizar ASM no desenvolvimento desse trabalho.

ED mensura a distância entre duas SCs, s_1 e s_2, pelo número mínimo de operações sobre caracteres individuais (inserção, remoção ou substituição) necessárias para transformar uma SC na outra. Na tabela 2.2 temos exemplos que mostram valores de ED entre palavras diferentes mas que ambas estão corretas (duas primeiras linhas) e também o ED entre a palavra com a ortografia correta e a mesma palavra mas com erros ortográficos (duas últimas linhas).

A representação matemática da ED para as duas SCs acima (s_1 e s_2), de comprimento $|s_1|$ e $|s_2|$, é dada por $lev_{s_1,s_2}(|s_1|, |s_2|)$ onde

$$lev_{s_1,s_2}(i,j) = \begin{cases} max(i,j) & \text{se } \min(i,j) = 0, \\ min = \begin{cases} lev_{s_1,s_2}(i-1,j)+1 \\ lev_{s_1,s_2}(i,j-1)+1 \\ lev_{s_1,s_2}(i-1,j-1)+1_{(s_{1_i} \neq s_{2_j})} \end{cases} & \text{senão} \end{cases}$$

onde $1_{(s_{1_i} \neq s_{2_j})}$ é a função de indicação igual a 0 (zero) quando $1_{(s_{1_i} = s_{2_j})}$ e igual a 1 nos outros casos. E $lev_{s_1,s_2}(i,j)$ é a distância entre os i primeiros caracteres de s_1 e os j primeiros caracteres de s_2.

Porém o uso de ED puro pode não ser adequado para tratar toda a gama de erros possíveis. Um mesmo valor de ED tem impacto diferente dependendo do tamanho das palavras comparadas, um ED=2 para uma palavra de tamanho 3 é mais impactante do que um ED=2 para uma palavra de tamanho 9, por alterar uma porcentagem maior da palavra. Para contornar situações como essa são necessários algoritmos que apresentam a similaridade entre palavras de forma normalizada, com valores entre 0 e 1 [Marzal e Vidal, 1993].

Abaixo são apresentados conceitos de outras métricas utilizadas para medir a distância entre SCs, e que consideram outros atributos das SCs comparadas além do número de operações sobre caracteres como faz o ED. Mas, ainda assim, todas baseadas no conjunto de caracteres que formam as SCs (similaridade léxica):

- *Longes Common SubString* (LCS): O problema dos algoritmos de LCS consiste em encontrar uma subsequência comum entre duas SCs, sendo a similaridade entre as SCs dada pelo comprimento máximo da subsequência [Bergroth et al., 2000].

- *Jaro-Winkler*: Na comparação, o algoritmo básico de Jaro-Winkler conta o número de caracteres comuns entre duas SCs e o número de transposições entre esses caracteres comuns [Yancey, 2005].

- *N-gram*: Consiste na quebra em subpartes das SCs a serem comparadas, o comprimento das subpartes da SC é igual a n, assim 2-gram significa que cada subparte terá um comprimento igual a 2. Então cada n-gram das duas SCs são comparados e a similaridade é dada pela divisão do número de n-grams similares pelo número máximo de n-grams [Barron-Cedeno et al., 2010].

- $String_{Sim}$: Algoritmo apresentado por [Tissot et al., 2014], que além de comparar os caracteres de mesma posição nas duas SCs, leva em consideração se um determinado caractere existe na outra sequência mesmo que em uma posição diferente, se existir a penalidade é menor do que para caracteres não existentes. Também considera a distância que esse caractere está da posição correta, quanto mais distante menor é a similaridade. Outro ponto analisado é o comprimento de ambas as SCs, quanto maior é a diferença de tamanho, menor será a similaridade.

Além das métricas que calculam a similaridade entre duas SCs analisando seus caracteres, existem algoritmos desenvolvidos para interagir com as SCs com o objetivo de alterar as mesmas para obter melhores condições para a comparação. Alguns exemplos são as conversões fonéticas e a remoção de partes ou troca da palavra por sua versão básica. Como opções de conversão fonética podem ser utilizados entre outros o Soundex e Double metaphone. Soundex [Russell, 1918] é um algoritmo fonético que representa palavras diferentes mas que tem o mesmo fonema, com a mesma sequência de caracteres. O algoritmo considera principalmente as consoantes, sendo que uma vogal só codificada apenas se for o caractere inicial da palavra. Isso permite associar palavras com diferenças na ortografia.

O Soundex utiliza um conjunto formado por uma letra seguida por três números para codificar uma palavra, exemplos da codificação são apresentados na tabela 2.3. O Soundex é utilizado como sinônimo de conversão fonética e serviu de base para outros algoritmos fonéticos,

Tabela 2.3: Exemplos de transformação de palavras utilizando Soundex.

Palavra	Codificação
Curitiba	C631
Washington	W252
Amoxicilina	A522
Hello	H400
Key	K000

Tabela 2.4: Exemplos de transformação de palavras utilizando Double Metaphone.

Palavra	Codificação 1	Codificação 2
Curitiba	KRTP	KRTP
Washington	AXNK	FXNK
Amoxicilina	AMKS	AMKS
Hello	HL	HL
Key	K	K

como o Double metaphone. O Double metaphone [Philips, 2000] é a uma evolução do Soundex, ele propõe melhorias em relação a codificação do Soundex considerando variações na pronuncia em inglês das palavras e características da ortografia em outros idiomas.

O Double metaphone analisa um conjunto maior de características para definir a codificação que vai representar cada palavra, mas assim como no Soundex o objetivo é representar palavras com pronuncia parecida através de um mesmo conjunto de caracteres (codificação). O termo ”Double” presente no nome do algoritmo refere a capacidade deste em produzir duas codificações fonéticas para uma mesma palavra. Na tabela 2.4 são apresentados exemplos de transformações realizadas seguindo as regras do Double Metaphone. As duas codificações produzidas podem ser iguais ou apresentarem diferenças em qualquer dos caracteres. A quantidade de caracteres utilizados em cada codificação pode variar, mas nunca superior a 4.

Outros algoritmos de transformação realizam transformações nas SCs considerando outras características que não a fonética. O ”Stemming” é o processo de redução de uma palavra flexionada ou derivada para sua forma básica, mas não precisa ser a palavra base exatamente, podendo ser apenas a parte principal da palavra. Vários algoritmos utilizam o stemming, sendo [Lovins, 1968] o primeiro algoritmo apresentado com esse recurso. Por exemplo, a palavra ”conectado” pode ser alterada para ”conect”, que é a parte central para outras variações como ”conectar” e ”conectou”. Assim a comparação entre as palavras considera apenas a parte central da palavra. Outra técnica para alterar a sequência de caracteres é a ”Lemmatization” [Kanis e Skorkovská, 2010], que assim como o Stemming procura transformar as palavras flexionadas para suas formas básicas, porém diferente do Stemming a Lemmatization não apenas remove parte da SC, ele troca uma palavra pela versão básica da mesma presente no dicionário. No exemplo utilizado no Stemming, o resultado foi uma palavra que não faz parte do idioma português. Para o mesmo exemplo, a Lemmatization transformaria a palavra para ”conecta”, que é a palavra base existente no idioma.

A utilização de recursos de transformação permite alterar as SCs de forma a reduzir ou até eliminar erros ou variações, permitindo que associações entre diferentes SCs sejam realizadas com menores valores para o erro máximo aceito, se comparado com as associações considerando as SCs originais. Nesse trabalho o algoritmo de transformação utilizado será o

Double Metaphone, devido sua capacidade de compressão das SCs transformadas e também por conseguir eliminar alguns erros fonéticos.

2.2.1 Associação aproximada entre sequências de caracteres

O problema da Associação Aproximada entre Sequências de caracteres (ASM) aborda sobre a busca por ocorrências aproximadas de um padrão em um texto, quando são aceitos erros no padrão bem como no texto [Navarro et al., 2001]. Uma definição formal do problema de ASM feita por [Navarro et al., 2001] é:

- Dado uma pequena sequência como padrão P de comprimento m e um texto T mais longo de comprimento n, ambas formadas sobre um mesmo alfabeto Σ de tamanho σ. E aceitando uma quantidade de erros κ. Recuperar as posições para T que podem ser associadas com P com o máximo de κ erros, $erro(T_{x...y}, P) \leq \kappa$, para $(0 < x \leq y)$ e $(x \leq y \leq n)$. Em [Peterson, 1986], é apresentado que de 81% a 90% dos erros de digitação são causados por uma letra extra adicionada à palavra, uma letra faltando na palavra ou por uma letra errada na palavra (ex.: a letra ”b” trocada pela letra ”p”), sendo os erros por uma letra os mais comuns e os por duas letras vindo em segundo lugar.

Uma vez que a métrica que vai calcular a similaridade foi definida, é preciso realizar a busca por ocorrências no texto das entradas fornecidas pelo dicionário. Os sistemas de busca podem ser divididos em duas versões, *on-line* e *off-line* [Ghodsi, 2009, Navarro et al., 2001]. Na versão *on-line*, é permitido pré-processar apenas o padrão. Assim, não é permitido criar um índice do texto antes de executar a busca. O desempenho de algoritmos *on-line* não é satisfatório quando executados sobre grandes textos, apesar de existir algoritmos rápidos. Algoritmos *off-line* permitem criar uma estrutura de dados sobre o texto e usar isto para aumentar a velocidade das buscas. Algumas estruturas de indexação bem conhecidas que são usadas nesses casos são suffix tree, suffix array, q-grams e q-samples. Todas elas apresentam uma estrutura com foco em buscas por aproximação.

2.2.2 Árvores de prefixo - Trie

Trie é uma estrutura de dados que utiliza uma árvore de prefixos para a indexação de SCs. Caracterizada por fazer uso compartilhado de uma mesma estrutura para representar um prefixo comum às diversas sequências indexadas, o que faz com que estas árvores também sejam chamadas de árvore de prefixo [Shang e Merrettal, 1996]. Outra característica é o fato de que os nós são rotulados por caracteres pertencentes ao alfabeto, sendo que cada nível da árvore representa um caractere da sequência, sendo assim o caminho da raiz (nó que representa sequência vazia) até determinado nó folha contém uma sequência indexada. Ao utilizar a mesma estrutura para indexar sequências que possuem um prefixo em comum, tries conseguem economizar espaço se comparadas com árvores de busca binária (binary search tree). [Oommen e Badr, 2007] ressalta que tries podem ser utilizadas para armazenar dicionários que são formados por conjuntos de palavras e informa também que o custo da busca sobre tries é independente do tamanho do arquivo armazenado. A figura 2.11 apresenta um exemplo de trie.

[Shang e Merrettal, 1996, Pradhan e Negi, 2014, Oommen e Badr, 2004, Oommen e Badr, 2007, Al-Turaiki et al., 2012] são exemplos de trabalhos que aplicam o conceito de trie ou uma variação dele para realizar ASM ou busca exata (quando as duas SCs comparadas devem ser idênticas). Uma das formas que trie é utilizada em sistemas de ASM é para indexar as SCs (entradas) fornecidas pelo dicionário e que se deseja buscar em um texto onde essas SCs podem conter erros [Pradhan e Negi, 2014]. Por todas as características

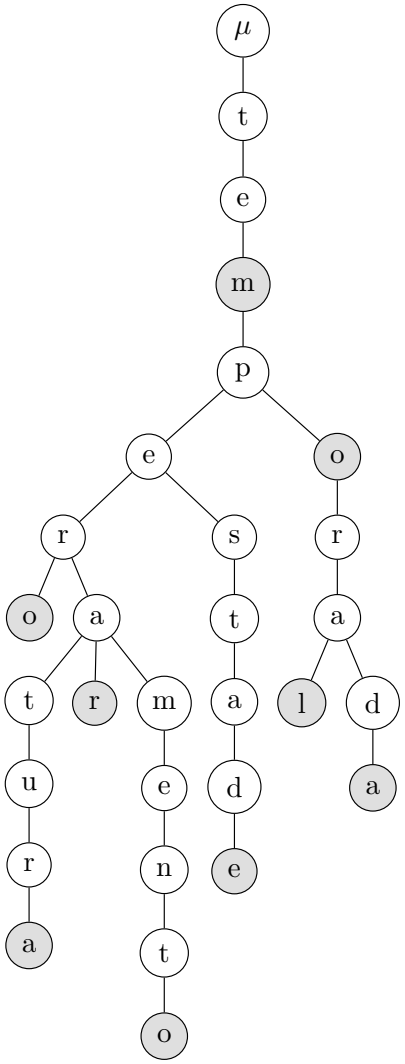

Figura 2.11: Árvore trie. Representa uma árvore trie indexando um dicionário com as palavras (tem, tempo, tempero, temporal, temperatura, temperar, tempestade, temperamento, temporada). O nó μ representa a raiz, e os nós em cinza representam finais de palavras.

acima apresentadas, trie é a estrutura escolhida para armazenar o dicionário que é utilizado na abordagem de NER proposta nesse trabalho.

2.3 Análise da literatura

Um percentual significativo da informação produzida em todo o mundo é disponibilizado na forma de textos em linguagem natural, esses textos são definidos como fontes não estruturadas de informação, sendo necessária a criação de ferramentas capazes de auxiliar na extração da informação relevante a um determinado interesse e sua disponibilização em uma estrutura pré-definida, com a qual sistemas computacionais (sistemas de mineração de dados e de suporte a decisão) consigam trabalhar automaticamente. A extração de informação (EI) é a área que trabalha no desenvolvimento de recursos capazes de realizar essa extração. Perguntas do tipo ”Quem?”, ”Onde?”, ”Quando?” é que queremos que sejam respondidas automaticamente através do processamento de textos.

A heterogeneidade (diferentes domínios, idiomas, gêneros, etc) que existe entre os textos, levou ao desenvolvimento de diferentes sistemas de EI que utilizam variadas técnicas e constantemente novos sistemas são apresentados. O NER é uma importante subtarefa do processo de EI, e é responsável por identificar entidades (pessoas, companhias, locais, moedas, etc.) presentes em textos. Porém analisando o conjunto de ferramentas de NER, é verificado que com poucas exceções, cada ferramenta trabalha com um conjunto de arquivos de um domínio bastante restrito, sem a intenção de ser flexível ao ponto de ser utilizada para tratar textos de grupos diferentes.

Apesar de nos últimos anos a abordagem de aprendizado de máquina estar em foco no desenvolvimento de sistemas de NER, a abordagem baseada em regras como a utilização de dicionário de entidades, ainda é uma boa escolha dependendo das características dos textos analisados e das entidades que devem ser anotadas. E não é só o desempenho que precisa ser considerado, mas também o esforço para desenvolver e manter o sistema, o que depende da abordagem utilizada. Muitos domínios já possuem sistemas de NER que atuam sobre eles, mas ainda há espaço para expansão, e acreditamos que essa expansão poderia ser facilitada se mais sistemas de NER forem projetados de forma a serem adaptáveis.

Outra característica de textos em linguagem natural que deve ser considerada para o desenvolvimento de abordagens de NER são os erros ortográficos. Abordagens de NER baseadas em dicionário e que realizam associação exata podem ter seu desempenho prejudicado quando aplicadas sobre textos que apresentam esses erros. Esse impacto no desempenho é maior em sistemas que utilizam arquivo de dicionário pois estes trabalham diretamente sobre a sequência dos caracteres e não com a semântica das palavras ou a estrutura do texto. Assim é importante desenvolver soluções capazes de trabalhar com textos superando erros ortográficos. Várias métricas para medir a similaridade entre duas SCs e assim conseguir identificar a distância entre as SCs foram propostas, podendo serem aplicadas no contexto de EI e no caso desse trabalho, especificamente para a subtarefa NER.

A área de EI ainda tem significativos avanços a realizar antes de podermos delegar aos computadores por completo a responsabilidade de definir a importância que cada frase ou palavra tem em um contexto. Para promover esse desenvolvimento da EI, avanços na subtarefa NER são necessários. Pois uma vez que as informações de um texto estão estruturadas de forma a ganharem semântica, outros processos computacionais de tratamento de informações vão conseguir trabalhar mais facilmente com essa informação.

Diante da relevância da subtarefa de NER no processo de EI e dos benefícios que podem ser gerados por uma abordagem de NER capaz de superar erros ortográficos e que seja flexível e adaptável à diferentes conjuntos de textos. Esse trabalho apresenta uma abordagem de NER baseado em dicionário e com recursos de ASM e proporcionar ao usuário recursos de configuração que permitam essa abordagem ser adequada à diferentes casos de estudo.

Capítulo 3

Reconhecimento de entidades nomeadas baseado em dicionário

Nesse capítulo, descrevo a abordagem de reconhecimento de entidades nomeadas baseado em dicionário, com recursos de associação inexata que foi desenvolvida nessa dissertação. Essa abordagem disponibiliza recursos de configuração com objetivo de permitir sua adaptação ao estudo de textos de diferentes domínios e com diferentes estruturas de escrita. Isso é feito através de parâmetros de configuração, aplicação de diferentes algoritmos de transformação de *tokens* e do uso de métricas para calcular a similaridade entre *tokens*, criando assim uma abordagem adaptável capaz de reconhecer entidades nomeadas em diferentes domínios.

Essa variedade de configurações possíveis busca, obter redução do tamanho da estrutura de dados utilizada para armazenar o dicionário, melhor desempenho da busca por associação inexata e o balanço entre precisão e cobertura. A estrutura do dicionário aqui utilizado foi desenvolvida para permitir ao usuário definir em um só arquivo os vários parâmetros de configuração utilizados. A figura 3.1 ilustra as tarefas da abordagem de NER apresentada nesse trabalho.

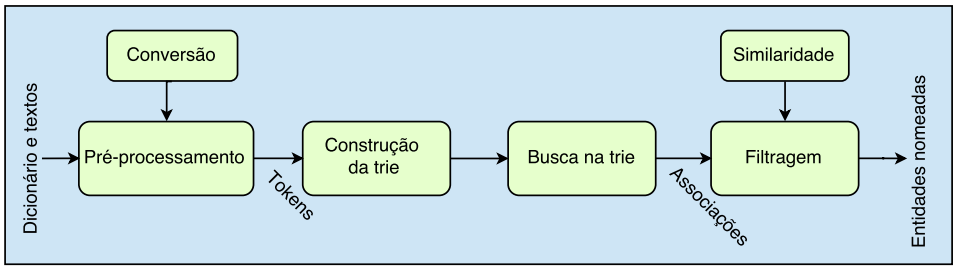

Figura 3.1: Ilustração da arquitetura geral do reconhecedor de entidades nomeadas apresentado nesse trabalho.

O restante do capítulo é organizado da seguinte maneira. Na seção 3.1 temos o pré-processamento, onde são descritos os parâmetros de configuração da abordagem, as regras utilizadas para a divisão em *tokens* e o módulo de conversão aplicado aos *tokens*. Esse pré-processamento prepara o reconhecedor de entidades nomeadas para realizar a construção da trie e também para realizar a busca sobre a trie. Na seção 3.2 descrevemos a construção da trie a partir do dicionário e a busca de entidades por associação inexata utilizando Edit distance. Na seção 3.3 é descrita a etapa de filtragem das associações que utiliza uma métrica de similaridade

entre sequências de caracteres para encontrar as melhores associações e então anotar as entidades nomeadas com os rótulos fornecidos no dicionário.

3.1 Pré-processamento

A abordagem aqui apresentada recebe como entrada um dicionário e textos para serem analisados. Ambos os documentos precisam ser pré-processados para serem utilizados na busca das entidades por associação aproximada. O primeiro pré-processamento aplicado tanto nas entradas do dicionário quanto nos textos é a divisão em *tokens*, isso é necessário uma vez que essa abordagem trabalha com associação aproximada entre *tokens* inteiros. O outro pré-processamento é realizado pelo módulo de conversão que aplica algoritmos de transformação na SC de cada *token* individualmente.

3.1.1 Divisão em tokens

Um *token* é uma sequência finita de caracteres que representam um subconjunto de uma outra sequência finita de caracteres construída sobre um determinado alfabeto. Os *tokens* são obtidos através da definição de delimitadores que ao serem aplicados sobre uma sequência de caracteres definem a posição inicial e final de cada *token* naquela sequência. Assim temos que ao aplicar os delimitadores em uma sequência de caracteres S com tamanho $|S|$, é definido um conjunto "P" de *tokens*. O tamanho de cada *token* t do conjunto P, representado por $|t|$, deve ser $0 < |t| \leq |S|$ e a soma do tamanho de todos os *tokens* de P deve ser $\sum |P_{1..n}| \leq |S|$.

As associações que representam as entidades nomeadas são realizadas considerando o *token* inteiro e não parcialmente, para permitir que a transformação seja aplicada sobre as mesmas condições tanto para o dicionário quanto para o texto. Nos casos de entradas compostas de múltiplos *tokens* a associação é realizada do início do primeiro até o final do último *token*. Com essa característica descrita temos a necessidade de um recurso que defina os limites de cada *token*. Por exemplo, o dicionário contém a entrada "Curitiba" e o texto é "Viajar de Curitiba para Curitibanos", com ED máximo aceito sendo 1. Temos como entidade no texto apenas a palavra "Curitiba". A palavra "Curitibanos" apesar de possuir uma subparte igual à entrada do dicionário não satisfaz o ED quando considerada em sua totalidade.

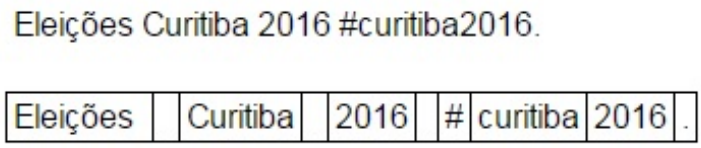

Figura 3.2: Divisão de uma frase em *tokens*.

A figura 3.2 ilustra a divisão de uma frase em *tokens* seguindo as regras utilizadas nesse trabalho. Abaixo temos a descrição das regras:

- Todo caractere seguinte a um espaço em branco e diferente de espaço em branco é considerado o início de um *token*;
- Um caractere de número após uma letra e o inverso, são considerados inícios de *tokens*;
- Todo caractere que não é letra nem número é considerado um *token*.

3.1.2 Módulo de conversão de tokens

O módulo de conversão consiste em recurso que permite aplicar algoritmos de transformação em uma sequência de caracteres produzindo como resultado uma sequência diferente. Assim temos que dada uma sequência de caracteres a e um módulo de conversão C, o resultado de $C_{(a)}$ é uma sequência b. Podendo b ser diferente de a em tamanho e alfabeto utilizado.

Cada *token* é convertido individualmente, isso é importante para que a conversão seja aplicada seguindo as mesmas regras tanto para as entradas do dicionário quanto para o texto. Cada *token* é enviado para o módulo de conversão o qual aplica um algoritmo de transformação (Double Metaphone, Soundex, stemming, etc.). Após a transformação o módulo de conversão retorna a nova SC que representa a forma convertida do *token* original. Se o *token* faz parte de uma entrada, a nova SC (transformada) vai ser agrupada com as demais SCs dos outros *tokens* da entrada. Após todos os *tokens* daquela entrada estarem convertidos, a entrada na forma convertida é enviada para ser indexada na trie. Se for um *token* do texto, as versões convertidas dos *tokens* são ordenadas seguindo a ordem dos *tokens* originais que elas representam. Nesse trabalho foi utilizado o algoritmo de transformação Double Metaphone, para realizar as transformações no módulo de conversão. Esse algoritmo além de eliminar alguns erros relacionados com fonética, comprime as SCs transformadas permitindo assim a redução da estrutura que armazena o dicionário.

Como descrito na seção 2.1.1, o reconhecimento de entidades nomeadas baseado em dicionário consiste em encontrar determinada cadeia de caracteres definida como entrada em uma cadeia de caracteres maior que seriam os textos. Quando a busca aceita que a entrada pode ocorrer com determinado grau de variação no texto, a busca se torna mais complexa do que quando realiza associação exata. Como o aumento do valor de ED máximo aceito entre as SCs aumenta o tempo gasto para a realização da busca sobre a trie para determinar as ocorrências de cada entrada no texto, foi utilizado o módulo de conversão para reduzir o ED entre as SCs antes de realizar a busca, permitindo produzir mais associações para um mesmo valor de ED definido como o máximo permitido.

Outra contribuição da utilização do módulo de conversão é a redução do comprimento das SCs. Essa característica faz com que o uso de funções de transformação promova a diminuição do tamanho da trie necessária para indexar o dicionário e assim diminui também a quantidade de nós a serem visitados pela busca na trie.

Procurando fornecer um sistema flexível que possa ser adaptado pelo usuário ao maior número de situações com o menor esforço possível, o módulo de conversão pode ser configurado pelo usuário. Uma vez que as regras de transformação estejam implementadas em uma biblioteca basta informar a classe e seu método que realiza a transformação. O módulo de conversão deve receber como parâmetro uma sequência de caracteres (original), aplicar as transformações e retornar outra sequência de caracteres (convertida). A utilização de acoplamento externo para o módulo de conversão permite ao usuário utilizar o algoritmo de transformação que deseja sem a necessidade de alterar o código fonte da abordagem de NER.

3.2 Busca de entidades nomeadas

Nessa seção são apresentadas as atividades para a construção da trie que indexa as entradas do dicionário e o processamento dos textos realizando a busca utilizando Edit distance para identificar as entidades nomeadas através de associação inexata.

3.2.1 Construção da trie

A utilização de uma trie para indexar o dicionário permite pesquisas mais eficientes sobre a lista de entradas. A trie permite desenvolver uma abordagem que utiliza a memória principal para armazenar dicionários maiores ao fazer uso compartilhado da estrutura para indexar prefixos iguais de diferentes entradas.

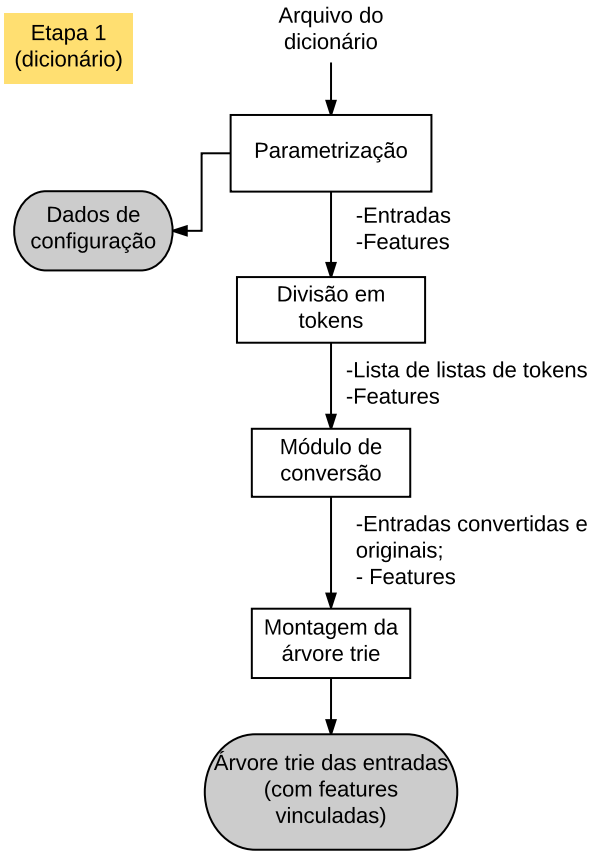

Figura 3.3: Arquitetura - etapa 1: Representa os processos para o tratamento do arquivo dicionário e montagem da trie.

As primeiras atividades realizadas em cada execução são referentes ao tratamento do dicionário e são ilustradas na figura 3.3. Os parâmetros de configuração obtidos do dicionário são descritos anteriormente na seção 3.4. Após a configuração ter sido realizada na etapa ”*Parametrização*”, inicia o processamento da lista de entradas e construção da trie. Além das informações de configuração, o dicionário utilizado nesse trabalho fornece as entradas e os rótulos que serão utilizados para anotar as entidades nomeadas identificadas.

Os rótulos que possuem um valor atrelado e ao serem vinculados com uma entidade do texto classificam a mesma. Por exemplo, uma entrada ”Curitiba” definida no texto com os rótulos ”Tipo=cidade” e ”Tamanho=grande”, permite encontrar as anotações que são do tipo cidade, pois outras entradas poderiam ter o mesmo rótulo ”Tipo” com o valor ”cidade” ou com valores diferentes como ”estado” ou ”país”. O recurso de anotação por rótulos foi adaptado das anotações realizadas pelo recurso de processamento Gazetteer do Gate [Cunningham et al., 2014].

Esse trabalho utiliza dois grupos de rótulos, os gerais e os individuais. Os rótulos gerais são fornecidos em um conjunto e são aplicados em todas as anotações produzidas, ou seja, estão vinculadas à todas as entradas do dicionário. Os rótulos individuais são informados na mesma linha que uma entrada e estão vinculados exclusivamente àquela entrada. A figura 3.4 ilustra

uma linha do dicionário com uma entrada e seus rótulos individuais e a figura 3.5 possui a indicação de cada elemento da linha e onde podemos perceber a utilização de alguns parâmetros de configuração como ”delimitador da entrada” e o ”separador de rótulos”.

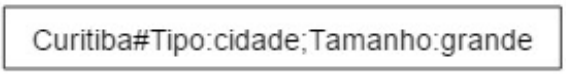

Figura 3.4: Linha com entrada fornecida no dicionário.

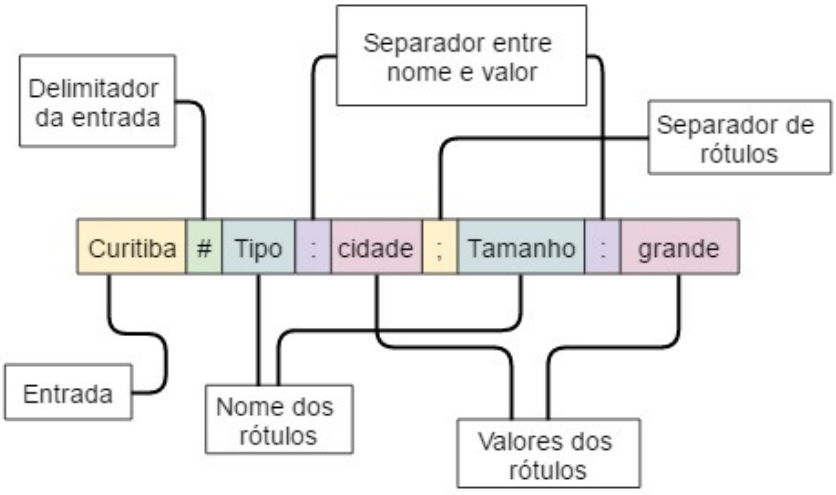

Figura 3.5: Identificação das partes de uma linha de entrada fornecida pelo dicionário.

Após a obtenção da lista de entradas e seus rótulos, cada entrada do dicionário passa pela ”*Divisão em tokens*” produzindo uma lista de *tokens* para representar cada entrada. Isso permite que cada *token* possa ser convertido em uma nova SC assim como será feito com os *tokens* do texto. Com a lista dos *tokens* de cada entrada definida pode-se aplicar as regras de conversão sobre os mesmos utilizando o *Módulo de conversão* e assim obter uma nova SC para cada entrada. A *Montagem da árvore trie* é feita utilizando a versão convertida de cada entrada. Cada *token* tanto da entrada, que pode ser composta de múltiplos *tokens*, quanto do texto, são convertidos individualmente e depois agrupados na ordem original.

A figura 3.6 representa a estrutura de cada nó da árvore trie. Cada nó da trie é formado por um conjunto de campos, alguns desses campos são preenchidos no momento de construção da trie e outros são utilizados no processo de busca. Abaixo são descritos os campos que compõe cada nó da trie:

- identificador: Campo que é preenchido com um número inteiro no momento da construção da árvore. Esse valor é único, não sendo repetido para outro nó.
- caractere: Esse campo armazena o caractere da palavra indexada. Cada caractere de uma palavra indexada é armazenado por um nó, o caractere seguinte da mesma palavra é indexado por um nó filho ao nó referente ao caractere anterior. Isso pode ser visualizado na figura 2.11.

Tipo	Nome
Número inteiro	identificador
Caractere	caractere
Lista	filhos
Booleano	fimPalavra
Booleano	EDvalido
Número inteiro	valorED
Número inteiro	nível
Lista	entradas

Figura 3.6: Estrutura de cada nó da trie com o nome de cada campo e os respectivos tipos de dados que armazenam.

- filhos: É uma lista com todos os nós filhos daquele caractere. Todo nó da trie que não é folha possui no mínimo um nó filho. Na figura 2.11, o nó que contém a letra ”p” (valor do campo caractere), possui dois filhos, os nós ”e” e ”o”.
- fimEntrada: Esse campo define se aquele nó representa um final da entrada, sendo preenchido com valores booleanos (verdadeiro ou falso). Nem todos os nó que são final de palavra são nós folhas, como podemos ver na figura 2.11, os nós pintados de cinza são nós que representam o final de uma palavra.
- EDvalido: Esse campo é utilizado durante as buscas sobre a trie. Ele é do tipo booleano (verdadeiro ou falso). Quando assinalado como verdadeiro, indica que aquele nó está a uma distância aceitável da palavra buscada (nó ativo). Na criação da trie esse campo recebe falso em todos os nós.
- valorED: Um número inteiro utilizado nas buscas para indicar o valor da métrica ED entre aquele nó e a SC buscada. Na criação da trie todos os nós recebem 0 (zero) nesse campo.
- nível: Campo que armazena um número inteiro referente ao nível do nó na árvore. O nível de um nó é definido pela quantidade de nós entre o nó e a raiz da trie. A raiz da trie sempre é nível 0 (zero) e seus filhos são nível 1 (um) e assim segue sempre incrementando em 1 o valor do nó filho em relação ao valor do nível de seu nó pai, sendo que os filhos de um mesmo nó tem o mesmo nível. Usando a figura 2.11 para exemplificar, o nó raiz (μ) é nível 0 (zero) e seus filhos (t) são nível 1 (um), aplicando essa regra a todos os nós, teremos que os dois nós filhos são nível 5 (cinco).
- entradas: Uma lista que armazena as entradas do dicionário que são representadas na trie pelo caminho da raiz até aquele nó.

Com a estrutura do nó definida podemos prosseguir para o processo de construção da trie. O algoritmo 1 representa os processos para incluir uma nova entrada na trie. A trie inicia com a definição do nó raiz que representa uma entrada sem caracteres, vazia. Após a raiz cada nó será representado por um caractere e terá ponteiros para nós filhos que representam o caractere seguinte daquela entrada. Caso duas ou mais entradas possuam um prefixo igual elas vão compartilhar a mesma estrutura (sequência de nós) da trie para aquele prefixo.

Na linha 7 do algoritmo 1 temos uma análise de condição que define se um novo nó filho precisa ser criado ou se já existe um nó filho para o caractere seguinte. Um novo nó é criado se não há um nó filho que represente o caractere que se deseja indexar. Exemplo: Para inserir a

entrada ”tampa” na trie da figura 2.11 seria utilizado o nó já existente para a letra ”t”, logo após a raiz, e então seria necessário criar um novo nó filho para representar o caractere ”a”. Quando um nó filho é criado para representar um caractere, todos os caracteres seguintes para aquela entrada vão exigir a criação de novos nós. Na linha 15 do algoritmo o nó que armazenou a última letra da entrada recebe verdadeiro para o campo ”fimEntrada” e a linha 16 adiciona a entrada atual em sua versão original ao conjunto de entradas daquele nó.

Algoritmo 1 Adiciona entrada na trie

```
1:  função ADICIONA ENTRADA(Nó trie, Entrada entrada)
2:      Nó nAtual ← trie
3:      int x ← 0
4:      enquanto x ≤ entrada.tamanho faça
5:          caractere c ← entrada[x]
6:          Nó nExiste ← nAtual[c]
7:          se nExiste = nulo então
8:              nAtual.filhos[c] ← Nó novoFilho
9:              nAtual ← novoFilho
10:         senão
11:             nAtual ← nExiste
12:         fim se
13:         x ← x + 1
14:     fim enquanto
15:     nAtual.fimEntrada ← verdadeiro
16:     nAtual.entradas ← entrada
17: fim função
```

3.2.2 Busca por entidades nomeadas utilizando Edit distance

Após processar o dicionário e montar a árvore trie podemos realizar a busca por ocorrências das entradas nos textos. A arquitetura da busca é ilustrada pela figura 3.7 e mais detalhadamente representada pelos algoritmos 2 e 3, sendo o algoritmo 3 executado pela linha 17 do algoritmo 2.

A busca utilizando trie parese abordagem é baseada em trabalhos que apresentam soluções de auto completar [Ji et al., 2009, Deng et al., 2016]. Ji desenvolveu um sistema de autocompletar que utilizou a estrutura de árvore trie para armazenar as palavras a serem sugeridas enquanto o usuário digita as letras uma após a outra. A busca por ele apresentada utiliza um princípio de ativação dos nós da trie. Os nós ativos são o conjunto de nós que representam as sequências que estão a uma distância de edição igual ou menor ao máximo ED aceito em relação a sequência de caracteres buscada.

Como a técnica de busca com ativação de nós se mostrou eficiente para sistemas de autocompletar, a abordagem aqui apresentada utiliza uma adaptação da mesma para realizar a busca no contexto de NER. Foram feitas alterações na estrutura dos nós da trie para armazenar a relação entre a versão convertida e a original de cada entrada, e para permitir a busca que considera *tokens* inteiros.

A figura 3.8 representa a busca com a ativação dos nós da trie realizada pelo reconhecedor de entidades nomeadas conforme os caracteres de um *token* são lidos. Para essa busca temos um dicionário que contém as entradas centro, cantar e dentro, e o texto no qual as entradas devem ser

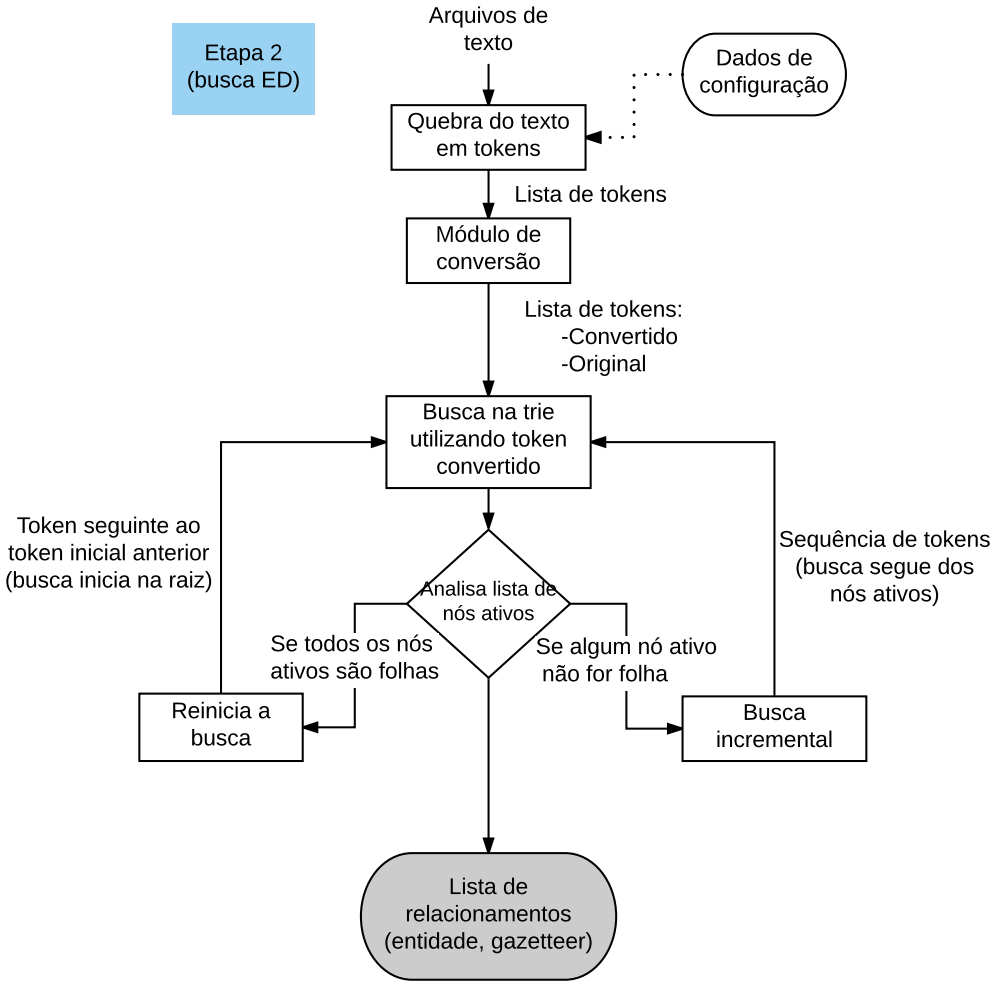

Figura 3.7: Arquitetura - etapa 2: Representa os processos da busca por ocorrências das entradas no texto.

localizadas é "cntro da cidade". Os caracteres no canto superior direito indicam o *token* que está sendo buscado. A busca pela letra ”c” produz a ativação dos nós que estão a uma distância ED menor ou igual ao máximo aceito (2 para esse exemplo). Os nós ativos são os que possuem o número em vermelho ao lado indicando o valor do ED daquele nó em relação a parte lida do *token*. Conforme são lidos os próximos caracteres do *token* a ativação dos nós segue pela árvore.

Quando o valor de ED de um nó da trie ultrapassa o máximo aceito, isso faz com que o nó seja retirado da lista de nós ativos. Ao ler todo o *token* o conjunto de nós ativos e que são finais de entradas definem as entradas que atendem a regra de ED máximo e assim podem ser associadas com a subparte do texto para de. No exemplo da figura 3.8 utilizado as associações retornadas serão <centro, cntro> com ED=1 e <dentro, cntro> com ED=2.

No algoritmo 2 são descritos procedimentos da busca iniciando pela quebra do texto em *tokens* e a conversão dos *tokens*, nas linhas 9 e 10 respectivamente. Após obter a lista de *tokens* cada um deles é submetido à busca (linha 11) e para cada caractere do *token* o algoritmo 3 define a lista de nós ativos. Se para algum caractere do *token* o conjunto de nós ativos for zerado (linha 18 alg 2), a leitura do *token* é finalizada e a lista de *tokens* que compõe a entidade é zerada (linha 37 alg 2), passando a busca para o próximo *token* do texto (linha 10 alg 2). Mas se o conjunto de nós ativos possuir elementos quando o último caractere do *token* for lido (linhas 18 e 19 agl 2), para cada nó ativo que representar o final de uma entrada (linha 21 alg 2) será criada

Algoritmo 2 Procura na árvore por entradas similares à SC lida no texto

```
1:  função BUSCA(Nó trie)
2:      Lista Token tokensTexto;
3:      Lista Token tokensConvertidos;
4:      Lista Nó nsAtivos;
5:      Lista Token entidade;
6:      Lista (entrada, entidade) associaes;
7:      Booleano continua;
8:      tokensTexto ← texto dividido em tokens;
9:      A lista tokensConvertidos recebe os tokens da lista tokensTexto após passarem pelo módulo de conversão;
10:     para cada token ∈ tokensConvertidos faça
11:         entidade ← token;
12:         tokenAtual ← token;
13:         nsAtivos ← raiz
14:         enquanto entidade ≠ nulo faça
15:             continua ← falso;
16:             para cada Caractere atualC ∈ tokenAtual faça
17:                 nsAtivos ← DEFINE_NOS_VALIDOS(n, n2, atualC, nsAtivos)        ▷ Algoritmo 3
18:                 se nsAtivos ≠ nulo então
19:                     se atualC é último caractere do tokenAtual então
20:                         para cada n2 ∈ nsAtivos faça
21:                             se n2.fimEntrada então
22:                                 associações ← (n2.entrada, entidade);
23:                             fim se
24:                         fim para
25:                     fim se
26:                     para cada n2 ∈ nsAtivos faça
27:                         se n2.filhos ≠ nulo então
28:                             tokenAtual ← próximo token;
29:                             entidade ← tokenAtual;
30:                             continua ← verdadeiro;
31:                         fim se
32:                     fim para
33:                     se continua = falso então
34:                         entidade ← nulo;
35:                     fim se
36:                 senão
37:                     entidade ← nulo;
38:                 fim se
39:             fim para
40:         fim enquanto
41:     fim para
42:     Retorna associaes;
43: fim função
```

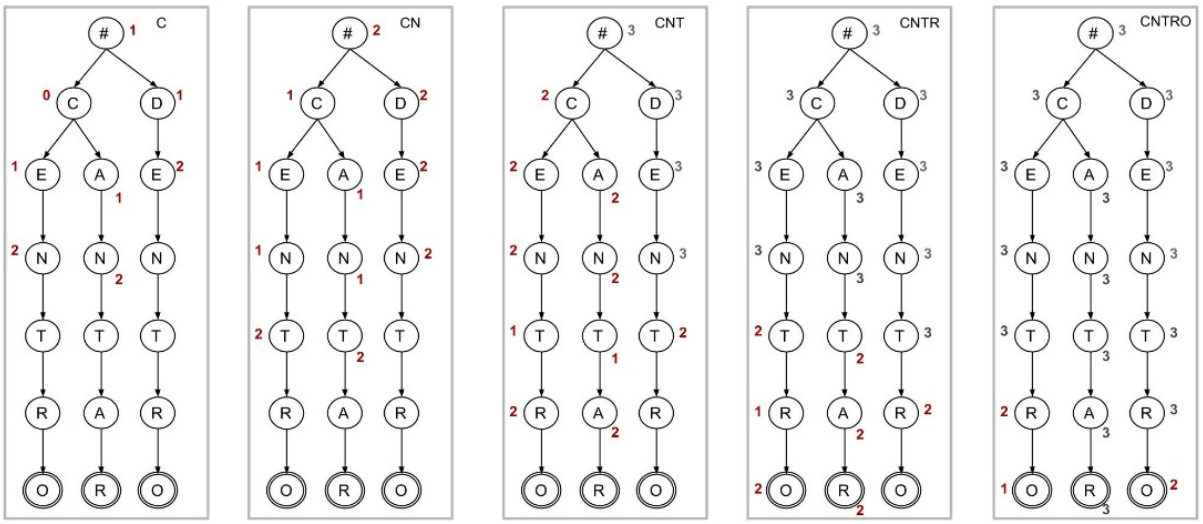

Figura 3.8: Busca da palavra ”cntro” (versão errada da palavra ”centro”) sobre uma trie com as entradas (centro, cantar e dentro). O ED máximo aceito é 2. Os nós com círculo duplo indicam final de uma entrada.

uma associação entre a entidade e as entradas daquele nó. Após a criação das associações, é verificado se algum nó do conjunto de nós ativos possui filhos (linhas 26 e 27 alg 2) é realizada a busca incremental. A busca incremental consiste em ler o próximo *token* e adicionar este ao final do anterior, criando assim uma possível entidade composta por múltiplos *tokens* (linhas 28, 29 e 30 alg 2).

A definição de nós ativos realizada no algoritmo 3 sempre inicia pelo nó ativo (linha 4 alg 3) mais profundo na trie. O nós são avaliados e se o nó filho do nó ativo for representado pelo caractere que está sendo buscado (linha 6 alg 3), o nó filho recebe como valor do ED o mesmo valor do ED do seu nó pai e se o caractere do nó filho for diferente (linha 16 alg 3), o valor do ED do nó filho será o ED do nó pai incrementado em 1. Se o ED de um nó já for igual ao ED máximo aceito, nenhum filho que seja diferente do caractere buscado será ativado, pois nesse caso o ED do nó filho sempre será maior que o máximo aceito.

As condições verificadas nas linhas 8 e 18 do algoritmo 3 aplicam o princípio de que um nó ativo sempre vai ter o menor valor de ED ao qual tem direito. E os comandos aplicados pela verificação da linha 30 definem que se um nó filho for ativado com um valor de ED menor que o máximo aceito, os nós abaixo dele serão ativados até que todos os ramos de sua subtrie obtenham o valor de ED igual ao máximo aceito.

Após todos os nós filhos do nó atual serem verificados o nó atual tem seu valor de ED incrementado em 1 e se assim ultrapassar o ED máximo aceito (linha 39 alg 3) o nó é desativado.

O resultado da busca sobre a trie é um conjunto de associações que definem as possíveis entidades nomeadas a serem identificadas pelo sistema. Após serem definidas as associações são enviadas para a etapa de filtragem. A filtragem elimina as associações que foram criadas entre entradas e entidades pouco similares.

3.3 Filtragem

A filtragem consiste na utilização do *módulo de similaridade* para aplicar uma métrica que calcula a similaridade entre a entrada e a entidade de cada associação. O valor retornado

Algoritmo 3 Define o conjunto de nós válidos

```
1:  Inteiro maxED;
2:  Lista Nó nosAtivosNovo;
3:  função DEFINE_NOS_VALIDOS(Nó n, Nó filho, Caractere atualC)
4:      para cada n ∈ nsAtivos faça
5:          para cada n_2 ∈ n.filhos faça
6:              se filho.caractere = atualC então
7:                  se nosAtivosNovo contém filho então
8:                      se filho.valorED > n.valorED então
9:                          filho.valorED ← n.valorED;
10:                     fim se
11:                 senão
12:                     filho.EDvalido ← verdadeiro;
13:                     filho.valorED ← n.valorED;
14:                     nosAtivosNovo ← filho;
15:                 fim se
16:             senão se (n.valorED + 1) ≤ maxED então
17:                 se nosAtivosNovo contém filho então
18:                     se filho.valorED > n.valorED + 1 então
19:                         filho.valorED ← (n.valorED + 1);
20:                     fim se
21:                 senão
22:                     filho.EDvalido ← verdadeiro;
23:                     filho.valorED ← (n.valorED + 1);
24:                     nosAtivosNovo ← filho;
25:                 fim se
26:             senão
27:                 n.EDvalido ← falso;
28:                 Remove n da lista nsAtivos;
29:             fim se
30:             se filho.valorED < maxED então
31:                 - Descer na subárvore do filho ativando os nós (nó.EDvalido ← vardadeiro) e setando o valorED.
32:                 - É mandatório que todo nó mantenha o menor valorED que tem direito.
33:                 - Sem associação, o valorED do filho recebe o (valorED do pai + 1).
34:                 - Com associação, na primeira associação do caminho após nó n, o filho recebe o valorED do
    seu nó pai, nos níveis seguintes o filho sempre recebe o (valorED do pai + 1).
35:                 - Devemos descer até o valorED do nó ser igual à maxED.
36:                 - Todos os nós ativos são adicionados à lista "nósAtivos".
37:             fim se
38:         fim para
39:         se n.valorED + 1 ≤ maxED então
40:             n.valorED ← n.valorED + 1;
41:             nosAtivosNovo ← n;
42:         fim se
43:     fim para
44:     Retorna nosAtivosNovo;
45: fim função
```

pelo módulo de similaridade é utilizado por filtros que selecionam as associações que atendem aos parâmetros fornecidos pelo usuário. A etapa de filtragem é representada pela figura 3.9.

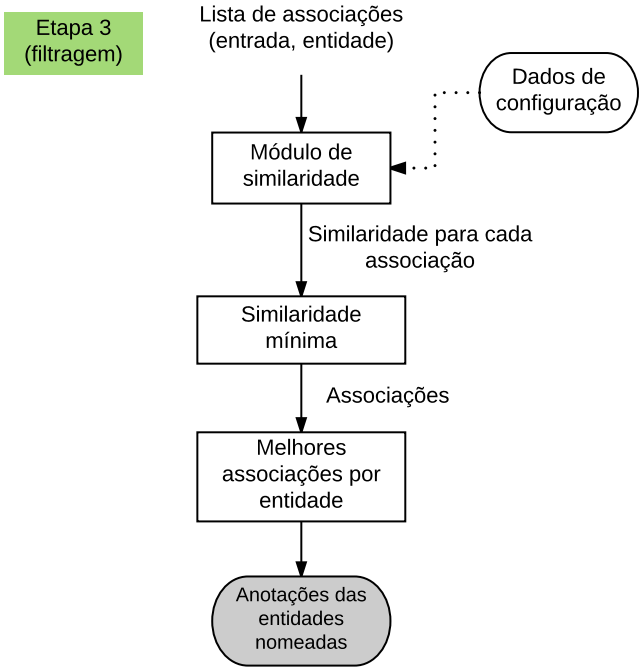

Figura 3.9: Arquitetura - etapa 3: procedimentos para filtrar as associações definidas na etapa 2.

3.3.1 Módulo de similaridade

O módulo de similaridade utiliza uma métrica de similaridade para fornecer um valor que represente quão significativa é a similaridade entre a entidade e a entrada. Para isso podem ser utilizadas métricas de similaridade entre SCs como Jaro-Winkler, n-gram, String$_{Sim}$, etc. Definir a similaridade entre as versões originais das SCs relacionadas é fundamental para a abordagem desenvolvida nesse trabalho, pois o ED é calculado utilizando a versão convertida tanto das entradas quanto dos textos, e a conversão pode gerar alterações significativas na cadeia de caracteres fazendo com que o ED não represente a real similaridade entre as SCs da associação.

Também com o objetivo de permitir ao usuário utilizar diferentes recursos, a abordagem foi construída de forma que ele possa alterar as regras que calculam a similaridade utilizando as configurações do arquivo de dicionário para informar a implementação que deseja utilizar. São duas chaves que devem ser informadas, a ”_similarityClass” informa a classe e a ”_similarity-Method” informa o nome do método dessa classe. O método deve receber como parâmetro duas SCs e fornecer como retorno um valor numérico que será considerado a similaridade entre a entidade e a entrada.

A métrica de similaridade deve retornar um valor maior para similaridades maiores. Essa é uma regra que deve ser seguida pois os filtros que serão aplicados consideram que associações com valores maiores são as melhores. Uma possibilidade é a normalização do valor da similaridade entre 0 e 1, sendo 1 (um) o valor de similaridade entre SCs idênticas e 0 (zero) indica SCs sem similaridade.

O módulo de similaridade associado ao valor máximo de ED aceito na busca sobre a trie permite ao usuário calibrar o sistema para obter o balanço desejado entre precisão e cobertura. Uma maior precisão indica que os resultados condizem com o conjunto de entradas fornecido. A

Entidade	Início entidade	Fim entidade	Entrada relacionada	Edit distance	Similaridade	Rótulos
comback	18	24	comeback	1	0,91	Tipo=ABC|Grupo=XYZ|
aiport	36	41	airport	1	0,95	Tipo=ABC|Grupo=XYZ|

Figura 3.10: Exemplo de anotações. A linha amarela contém o rótulo das informações de cada anotação. Os valores de "Similaridade" foram obtidos com a métrica String_{Sim}.

cobertura é a porcentagem de ocorrências de entidades encontradas pelo sistema em relação ao total de ocorrências realmente existentes no texto. Se as regras de comparação forem relaxadas (alto valor de ED aceito e baixa similaridade mínima) a tendência é que a busca identifique mais ocorrências de entidades ao custo de diminuição da precisão já que mais associações falsas vão ser produzidas [Navarro et al., 2003]. Após a similaridade ser definida para cada associação, é iniciada a filtragem para obtenção das associações mais adequadas.

3.3.2 Filtragem das associações pela similaridade

Após o valor da similaridade ser definido entre a entrada e a entidade de todas as associações, são aplicadas duas regras de filtragem. A primeira é a *similaridade mínima*, essa regra elimina as associações com similaridade inferior a um valor definido. Com o objetivo de proporcionar maior flexibilidade para a abordagem de NER desse trabalho, é permitido ao usuário informar qual a similaridade mínima que ele aceita através de um parâmetro de configuração. Com isso similaridades com valor menor ao informado não serão consideradas pelo sistema.

A segunda regra é aplicada para as associações aprovadas na regra anterior. Essa regra seleciona para cada entidade as associações com as maiores similaridades, ou seja, define o ranque para as entradas do dicionário que foram relacionadas com uma mesma SC (mesma posição inicial e final) do texto. A quantidade de maiores similaridades aceita é definida pelo usuário também através de um parâmetro de configuração. Assim temos que sendo n a quantidade de maiores similaridades aceitas, apenas as entradas que obtiveram similaridade entre as n maiores similaridades de cada entidade serão aceitas. É importante frisar que são as maiores similaridades, o que permite que mais de n associações sejam aceitas uma vez que entradas diferentes podem apresentar a mesma similaridade com uma determinada entidade.

Exemplo: Dadas 3 (três) associações a, b e c para uma mesma entidade do texto (mesma posição inicial e final no texto). E definida a quantidade de maiores similaridades como 1 (um), assim apenas as primeiras colocadas no ranqueamento serão aceitas. E sendo o valor das similaridades (Sim) das associações $Sim(a) = 0{,}9$, $Sim(b) = 0{,}8$ e $Sim(c) = 0{,}9$. Serão aceitas para anotações as associações a e c, pois ambas possuem a maior similaridade.

Após esse procedimento as associações aprovadas vão ser anotadas como entidades nomeadas. Essa anotação contém a posição de início e fim da entidade no texto, a entrada associada, o valor do ED, o valor da similaridade e os rótulos atribuídos à entrada.

A figura 3.10 apresenta exemplos de anotações geradas pelo sistema. Abaixo descrevemos o que cada coluna representa:

1. *Entidade*: Subparte do texto que foi definida como uma entidade nomeada.
2. *Início entidade*: Posição no texto do primeiro caractere da entidade.
3. *Fim entidade*: Posição no texto do último caractere da entidade.
4. *Entrada relacionada*: Entrada que anotou aquela entidade.

5. *Edit distance*: Valor do ED definido durante a busca aproximada sobre a trie.

6. *Similaridade*: Valor calculado pela métrica do módulo de similaridade e utilizada na filtragem.

7. *Rótulos*: Conjunto de rótulos vinculados à entrada utilizado na anotação.

Ao produzir as anotações para o último arquivo de texto, o reconhecimento de entidades nomeadas está finalizado.

3.4 Parametrização

A parametrização consiste na aplicação de valores fornecidos pelo usuário utilizando o dicionário para definir o comportamento da abordagem de NER desse trabalho e afetam a construção da trie, a busca por associação inexata sobre a trie e a filtragem das associações. Cada parâmetro é passado através de um par formado por uma chave e seu valor (chave=valor) e o conjunto dos parâmetros compõe a parte do dicionário que é denominada de *cabeçalho de configuração*. Abaixo é explicado cada parâmetro de configuração necessário para realizar a execução.

(**_maxEditDistance=1**) Valor máximo do Edit Distance aceito na busca sobre a trie. Nesse caso o valor é 1.

(**_quantityBetterSimilarities=1**) Define quantas melhores similaridades devem ser consideradas para anotar cada entidade. Se na busca sobre a árvore trie foram encontradas 5 entradas com ED aceito, mas o sistema estiver definido para retornar apenas a melhor similaridade. Apenas as entradas que apresentarem a maior similaridade vão anotar a entidade. Pode ser um conjunto de entradas, já que diferentes entradas pode apresentar uma mesma similaridade com determinada entidade. O valor informado para essa configuração deve ser um número inteiro e maior que zero.

(**_minSimilarityAccepted=0.9**) Similaridade mínima aceita entre a entidade e a entrada de cada associação. Deve ser um valor entre o mínimo e máximo retornado pela métrica de similaridade definida para o módulo de similaridade. Se nenhum relacionamento possuir similaridade maior ou igual ao valor definido nessa chave, a entidade não receberá anotação.

(**_editDistanceFeatureName=ED**) Rótulo utilizado para identificar que vai anotar o valor do ED na entidade.

(**_similarityClass=br.ufpr.stringsim.StringSim**) O valor para essa chave deve indicar a classe que contém o método com a métrica de similaridade.

(**_similarityMethod=stringSim**) Esta chave define o método que será utilizado no módulo de similaridade, ou seja, esse método deve implementar a métrica de similaridade que se deseja utilizar.

(**_convertionClass=br.com.ftdistance.util.StringConvertUtil**) Informa a classe que contém o método que implementa o algoritmo de transformação. Deve ser informado o caminho até a classe mais o nome da classe (StringConvertUtil).

(**_convertionMethod=doubleMetaphoneConverter**) Informar o nome do método presente na classe informada na chave "_convertionClass". Esse deve ser o método que implementa a transformação de SCs que deseja-se utilizar.

(**_similarityFeatureName=Similaridade**) Rótulo utilizado para anotar a entidade com o valor da similaridade calculada entre a entidade e a entrada. Para exemplificar, se a similaridade retornada for 0,95, seria criada na respectiva entidade a seguinte anotação, "Similaridade=0,95".

(**_featureSeparator=;**) Caractere de separação utilizado no dicionário para separar os diversos rótulos de uma mesma entrada.

(**_featureValueSeparator=:**) Caractere que separa o nome do rótulo do seu valor.

(**_gazetteerDelimiter=#**) Caractere que define o final da sequência de caracteres considerada como entrada. O caractere anterior a este é o último caractere da entrada e o caractere seguinte é o primeiro do nome do rótulo.

(**_textListFile=listaTextos.txt**) O valor dessa chave é o nome do arquivo que contém a relação dos arquivos dos textos que devem ser analisados.

O cabeçalho de configuração é delimitado dentro do dicionário por marcadores. A ordem em que as chaves são definidas não afeta o sistema, porém algumas regras de formatação devem ser seguidas:

- Cada conjunto (chave=valor) deve estar em uma linha;
- A descrição da chave não deve ser alterada. O sistema procura especificamente pelas chaves descritas anteriormente.

O cabeçalho de configuração aceita a inserção de linhas de comentários. Os comentários são ignorados durante a execução e servem como recurso de orientação para o usuário. Uma linha de comentário deve ser iniciada com ”%%” (sem aspas). Uma linha utilizada como comentário não deve conter informações relevantes ao sistema e o inverso é verdadeiro.

A disponibilidade dos parâmetros configuração e a modularização dos processos de transformação das SCs (Módulo de conversão) e de cálculo de similaridade (Módulo de similaridade), são os recursos utilizados para proporcionar adaptabilidade à abordagem para o trabalho em diferentes contextos. Os parâmetros permitem a personalização da formatação do dicionário e da calibração do reconhecedor de entidades nomeadas para atingir o balanço desejado entre precisão e cobertura. Já os módulos permitem que diferentes algoritmos sejam utilizados tanto para a transformação quanto para o cálculo de similaridade, e com isso podem ser valorizadas determinadas características dos textos que são mais relevantes para o contexto do estudo.

3.5 Experimentos

Nessa seção são apresentados os dados de entrada utilizados para avaliação, as diferentes configurações aplicadas e os resultados obtidos por pela abordagem desenvolvida nesse trabalho. Foram utilizados diferentes valores de configuração para destacar o impacto que cada parâmetro tem nos resultados. Os resultados são descritos em valores de precisão e cobertura. Os testes foram repetidos com dois algoritmos diferentes para cada módulo, permitindo verificar as possibilidades que a utilização de uma abordagem flexível fornece. Esses módulos podem ser trocados para diferentes implementações de variadas técnicas.

3.5.1 Configurações dos experimentos

Nos testes realizados foi utilizado um dicionário formado por uma lista de 76.912 palavras em inglês com ortografia correta obtidas do WordNet[1] que representam o conjunto de entradas. Para o conjunto de entidades com erros que representam os textos foram utilizadas 1000 palavras selecionadas randomicamente de uma coleção fornecida pela Wikipédia[2] que contém

[1] http://wordnet.princeton.edu

[2] https://en.wikipedia.org/wiki/Wikipedia:Lists_of_common_misspellings

Tabela 3.1: Exemplos dos erros utilizados nos experimentos.

Correta	Errada
comeback	comback
holistic	hollistic
acquaintance	aquiantance

palavras com erros ortográficos e a respectiva forma correta. Na tabela 3.1 são apresentados alguns exemplos dos erros presentes nas palavras utilizadas nos experimentos.

Para esses arquivos de entrada foram realizadas execuções com diferentes configurações permitindo observar os efeitos que a mudança em cada parâmetro provoca nos resultados. Os experimentos foram realizados em um computador com um processador Intel(R) Core(TM) i5-3210M de 2.50GHz, com memória principal de 4GB ddr3 e com sistema operacional Windows 8.1 Pro versão X64.

O módulo de conversão foi aplicado através de duas variações. Utilizando Double metaphone e sem aplicação de conversão. A escolha pela conversão fonética Double metaphone objetiva dois benefícios. Primeiro é superar erros fonéticos através da conversão e segundo realizar a redução no tamanho das SCs a. O teste sem conversão serve como base para comparação com a busca por aproximação utilizando apenas o ED. A implementação de Double metaphone utilizada foi a disponibilizada pela Apache na biblioteca commons-codec-1.10.jar[3].

Para cada condição de conversão duas métricas de similaridade foram aplicadas. A métrica Jaro-Winkler implementada pela biblioteca lucene-suggest-5.2.1.jar[4] da Apache e a métrica String$_{Sim}$ de [Tissot et al., 2014] utilizando uma versão portada por nós para a linguagem Java. A Jaro-Winkler se destaca pela sua disseminação com variedade de implementações disponíveis e por servir de base para outras métricas desenvolvidas para determinados domínios. A String$_{Sim}$ por sua vez é uma métrica recente que propõe valores de similaridade com melhor escalonamento que a Jaro-Winkler. Ambas as métricas possuem como característica calcularem a similaridade considerando todos os caracteres de ambas as SCs.

Para cada métrica de similaridade, o sistema foi executado com 3 diferentes valores de ED máximo (0, 1 e 2). O valor 0 (zero) foi utilizado para avaliar o efeito do módulo de conversão na eliminação de erros. O valor 2 foi estabelecido como máximo pois a quantidade de falsos positivos já é muito elevada. Outro ponto que limitou o valor do ED foi o aumento no tempo de processamento das buscas, que com valores acima de 2 inviabiliza as buscas.

Para cada valor de ED diferente foram utilizadas 3 similaridades mínimas 0,7, 0,8 e 0,9. O valor máximo de similaridade foi definido em 0,9 por selecionar associações com maior similaridade, uma vez que a associação exata é definida por 1,0 em ambas as métricas utilizadas. Valores de similaridade menores que 0,7 permitem SCs muito diferentes aumentando a quantidade de associações falsas. A estratificação em 0,1 para a similaridade foi definido considerando o quanto essa alteração relaxa ou restringe a regra de similaridade mínima.

A última configuração foi a quantidade de maiores similaridades aceita. Foram utilizados os valores 1 e 3 para cada similaridade mínima. O número 1 com o objetivo de elevar a precisão do sistema ao máximo, retornando apenas a associações com a maior similaridade. O valor 3 objetiva elevar o valor da cobertura do sistema, permitindo localizar o maior número de entidades presentes no texto. porém limitamos a 3 para evitar elevar muito o número de falsas associações.

Esse conjunto de parâmetros acima descrito gerou 72 configurações para a abordagem:

[3]https://commons.apache.org/proper/commons-codec/download_codec.cgi

[4]http://lucene.apache.org/

(2 (conversões) x 2 (funções similaridade) x 3 (ED) x 3 (similaridades mínimas) x 2 (maiores similaridades)) = 72

Para comparação entre os resultados obtidos com as 72 configurações foram observados 7 dados em 3 execuções de cada configuração: Precisão, cobertura, F1 Score, 3 medidas de tempo e o tamanho da trie.

- Precisão: Percentual de anotações corretas do total de anotações produzidas.
- Cobertura: Percentual de entidades anotadas em relação ao total presente nos dados.
- F1 Score: É a média harmônica entre a precisão e a cobertura.
- Tempo de construção da árvore trie: É o tempo gasto para processar o arquivo dicionário, desde sua leitura até completar a construção da árvore trie. Representa os processos ilustrados na figura 3.3.
- Tempo busca sobre a trie utilizando ED:É o tempo gasto para realizar os processos da segunda etapa do sistema (figura 3.7). Que vai desde a leitura do arquivos texto até o retorno de associações entre entidades e entradas.
- Tempo gasto pela métrica de similaridade: Representa o tempo gasto para calcular a similaridade para todas as associações obtidas pela busca das entradas no texto e a filtragem das mesmas de acordo com a quantidade de melhores resultados que devem ser considerados.
- Número de nós da trie: É quantidade de nós necessários para indexar o dicionário e consequente quantidade de memória utilizada para armazenar o índice.

3.5.2 Resultados sobre a construção da trie

A tabela 3.2 apresenta uma comparação da utilização do Double metaphone para conversão das entradas com a utilização das entradas sem conversão. A utilização do Double metaphone conseguiu produzir uma redução superior a 95% na quantidade de nós da trie. O que implica em redução da memória necessária para armazenar o índice do dicionário.

Tabela 3.2: Dados da indexação do dicionário com 76.912 entradas.

Conversão	Total de caracteres	Maior entrada (caracteres)	Tamanho médio das entradas (caracteres)	Tamanho da trie (Número de nós)	Tempo médio de indexação (milissegundos)
Sem conversão	658.774	31	8,6	240.484	1120
Double metaphone	283.575	4	3,7	10.330	915

A redução no número total de caracteres que constituem todas as entradas foi de 56% quando utilizado o Double metaphone. Essa redução menor de número de caracteres em relação a redução da quantidade de nós, indica que mais prefixos iguais foram criados pelo processo de conversão. Permitindo que valores de ED menores produzam um maior número de associações e assim permitem identificar mais entidades nomeadas do que a busca com o mesmo ED sobre a trie sem conversão.

Outro ponto observado foi o tempo gasto para a montagem da trie apresentado na coluna ”Tempo médio de indexação”. Esse valor é a média dos tempos obtidos nas 36 configurações utilizadas para cada uma das regras de conversão. Apesar das execuções com conversão terem

que utilizar o módulo de conversão o tempo gasto para construção da trie foi menor. O que mostra que o tempo gasto na conversão é compensado pelo menor número de nós que precisam ser criados na trie. Os tempos de cada configuração estão disponíveis individualmente nas tabelas 3.3 e 3.4 para a conversão DM e nas tabelas 3.5 e 3.6 para as configurações sem conversão.

3.5.3 Resultados das buscas

Alterações em certos parâmetros de configuração tem mais impacto nos resultados. Isso acontece porque o valor de um parâmetro pode neutralizar os efeitos da alteração nos valores de outro. Podemos dar como exemplo a relação entre o ED máximo aceito e a similaridade mínima. Se o ED máximo aceito for 0 (zero) para uma execução utilizando o Double metaphone, o efeito de mudar a similaridade mínima entre 0,9 e 0,7 tem seu efeito sobre o resultado minimizado, pois o valor baixo do ED evita que associações com baixa similaridade sejam criadas.

Os resultados das 72 configurações são apresentados nas tabelas 3.3, 3.4, 3.5 e 3.6 e essas são ilustradas respectivamente pelos gráficos das figuras 3.11, 3.12, 3.13 e 3.14. Cada tabela agrupa os 18 resultados das diferentes configurações para um determinado par de algoritmos de transformação e métrica de similaridade. Os gráficos apresentam os valores de cada uma das configurações e cada gráfico contém os dados de uma determinada tabela. Cada conjunto com 3 colunas representa a precisão, a cobertura e o F1 para uma determinada configuração (Ranque: Quantidade de maiores similaridades consideradas; SM: Similaridade mínima aceita; ED: O maior Edit Distance permitido).

Tabela 3.3: Resultados das 18 configurações utilizando a conversão Double metaphone (DM) e similaridade Jaro-Winkler (JW).

ED	Similaridade Mínima	Maiores Similaridades	Precisão	Cobertura	F1	Tempo Árvore (ms)	Tempo Busca (ms)	Tempo métrica Similaridade (ms)
0	0,7	3	27,39%	80,40%	40,86%	906	62	204
0	0,7	1	75,05%	72,50%	73,75%	937	63	203
0	0,8	3	29,69%	80,40%	43,37%	1.062	47	219
0	0,8	1	77,29%	72,50%	74,82%	828	47	203
0	0,9	3	39,94%	78,00%	52,83%	922	47	235
0	0,9	1	**83,29%**	70,80%	76,54%	937	63	218
1	0,7	3	27,48%	91,00%	42,22%	844	391	859
1	0,7	1	75,46%	77,50%	76,47%	890	391	922
1	0,8	3	27,53%	91,00%	42,27%	937	422	1.078
1	0,8	1	75,46%	77,50%	76,47%	953	391	875
1	0,9	3	34,06%	88,90%	49,25%	844	359	922
1	0,9	1	78,21%	76,80%	77,50%	890	359	859
2	0,7	3	29,01%	**96,00%**	44,56%	1.000	2.328	5.787
2	0,7	1	79,40%	81,70%	**80,53%**	1.063	2.747	5.469
2	0,8	3	29,01%	**96,00%**	44,56%	844	2.344	5.578
2	0,8	1	79,40%	81,70%	**80,53%**	1.000	2.610	5.362
2	0,9	3	33,63%	94,10%	49,55%	985	2.547	5.434
2	0,9	1	79,94%	80,90%	80,42%	859	2.453	5.268

Mas variações conjuntas em diferentes parâmetros de configuração conseguem grandes alterações nos resultados. Com alterações de ED, similaridade mínima e maiores similaridades, foram gerados valores de cobertura com variação de 25 pontos percentuais. Vemos essa variação no gráfico da Figura 3.11 que para os testes utilizando a similaridade Jaro-Winkler (JW) foram obtidos como menor e maior valores de cobertura 70,80% e 96,00%.

A cobertura mínima de 70,80% obtida entre as configurações que utilizam a conversão DM com ED máximo igual a 0 indica o efeito positivo da utilização da conversão. A redução no tamanho da trie e da criação de prefixos iguais promovidos pela utilização da função de

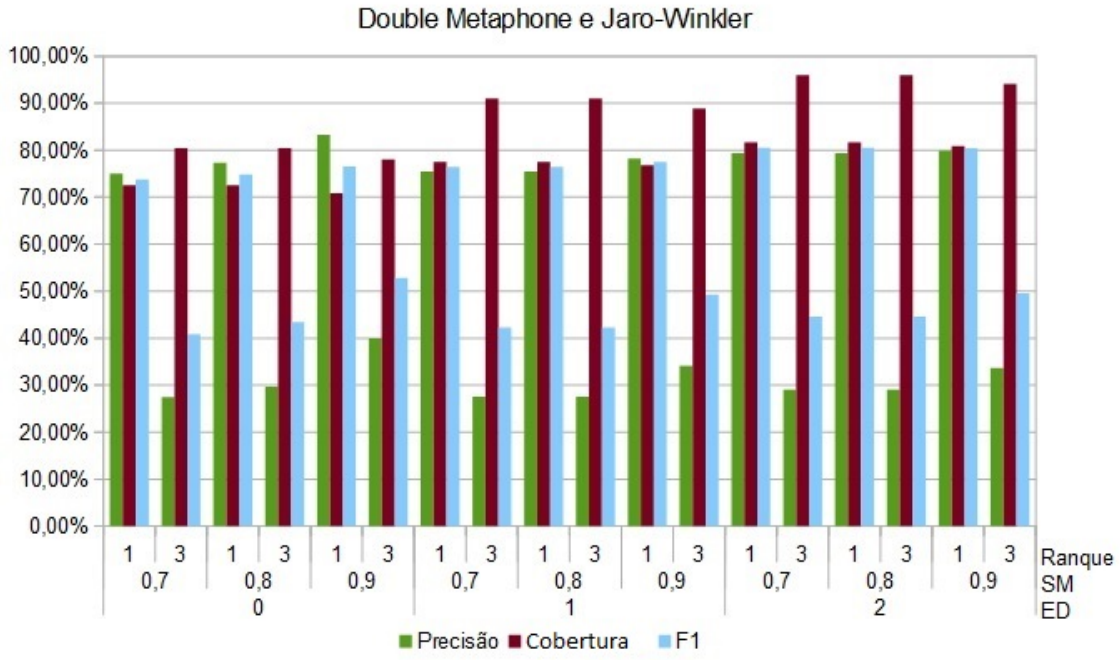

Figura 3.11: Precisão, cobertura e F1 dos testes utilizando transformação Double Metaphone e a métrica de similaridade Jaro-Winkler (Tabela 3.3)

.

conversão permitiram à busca exata identificar entidades com erros. Como todas as palavras presentes no arquivo texto contém pelo menos 1 erro, as buscas com ED=0 não deveriam retornar nenhuma associação. Isso é confirmado pelos testes realizados sem conversão (Figuras 3.13 e 3.14) e não identificaram nenhuma entidade ao utilizar ED=0. Resultado positivo uma vez que as configurações que utilizam a conversão DM com ED=0 representam as execuções mais rápidas. Outro resultado positivo do uso do módulo de conversão foi a obtenção de valores de F1 Score de até 80,04% quando utilizando ED=0, ficando apenas 8 pontos percentuais abaixo do melhor F1 Score dentre todos.

3.5.4 Resultados para alterações na filtragem

A métrica de similaridade utilizada para filtrar as associações, juntamente com valores adequados de configuração da mesma, impactam no desempenho do sistema. Nos testes são observadas variações para F1 Score superiores a 40%, causadas por mudanças da métrica de similaridade em associação com alterações nos valores de similaridade mínima e quantidade de melhores similaridades.

A etapa de filtragem produz alterações claras nos valores de cobertura e precisão. Os valores mais baixos obtidos para a precisão em ambos os módulos de conversão são referentes às configurações com similaridade mínima 0,7 e 0,8 e 3 maiores similaridades. A baixa precisão está relacionada com características das entradas e com o alívio nas regras de filtragem. Dicionários que contém entradas com 3 ou menos caracteres e entradas similares entre si acabam produzindo um elevado número de falsos positivos. Esse efeito foi agravado nas execuções utilizando a conversão DM por esta reduzir o tamanho das entradas e assim sendo mais dependente da etapa de filtragem.

O tempo para realização da filtragem está diretamente relacionado com a quantidade de associações produzidas na busca sobre a trie e o tempo gasto pela métrica na filtragem para

Tabela 3.4: Resultados das 18 configurações utilizando a conversão Double metaphone (DM) e similaridade String$_{Sim}$ (SS).

ED	Similaridade Mínima	Maiores Similaridades	Precisão	Cobertura	F1	Tempo Árvore (ms)	Tempo Busca (ms)	Tempo métrica Similaridade (ms)
0	0,7	3	30,19%	80,60%	43,92%	1.010	74	518
0	0,7	1	79,96%	75,80%	77,82%	864	65	511
0	0,8	3	39,33%	80,00%	52,74%	867	51	476
0	0,8	1	85,04%	75,60%	80,04%	866	52	469
0	0,9	3	75,97%	62,90%	68,82%	880	47	541
0	0,9	1	**90,12%**	61,10%	72,82%	1.092	59	565
1	0,7	3	28,17%	91,60%	43,09%	962	439	3.846
1	0,7	1	81,45%	84,30%	82,85%	947	393	3.031
1	0,8	3	32,23%	91,20%	47,62%	905	365	3.095
1	0,8	1	82,87%	84,20%	83,53%	824	387	3.126
1	0,9	3	69,43%	71,30%	70,35%	904	405	3.418
1	0,9	1	87,90%	69,00%	77,31%	837	397	3.093
2	0,7	3	29,18%	**97,10%**	44,87%	956	2.347	21.632
2	0,7	1	84,61%	88,50%	86,51%	856	2.625	21.425
2	0,8	3	31,35%	96,80%	47,36%	864	2.247	21.588
2	0,8	1	84,76%	88,40%	**86,54%**	896	2.394	21.691
2	0,9	3	66,37%	75,80%	70,77%	868	2.465	21.536
2	0,9	1	87,41%	72,90%	79,50%	838	2.381	21.930

classificar cada associação. Uma trie construída utilizando entradas curtas combinada com uma busca que aceita alto grau de erro pode resultar em muitas associações falsas e assim elevar o tempo gasto para a filtragem.

Outro fator a ser observado é a variação do F1 Score para as duas métricas de similaridade utilizadas. A métrica String$_{Sim}$ obteve os maiores valores entre as execuções com e sem a utilização de conversão. O melhor resultado de F1 Score obtido entre as execuções com uso do Double metaphone foi 6 pontos percentuais maior quando utilizada a métrica String$_{Sim}$ em comparação com o maior resultado obtido pela Jaro-Winkler. Para as execuções sem uso de conversão o maior resultado da String$_{Sim}$ superou em quase 5 pontos percentuais o maior resultado da Jaro-winkler.

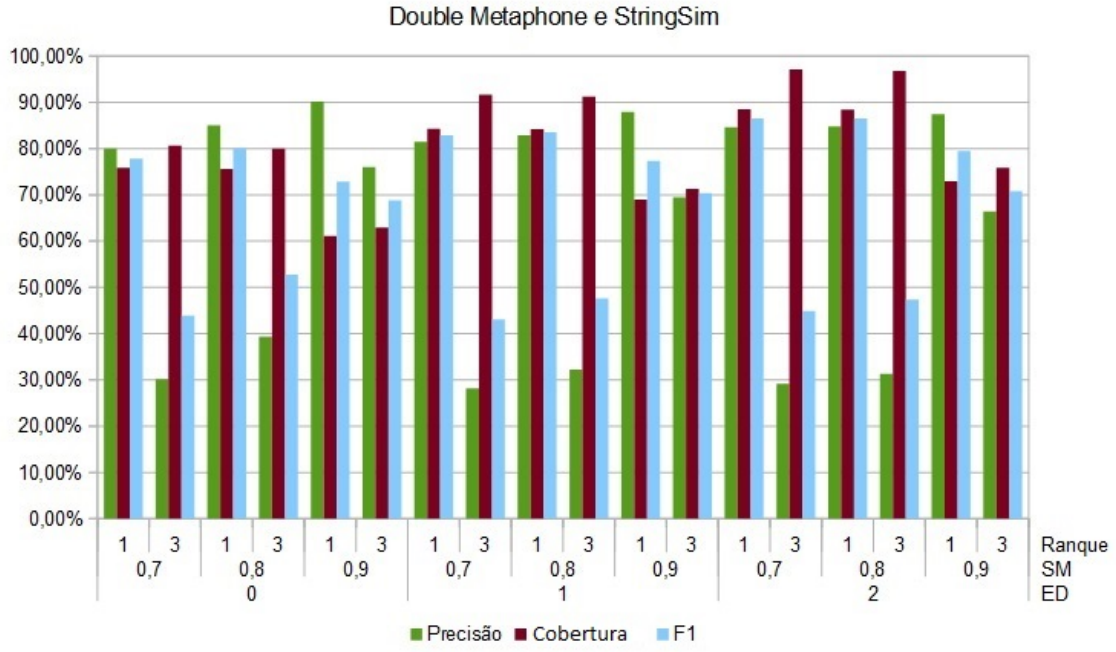

Figura 3.12: Precisão, cobertura e F1 dos testes utilizando transformação Double Metaphone e a métrica de similaridade String_{Sim} (Tabela 3.4).

Tabela 3.5: Resultados das 18 configurações sem conversão e similaridade Jaro-Winkler (JW).

ED	Similaridade Mínima	Maiores Similaridades	Precisão	Cobertura	F1	Tempo Árvore (ms)	Tempo Busca (ms)	Tempo métrica Similaridade (ms)
0	0,7	1	0,00%	0,00%	0,00%	1.188	125	0
0	0,7	3	0,00%	0,00%	0,00%	1.109	141	0
0	0,8	1	0,00%	0,00%	0,00%	1.125	125	0
0	0,8	3	0,00%	0,00%	0,00%	1.110	140	0
0	0,9	3	0,00%	0,00%	0,00%	1.125	156	16
0	0,9	1	0,00%	0,00%	0,00%	1.109	172	0
1	0,7	3	74,71%	77,70%	76,18%	1.188	625	47
1	0,7	1	87,15%	71,90%	78,79%	1.141	625	47
1	0,8	3	74,71%	77,70%	76,18%	1.094	593	47
1	0,8	1	87,15%	71,90%	78,79%	1.125	610	62
1	0,9	3	77,38%	75,60%	76,48%	1.110	578	63
1	0,9	1	**87,16%**	70,60%	78,01%	1.094	578	62
2	0,7	3	41,43%	**95,90%**	57,86%	1.156	3.422	78
2	0,7	1	83,20%	84,20%	**83,70%**	1.172	3.359	78
2	0,8	3	41,77%	**95,90%**	58,19%	1.125	3.376	78
2	0,8	1	83,20%	84,20%	**83,70%**	1.109	2.922	78
2	0,9	3	49,89%	93,50%	65,07%	1.141	3.859	79
2	0,9	1	83,28%	82,70%	82,99%	1.109	3.000	62

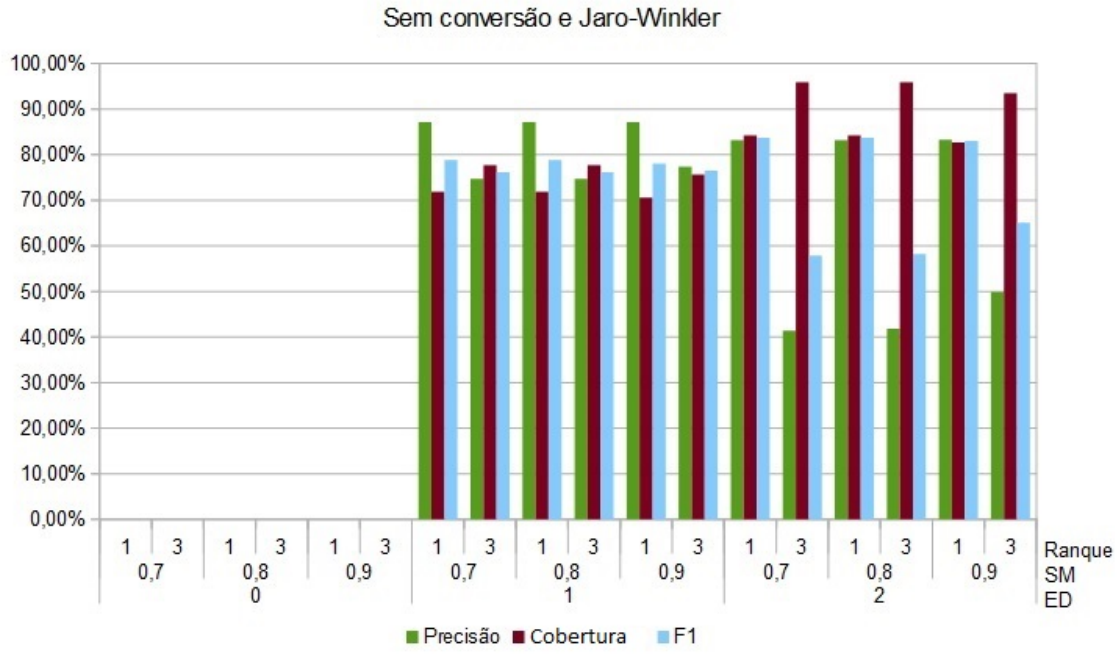

Figura 3.13: Precisão, cobertura e F1 dos testes sem transformação e com a métrica de similaridade Jaro-Winkler (Tabela 3.5).

Tabela 3.6: Resultados das 18 configurações sem conversão e similaridade String$_{Sim}$ (SS).

ED	Similaridade Mínima	Maiores Similaridades	Precisão	Cobertura	F1	Tempo Árvore (ms)	Tempo Busca (ms)	Tempo métrica Similaridade (ms)
0	0,7	1	0,00%	0,00%	0,00%	1.141	156	0
0	0,7	3	0,00%	0,00%	0,00%	1.141	156	16
0	0,8	1	0,00%	0,00%	0,00%	1.094	141	0
0	0,8	3	0,00%	0,00%	0,00%	1.093	157	0
0	0,9	1	0,00%	0,00%	0,00%	1.125	140	0
0	0,9	3	0,00%	0,00%	0,00%	1.094	156	16
1	0,7	3	74,78%	77,70%	76,21%	1.172	563	93
1	0,7	1	89,30%	74,30%	81,11%	1.109	610	78
1	0,8	3	75,73%	77,70%	76,70%	1.078	594	94
1	0,8	1	89,52%	74,30%	81,20%	1.148	625	78
1	0,9	3	85,79%	67,00%	75,24%	1.062	578	94
1	0,9	1	**91,84%**	65,30%	76,33%	1.078	578	94
2	0,7	3	42,66%	**96,50%**	59,17%	1.078	3.157	172
2	0,7	1	87,38%	89,30%	**88,33%**	1.125	3.296	141
2	0,8	3	45,66%	96,30%	61,95%	1.188	3.359	141
2	0,8	1	87,35%	89,10%	88,22%	1.109	2.979	141
2	0,9	3	73,10%	76,10%	74,57%	1.078	2.969	125
2	0,9	1	89,06%	73,30%	80,42%	1.094	3.328	125

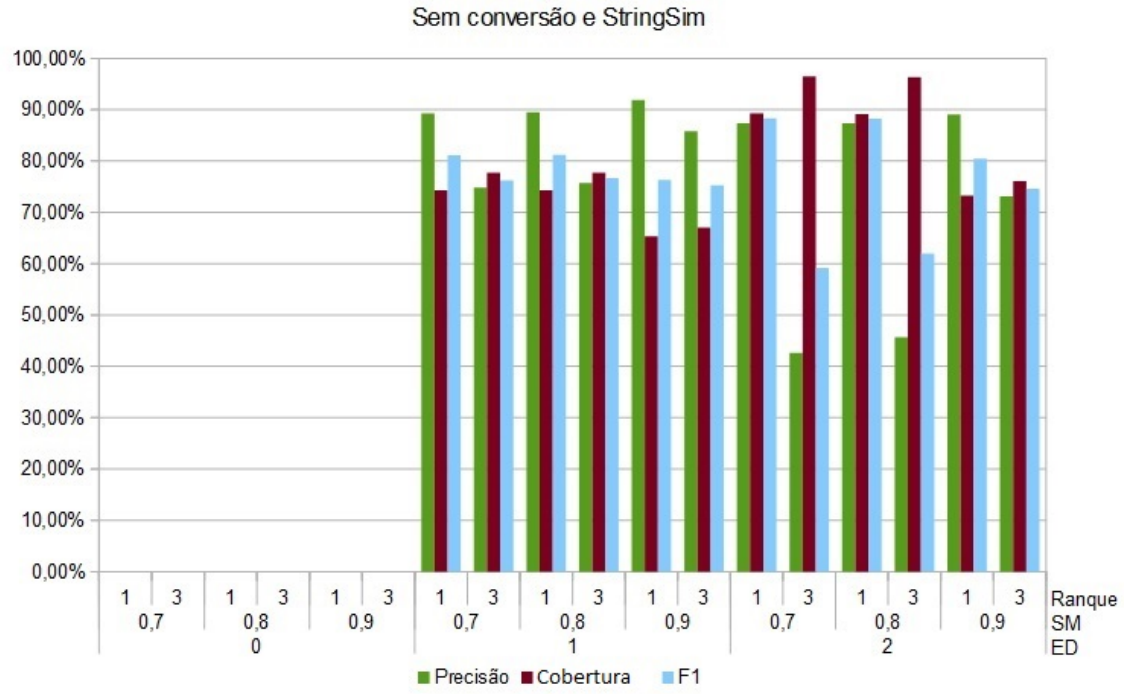

Figura 3.14: Precisão, cobertura e F1 dos testes sem transformação e com a métrica de similaridade String_{Sim} (Tabela 3.6).

Capítulo 4

Conclusão

Nessa dissertação é apresentada uma abordagem de reconhecimento de entidades nomeadas através de associação aproximada entre sequências de caracteres e que é adaptável através da modularização de funcionalidades e definição de parâmetros de configuração. Essa solução procura reduzir os impactos de erros ortográficos no processo de extração de informação de fontes não estruturadas.

As contribuições dessa dissertação são uma abordagem de NER adaptável ao caso de estudo, com redução no tamanho da trie que indexa o dicionário através de transformações das SCs. Uma busca por aproximação com a capacidade de reconhecer entidades nomeadas utilizando menores valores de ED do que o necessário em comparações sem transformação das SCs envolvidas e assim diminuir o tempo da busca sobre o dicionário. Adaptações na estrutura da trie para ser possível trabalhar com SCs transformadas e armazenar os rótulos de anotação das entidades e a alteração da busca por ASM para considerar *tokens* completos, permitiram utilizar a busca por conjunto de nós ativos no contexto de NER.

Os módulos de conversão e similaridade permitem a utilização de diferentes técnicas de transformação e métricas de similaridade para adaptar a abordagem ao caso de estudo de interesse. Nos experimentos a combinação de técnicas de transformação fonética para o módulo de conversão com a utilização de uma árvore de prefixos (trie) para indexar o dicionário, proporcionou economia de espaço em memória e assim permite indexar dicionários maiores em comparação com a utilização das SCs originais que representam as entradas. Outra contribuição produzida pelo uso de transformação fonética foi a redução do erro entre entradas e entidades permitindo localizar entidades aceitando um erro menor do que o erro presente na sequência original de caracteres.

Os resultados obtidos indicam alteração de desempenho para diferentes configurações de execução e assim comprovando que a personalização que pode ser realizada na abordagem proporciona a capacidade de priorizar diferentes características das entradas e dos textos, com o objetivo de obter os melhores valores de precisão e cobertura para determinado contexto. A utilização do Double metaphone como recurso de conversão reduziu o tamanho da trie para menos de 5% do tamanho original e conseguiu com a busca exata valores de até 85,04% para precisão com cobertura de 75,60% e F1 Score de 80,04% para arquivos de entrada em que a busca exata sem conversão não reconheceu nenhuma entidade. Porém o maior F1-score obtido sem o uso de conversão foi 2 pontos percentuais superior ao maior F1 Score com utilização da conversão fonética.

Os experimentos também nos permitem concluir que as transformações utilizadas no módulo de conversão devem conseguir eliminar os erros existentes nas entidades para que seja possível reconhecer o maior número de entidades com valores de ED abaixo de 2. Os testes que

utilizaram 0 (zero) ou 1 (um) como valores de ED, demonstram um desempenho melhor para as configurações que utilizam a conversão fonética. O aumento do valor de ED tem impacto negativo maior quando a abordagem utiliza conversão. Como as entradas ficam menores e mais similares entre si quando utilizada a conversão, a quantidade de associações candidatas erradas aumenta mais para cada aumento do valor de ED em comparação com as configurações sem conversão. Isso valoriza a característica de troca dos módulos de conversão e cobertura, para opções mais adequadas aos textos do caso de estudo.

4.1 Trabalhos futuros

Foram identificadas possibilidades para trabalhos futuros com a finalidade de aperfeiçoar e ampliar os recursos atuais dessa abordagem de NER bem como avaliar a mesma em diversas áreas específicas de estudo.

A capacidade de configurar a utilização de módulos de conversão e similaridade permitiu ao reconhecedor de entidades nomeadas apresentado obter resultados variados, o que indica capacidade de adaptação à diferentes casos de estudo. Assim a modularização de outros processos podem contribuir para uma maior flexibilidade da abordagem. Uma atividade que pode contribuir com a adaptabilidade da abordagem ao ser disponibilizada na forma de módulo é a divisão em *tokens*. Sendo essencial para utilização em diferentes idiomas.

A filtragem tem papel fundamental para melhorar os resultados através da eliminação de entidades falsas. Para permitir uma melhor filtragem, a escolha dos melhores resultados que hoje utiliza o valor da métrica de similaridade, pode ser expandido para analisar o conjunto de associações realizadas observando outras características. Algumas opções são o uso de estatística para definir qual das entradas mais apareceu para aquele texto ou um recurso com aprendizado de máquina treinado para decidir qual a entrada que melhor representa aquela sequência de caracteres. Esses recursos podem ser usados para ajudar nos casos em que os valores de similaridade de várias associações de uma mesma entidade forem muito próximos ou iguais.

A aplicação da nossa abordagem de NER sobre documentos de domínios específicos para verificar os resultados que os recursos de configuração e a utilização de diferentes técnicas de conversão e similaridade permitem obter em comparação com abordagens específicas daquele domínio. Um exemplo são os registros médicos que contém referências de drogas prescritas e nomes de doenças.

O tempo de execução da nossa abordagem pode ser diminuído com alterações no processamento das atividades para utilizar todo o potencial de processadores com múltiplos núcleos. Hoje é necessário localizar todas as associações candidatas do texto para então realizar a filtragem. Nos experimento percebemos que o uso do processador não passou dos 30%, então alterações para que a busca e a filtragem sejam realizadas de forma paralela em um processador com múltiplos núcleos pode diminuir o tempo da execução.

Referências Bibliográficas

[Al-Jumaily et al., 2012] Al-Jumaily, H., Martínez, P., Martínez-Fernández, J. e Van der Goot, E. (2012). A real time named entity recognition system for arabic text mining. *Language Resources and Evaluation*, 46(4):543–563.

[Al-Turaiki et al., 2012] Al-Turaiki, I., Badr, G. e Mathkour, H. (2012). Trie-based apriori motif discovery approach. Em Bleris, L., Mandoiu, I., Schwartz, R. e Wang, J., editores, *Bioinformatics Research and Applications*, volume 7292 de *Lecture Notes in Computer Science*, páginas 1–12. Springer Berlin Heidelberg.

[Arasu et al., 2009] Arasu, A., Chaudhuri, S. e Kaushik, R. (2009). Learning string transformations from examples. *Proc. VLDB Endow.*, 2(1):514–525.

[Barron-Cedeno et al., 2010] Barron-Cedeno, A., Rosso, P., Agirre, E. e Labaka, G. (2010). Plagiarism detection across distant language pairs. Em *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, páginas 37–45, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Bergroth et al., 2000] Bergroth, L., Hakonen, H. e Raita, T. (2000). A survey of longest common subsequence algorithms. Em *String Processing and Information Retrieval, 2000. SPIRE 2000. Proceedings. Seventh International Symposium on*, páginas 39–48.

[Bird et al., 2009] Bird, S., Klein, E. e Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edition.

[Boulaknadel et al., 2014] Boulaknadel, S., Talha, M. e Aboutajdine, D. (2014). Amazighe named entity recognition using a a rule based approach. Em *Computer Systems and Applications (AICCSA), 2014 IEEE/ACS 11th International Conference on*, páginas 478–484.

[Chiticariu et al., 2013] Chiticariu, L., Li, Y. e Reiss, F. R. (2013). Rule-based information extraction is dead! long live rule-based information extraction systems! Em *EMNLP*, páginas 827–832.

[Cunningham, 2005] Cunningham, H. (2005). Information Extraction, Automatic. *Encyclopedia of Language and Linguistics, 2nd Edition*.

[Cunningham et al., 2002] Cunningham, H., Maynard, D., Bontcheva, K. e Tablan, V. (2002). Gate: An architecture for development of robust hlt applications. Em *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, páginas 168–175, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Cunningham et al., 2014] Cunningham, H., Tablan, V., Bontcheva, K., Maynard, D. e Wilks, Y. (2014). *Developing Language Processing Components with GATE Version 8*. University of Sheffield Department of Computer Science.

[Deng et al., 2016] Deng, D., Li, G., Wen, H., Jagadish, H. e Feng, J. (2016). Meta: An efficient matching-based method for error-tolerant autocompletion. *Proceedings of the VLDB Endowment*, 9(10).

[Gerner et al., 2010] Gerner, M., Nenadic, G. e Bergman, C. M. (2010). Linnaeus: a species name identification system for biomedical literature. *BMC bioinformatics*, 11(1):85.

[Ghodsi, 2009] Ghodsi, M. (2009). Approximate string matching using backtracking over suffix arrays.

[Gomaa e Fahmy, 2013] Gomaa, W. H. e Fahmy, A. A. (2013). A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13):13–18.

[Goutte e Gaussier, 2005] Goutte, C. e Gaussier, E. (2005). *A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation*, páginas 345–359. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Grishman e Sundheim, 1996] Grishman, R. e Sundheim, B. (1996). Message understanding conference-6: A brief history. Em *COLING*, volume 96, páginas 466–471.

[Hakenberg et al., 2011] Hakenberg, J., Gerner, M., Haeussler, M., Solt, I., Plake, C., Schroeder, M., Gonzalez, G., Nenadic, G. e Bergman, C. M. (2011). The gnat library for local and remote gene mention normalization. *Bioinformatics*, 27(19):2769–2771.

[Hemdev, 2011] Hemdev, P. (2011). *Information Extraction: A Smart Calendar Application*. VDM Verlag Dr. Müller.

[Ji et al., 2009] Ji, S., Li, G., Li, C. e Feng, J. (2009). Efficient interactive fuzzy keyword search. Em *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, páginas 371–380, New York, NY, USA. ACM.

[Kanis e Skorkovská, 2010] Kanis, J. e Skorkovská, L. (2010). *Comparison of Different Lemmatization Approaches through the Means of Information Retrieval Performance*, páginas 93–100. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Leaman et al., 2008] Leaman, R., Gonzalez, G. et al. (2008). Banner: an executable survey of advances in biomedical named entity recognition. Em *Pacific Symposium on Biocomputing*, volume 13, páginas 652–663. Citeseer.

[Levenshtein, 1966] Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. Em *Soviet physics doklady*, volume 10, páginas 707–710.

[Li et al., 2012] Li, C., Weng, J., He, Q., Yao, Y., Datta, A., Sun, A. e Lee, B.-S. (2012). Twiner: Named entity recognition in targeted twitter stream. Em *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, páginas 721–730, New York, NY, USA. ACM.

[Lovins, 1968] Lovins, J. B. (1968). *Development of a stemming algorithm*. MIT Information Processing Group, Electronic Systems Laboratory Cambridge.

[Maedche et al., 2002] Maedche, A., Neumann, G. e Staab, S. (2002). *Bootstrapping an Ontology-Based Information Extraction System*. Physica-Verlag HD, 1st edition.

[Marzal e Vidal, 1993] Marzal, A. e Vidal, E. (1993). Computation of normalized edit distance and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):926–932.

[Maynard et al., 2002] Maynard, D., Cunningham, H., Bontcheva, K. e Dimitrov, M. (2002). *Adapting a Robust Multi-genre NE System for Automatic Content Extraction*, páginas 264–273. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Maynard et al., 2003] Maynard, D., Tablan, V., Bontcheva, K., Cunningham, H. e Wilks, Y. (2003). Muse: a multisource entity recognition system. *Submitted to Computers and the Humanities*.

[Moens, 2006] Moens, M.-F. (2006). *Information extraction: algorithms and prospects in a retrieval context*, volume 21. Springer Science & Business Media.

[Nadeau e Sekine, 2007] Nadeau, D. e Sekine, S. (2007). A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26.

[Navarro, 2001] Navarro, G. (2001). A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88.

[Navarro et al., 2003] Navarro, G., Baeza-Yates, R. e Marcelo Azevedo Arcoverde, J. (2003). Matchsimile: A flexible approximate matching tool for searching proper names. *Journal of the American Society for Information Science and Technology*, 54(1):3–15.

[Navarro et al., 2001] Navarro, G., Baeza-Yates, R. A., Sutinen, E. e Tarhio, J. (2001). Indexing methods for approximate string matching. *IEEE Data Eng. Bull.*, 24(4):19–27.

[Oommen e Badr, 2004] Oommen, B. e Badr, G. (2004). Dictionary-based syntactic pattern recognition using tries. Em Fred, A., Caelli, T., Duin, R., Campilho, A. e de Ridder, D., editores, *Structural, Syntactic, and Statistical Pattern Recognition*, volume 3138 de *Lecture Notes in Computer Science*, páginas 251–259. Springer Berlin Heidelberg.

[Oommen e Badr, 2007] Oommen, B. e Badr, G. (2007). Breadth-first search strategies for trie-based syntactic pattern recognition. *Pattern Analysis and Applications*, 10(1):1–13.

[Peterson, 1986] Peterson, J. L. (1986). A note on undetected typing errors. *Commun. ACM*, 29(7):633–637.

[Philips, 2000] Philips, L. (2000). The double metaphone search algorithm. *C/C++ Users J.*, 18(6):38–43.

[Pradhan e Negi, 2014] Pradhan, S. e Negi, A. (2014). An improved approach of dictionary based syntactic pr using trie. Em *Electronic Systems, Signal Processing and Computing Technologies (ICESC), 2014 International Conference on*, páginas 386–391.

[QUEIROZ, 2005] QUEIROZ, R. (2005). A informação escrita: do manuscrito ao texto virtual.

[Ritter et al., 2011] Ritter, A., Clark, S., Mausam e Etzioni, O. (2011). Named entity recognition in tweets: An experimental study. Em *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, páginas 1524–1534, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Rocktäschel et al., 2012] Rocktäschel, T., Weidlich, M. e Leser, U. (2012). Chemspot: a hybrid system for chemical named entity recognition. *Bioinformatics*, 28(12):1633–1640.

[Russell, 1918] Russell, R. (1918). Index. US Patent 1,261,167.

[Sarawagi, 2008] Sarawagi, S. (2008). Information extraction. *Found. Trends databases*, 1(3):261–377.

[Shang e Merrettal, 1996] Shang, H. e Merrettal, T. (1996). Tries for approximate string matching. *Knowledge and Data Engineering, IEEE Transactions on*, 8(4):540–547.

[Tissot et al., 2015] Tissot, H., dos Santos, C. F. H., Gorrell, G., Roberts, A., Derczynski, L. e Del Fabro, M. D. (2015). Ufprsheffield: Contrasting rule-based and support vector machine approaches to time expression identification in clinical tempeval. *SemEval-2015*, página 835.

[Tissot et al., 2014] Tissot, H., Peschl, G. e Del Fabro, M. D. (2014). *Fast Phonetic Similarity Search over Large Repositories*, páginas 74–81. Springer International Publishing, Cham.

[van Berkel e De Smedt, 1988] van Berkel, B. e De Smedt, K. (1988). Triphone analysis: A combined method for the correction of orthographical and typographical errors. Em *Proceedings of the Second Conference on Applied Natural Language Processing*, ANLC '88, páginas 77–83, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Viola, 2006] Viola, A. C. T. K. A. M. P. (2006). Corrective feedback and persistent learning for information extraction. *Artificial Intelligence*, 170(14):1101–1122.

[Wang e Patrick, 2009] Wang, Y. e Patrick, J. (2009). Cascading classifiers for named entity recognition in clinical notes. Em *Proceedings of the Workshop on Biomedical Information Extraction*, WBIE '09, páginas 42–49, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Wimalasuriya e Dou, 2010] Wimalasuriya, D. C. e Dou, D. (2010). Ontology-based information extraction: An introduction and a survey of current approaches. *Journal of Information Science*.

[Yancey, 2005] Yancey, W. E. (2005). Evaluating string comparator performance for record linkage. *Statistical Research Division Research Report, http://www. census. gov/srd/papers/pdf/rrs2005-05. pdf.*

[Zhou e Su, 2002] Zhou, G. e Su, J. (2002). Named entity recognition using an hmm-based chunk tagger. Em *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, páginas 473–480, Stroudsburg, PA, USA. Association for Computational Linguistics.