OSVALDO MARCIO CAVALIERI

UM MÉTODO COMPLEMENTAR AO PROCESSO DE SANITIZAÇÃO DE REGISTROS DUPLICADOS EM BASES DE DADOS CADSUS-MULTIPLATAFORMA

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Marcos Sfair Sunye
Coorientador: Prof. Dr. Bruno Müller Junior

CURITIBA

2014

C376m Cavalieri, Osvaldo Marcio
Um método complementar ao processo de sanitização de registros duplicados em bases de dados cadsus-multiplataforma / Osvaldo Marcio Cavalieri . – Curitiba, 2014
84 f. : il.; tabs.

Dissertação (mestrado) – Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática.
Orientador: Marcos Sfair Sunye. - Coorientador: Bruno Müller Junior

1. Banco de dados. 2. Documentos eletrônicos - Duplicidade.
I. Sunye, Marcos Sfair. II.Müller Junior, Bruno. III. Título.

CDD: 005.74

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, do aluno Osvaldo Marcio Cavalieri, avaliamos o trabalho intitulado, *"Um método complementar ao processo de sanitização de registros duplicados em bases de dados Cadsus-Multiplataforma"*, cuja defesa foi realizada no dia 17 de julho de 2014, às 14:00 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela: (✓)**aprovação** do candidato. ( )**reprovação** do candidato.

Curitiba, 17 de julho de 2014.

Prof. Dr. Marcos Sfair Sunye
DINF/UFPR – Orientador

Prof. Dr. Bruno Müller Junior
DINF/UFPR – Coorientador

MINISTÉRIO DA EDUCAÇÃO - UNIVERSIDADE FEDERAL DO PARANÁ - Setor de Ciências Exatas - Coordenação de Pós-Graduação em Informática

Profa. Dra. Luciana Schleder Gonçalves
DENF/UFPR – Membro Externo

Profa. Dra. Leticia Mara Peres
DINF/UFPR – Membro Interno

SUMÁRIO

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE FIGURAS

LISTA DE ABREVIAÇÕES

ANUOS LICENCE - Australian National University Open Source License

API - Application Programming Interface

CADSUS - Cartão SUS

CEP - Código de Endereçamento Postal

CPF - Código de Pessoa Física

CSV - Comma Separated Values

DUDE - Duplicate Detection

DTS - Data Transformation Services

FEBRL - Freely Extensible Biomedical Record Linkage

FRIL - Fine-Grained Records Integration and Linkage

IBGE - Instituto Brasileiro de Geografia e Estatística

INAMPS - Instituto Nacional de Assistência Médica da Previdência Social

ODBC - Open Database Connectivity

SUS - Sistema Único de Saúde

UBS - Unidade Básica de Saúde

XML- eXtensible Markup Language

SVM- Suport Vector Machine

TF-IDF - Term-Frequency / Inverse Document Frequency

RESUMO

Diante o recente crescimento no volume de dados e queda dos preços de armazenamento, dados duplicados poderiam não representar problemas, apenas uso desnecessário de recursos. Porém, dependendo do contexto, por exemplo, na área da saúde, registros duplicados devem ser evitados pois podem causar sérios danos. Entre os aplicativos utilizados na gestão do Sistema Único de Saúde (SUS), o aplicativo CADSUS-multiplataforma, entre seus objetivos, visa evitar que existam cadastros de usuários duplicados, porém, este objetivo não é de todo cumprido. Este trabalho contextualiza o problema da duplicação na área da saúde, apresenta conceitos sobre qualidade de dados, descreve de maneira geral algumas ferramentas para deduplicação (identificação de duplicidades) e apresenta um processo de deduplicação e a aplicação destes numa base de dados CADSUS-multiplataforma. O processo utilizado segue o modelo proposto por Peter Christen, para auxiliar a execução é utilizada a ferramenta FEBRL - *Freely Extensible Biomedical Record Linkage* que o suporta, ainda, foram analisados cadastros reais de uma base de dados CADSUS de um município. O trabalho resultou na análise de 238.691 cadastros, destes 13,98% foram classificados como duplicidades efetivas e 0,40% como possíveis duplos. Dado o tempo para execução da deduplicação, aproximadamente 37 horas, o alto número de duplicidades encontradas, 33.368, e considerando a pequena quantidade de possíveis duplos, 973 cadastros, e ainda, o procedimento para eliminar as duplicidades no CADSUS, que deve ser feito manualmente, cadastro a cadastro, a utilização do processo de deduplicação para detectar os similares pode ser considerada viável, pois, sem aquele, a identificação dos cadastros similares também teria que ser manual, tornando o processo possivelmente impraticável.

ABSTRACT

Facing the recent growth in data volume and decreasing on the prices of storage, duplicate data may not represent problems, maybe only and unnecessary use of resources. However, depending on the context, e.g. in health, duplicate records should be avoided because it can cause serious damage. Among the applications used in the management of the Unified Health System (SUS), the CADSUS-multiplataforma aims, among other objectives, avoid duplicate user entries, but, this is not completely fulfilled.This work contextualizes the problem of duplication in health, presents concepts on data quality, describes some tools for deduplication (identification of duplicates) and expose a deduplication process and its application in CADSUS data base. To analyze real entries of a CADSUS municipal data base, the process described on this document follows the model proposed by Peter Christen. To assist the implementation of the model, the FEBRL - Freely Extensible Biomedical Record Linkage tool was used. From the analysis of 238.691 entries, 13,98% of them were classified as effective duplicates and 0,40 % as possible duplicates. Given the time for implementing deduplication , approximately 37 hours, the great amount of duplicates (33.368), and considering the small amount of possible duplicates (973 entries) even considering the procedure to eliminate duplications in CADSUS - which must be done manually, entry by entry - the use of the deduplication process to detect the alike can be considered viable. Without identification of similar entries, this process also have to be performed manually, possibly making the process impossible to be done.

CAPÍTULO 1

INTRODUÇÃO

Nos últimos anos observa-se a popularização da área de tecnologia da informação. A evolução é contínua e uma das consequências diretas é o grande crescimento no volume de dados digitais. De forma geral, em todas as áreas, o volume de dados dobra a cada dois anos e pequisa recente [25] indica que a quantidade de dados criados e duplicados atingiu a casa de 1.8 zettabytes, ou seja, 1.8 trilhões de gigabytes . Em paralelo, o custo de armazenamento é cada vez menor, por exemplo, o custo por gigabyte em 2005 era aproximadamente U$ 15 dólares, em 2010 este custo estava abaixo de U$ 5 dólares [25].

Na área de TI aplicada em saúde, o ritmo é o mesmo. O uso de registros médicos em meio eletrônico e a utilização de imagens médicas, grandes em tamanho, dada a alta resolução, gerou uma explosão no volume de dados do paciente. Estima-se que nos últimos cinco anos, em hospitais dos Estados Unidos, aconteceu um crescimento de aproximadamente vinte por cento ao ano no arquivamento de imagens, entre outros, atingindo algo em torno de 27.000 terabytes, ou seja, 27 milhões de gigabytes em 2011[5].

Na saúde pública brasileira, a realidade não é diferente. Nos últimos anos, o governo brasileiro, por meio do Ministério da Saúde, tem desenvolvido diversas políticas que visam melhorar a qualidade de vida e saúde do cidadão. Via de regra, estas políticas são implementadas pelo SUS - Sistema Único de Saúde, que é a rede de saúde do estado, responsável por planejar e gerir os programas[1] sociais vinculados as políticas.

Para o acompanhamento da execução dos programas sociais foram criados vários aplicativos, tipicamente um aplicativo para cada programa. Gerando assim um significativo aumento no volume de dados, tanto por conta do armazenamento de todo o histórico médico de cada paciente a fim de cumprir metas, quanto da mínima integração entre os aplicativos, que por consequência armazenam dados em duplicidade.

[1]Para fins de padronização, neste documento, o termo "programa" indica "programa social", um plano a ser seguido, para indicar um "programa de computador" é convencionado o termo "aplicativo".

Este cenário, de fragmentação dos aplicativos e duplicidade dos dados do paciente é conhecido do Ministério da Saúde brasileiro. Data do ano de 1996 a iniciativa de criação do projeto Cartão Nacional de Saúde - Cartão SUS, que entre outros objetivos, visava "identificar o usuário do SUS, integrar informações e construir a base de dados de atendimentos em saúde."[13], além de propiciar aos gestores, "o acesso e a geração de informações individualizadas, referenciadas a diferentes bases territoriais, acompanhadas e controladas ao longo do tempo"[13].

De lá para cá, o projeto do Cartão SUS amadureceu e, para suportar o programa, foi desenvolvido um conjunto de aplicativos denominado CADSUS multiplataforma. Implantado em grande parte do território nacional, o CADSUS multiplataforma tornou-se uma ferramenta essencial na administração das Unidades Básicas de Saúde - UBS dos municípios. Porém, em alguns casos, apresenta inconsistência nos dados, por exemplo, cadastros duplicados[18].

Na área de saúde, dados duplicados são fontes de problemas. Solicitações de financiamento ou planejamentos de ações realizados com base em números totalizados em duplicidade, um único resultado de exame enviado para um endereço desatualizado, atrasos nos atendimentos devido a inconsistências dos dados, são exemplos que podem gerar resultados desastrosos.

A principal proposta deste trabalho é apresentar um método semi-automatizado para detecção de cadastros duplicados e possivelmente duplicados em base de dados CADSUS-multiplataforma. Para que os próprios operadores do aplicativo CADSUS tenham um processo que os ajude a detectar cadastros duplicados. Espera-se, secundariamente, de forma subjetiva, mostrar que o método utilizado pode ser utilizado para auxiliar na detecção de dados duplicados em uma infraestrutura assíncrona, ou seja, uma base secundária desconectada da base principal; funcione como um pré-filtro na inserção de dados, já indicando duplicidades nesta etapa; e ainda, possa ser utilizado na detecção de registros duplicados em qualquer base de dados que tenha o mesmo padrão dos registros CADSUS.

O texto está organizado da seguinte forma: o Capítulo 2 apresenta uma visão geral do Sistema Único de Saúde, com um resumo dos objetivos de uma unidade básica de

saúde, em seguida trata-se o assunto deduplicação, onde é também é descrito o processo de deduplicação proposto por Christen [10], quando são detalhadas, tanto as etapas do processo quanto as técnicas que cada etapa oferece. Em seguida é apresentado alguns aplicativos auxiliares ao processo de deduplicação. Por fim, é feita introdução em qualidade da informação e os defeitos que podem comprometê-la.

A problemática e consequências da duplicação de dados em bases de dados CADSUS são abordadas no Capítulo 3, que também apresenta evidências comprovando a existência de dados duplicados em bases de dados CADSUS. Já o Capítulo 4, descreve a proposta de metodologia de execução que este trabalho seguirá.

O Capítulo 5 trata o procedimento de deduplicação de uma base de dados CADSUS, as escolhas das técnicas em cada etapa e os resultados obtidos da deduplicação.

Encerrando o documento, no Capítulo 6 é apresentado a conclusão das atividades realizadas e possíveis oportunidades para trabalhos futuros.

CAPÍTULO 2

REVISÃO DE LITERATURA

Este Capítulo contextualiza o papel de uma unidade básica de saúde dentro do SUS. Apresenta uma revisão da literatura sobre deduplicação e faz considerações sobre qualidade da informação no contexto da localização de registros duplicados.

A seção 2.1 traz uma visão geral do histórico e estrutura do Sistema Único de Saúde - SUS, seguido da seção 2.1.1 que apresenta de maneira ampla os objetivos e estrutura de uma unidade básica de saúde. Também é apresentada a estrutura do CADSUS multiplataforma. O assunto deduplicação é tratado na seção 2.2, onde é feito um breve histórico das principais propostas para detecção de registros duplicados, seguido do detalhamento de um método atual para deduplicação. A seção 2.2.7 apresenta alguns aplicativos auxiliares ao processo de deduplicação. Por último, a seção 2.3 trata o tema qualidade dos dados, com foco nos critérios de qualidade mais relevantes para o contexto de deduplicação, apresenta também alguns defeitos nos dados que podem comprometer a qualidade.

2.1 Sistema Único de Saúde - SUS

Em termos gerais, o SUS "é a rede de saúde que reúne postos de saúde, ambulatórios, hospitais, laboratórios, enfim, todos os estabelecimentos públicos responsáveis por garantir o direito dos cidadãos a consultas, exames, internações e tratamentos de saúde." [19]

O artigo 196 da Constituição Federal Brasileira de 1988, define que "A saúde é direito de todos e dever do estado, garantido mediante políticas sociais e econômicas que visem à redução do risco de doença e de outros agravos e ao acesso universal e igualitário às ações e serviços para sua promoção, proteção e recuperação." Neste contexto, estado não figura somente como o governo federal, mas sim, o poder público que compreende: União, Estados, Distrito Federal e Municípios.

Até 1988, quando foi criado o SUS, quase todas as ações de promoção da saúde e

prevenção de doenças eram desenvolvidas quase que somente pelo Ministério da Saúde, através do Instituto Nacional de Assistência Médica da Previdência Social - Inamps[13]. Porém, com a criação do SUS, as responsabilidades sobre a saúde pública foi dividida nas diferentes esferas governamentais. A União é responsável pelas políticas nacionais, mas a implementação fica a cargo dos parceiros, isto é, estados, municípios, organizações não governamentais e iniciativa privada. Neste contexto, um dos principais papéis é o do município, pois ele é o responsável pela saúde pública de seus munícipes.

O governo federal, utiliza os dados obtidos a partir dos municípios, para fazer a gestão de suas iniciativas, em outras palavras, de maneira ampla, define as estratégias gerais para a saúde. Mas quem planeja as ações e efetivamente as executa, junto ao cidadão é o município.

2.1.1 Unidade Básica de Saúde - UBS

Dentro do SUS, várias entidades podem realizar atendimento para a população, porém, a porta de entrada para o SUS é a UBS ou, popularmente, posto de saúde. Nela devem ser realizados os atendimentos básicos em clínica geral, pediatria, enfermagem, odontologia e ginecologia, "Os principais serviços oferecidos pelas UBS são consultas médicas, inalações, injeções, curativos, vacinas, coleta de exames laboratoriais, tratamento odontológico, encaminhamentos para especialidades e fornecimento de medicação básica."[19]. Ainda, a UBS que é um centro de tratamento reativo, conforme a demanda da população, também tem caráter pró-ativo, por intermédio do Agente Comunitário de Saúde - ACS, que é um dos responsáveis em realizar a promoção de saúde, visitando e orientando os moradores da região vinculada a uma UBS.

É atribuição do ACS registrar os dados dos pacientes que ele visitou. Para auxiliá-lo nesta tarefa, o SUS disponibiliza um conjunto de aplicativos, que foram agrupados em um único aplicativo, O CADSUS multiplataforma, explicado a seguir.

2.1.2 CADSUS multiplataforma

O conjunto de aplicativos CADSUS multiplataforma, visa atender o programa Cartão Nacional de Saúde, que por sua vez, tem o objetivo fundamental: "a identificação unívoca do usuário do SUS e o acompanhamento do conjunto de atendimentos realizados pelo sistema de saúde, onde quer que eles aconteçam, através do acesso a uma base nacional de dados de saúde do cidadão.". [20]

Desenvolvido em linguagem java, o CADSUS multiplataforma está apto para funcionar com qualquer banco de dados SQL-ANSI, por exemplo: Oracle[1], SQL server[2] e Postgresql[3].

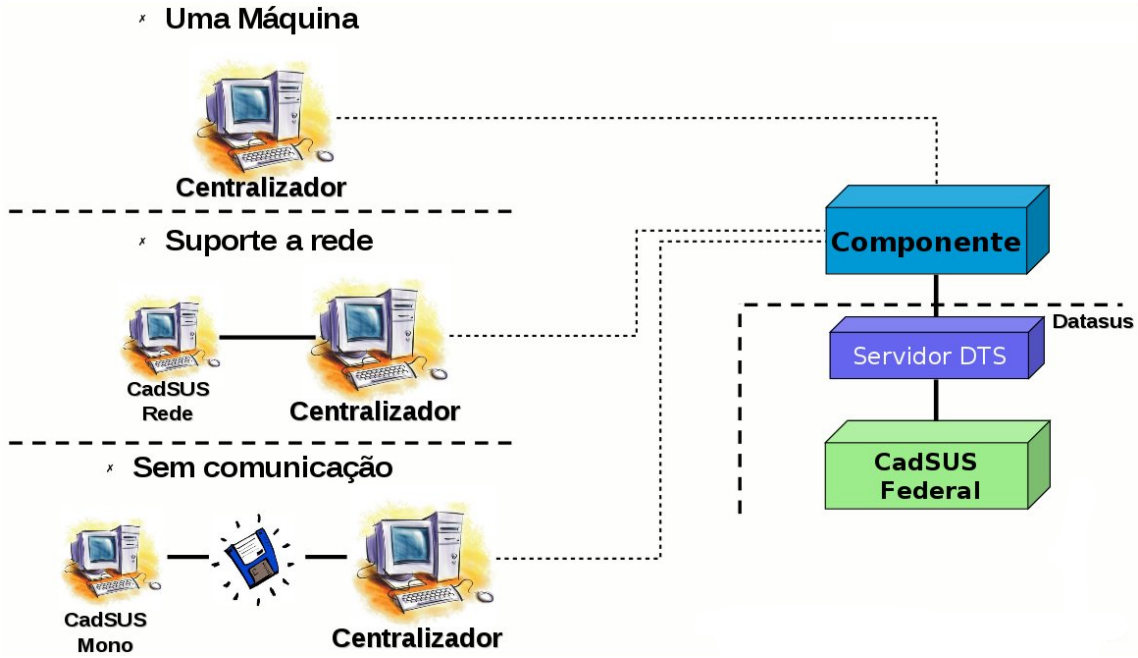

Figura 2.1: CADSUS Multiplataforma

Fonte: MS/Datasus - adaptado

Na Figura 2.1 é apresentada uma abstração dos aplicativos que compõe o CADSUS. Esta estrutura se justifica devido a quantidade de municípios e consequente diversidade das infraestruturas: municípios com um só computador, municípios com uma estrutura de rede completa ou municípios com computadores mas sem nenhuma estrutura de rede. Diante desta realidade o conjunto de aplicativos é organizado da seguinte forma:

[1]http://www.oracle.com/br
[2]http://www.microsoft.com/en-us/server-cloud/products/sql-server/
[3]http://www.postgresql.org/

- **CADSUS centralizador multiplataforma** - O CADSUS centralizador é o principal aplicativo do conjunto, tem por objetivo o cadastro, manutenção, e centralização dos dados municipais. Através de conexão em rede ou do envio de informações via algum tipo de mídia, centraliza o conteúdo de outras bases de dados CADSUS do município. Também pode ser conectado, via internet, na base de dados nacional, conhecida como "base federal", para, entre outras funcionalidades, realizar buscas por usuários na base nacional.

- **CADSUS municipal multiplataforma - rede** - Utilizado para cadastro e manutenção de dados dos usuários, o CADSUS municipal rede, necessita do CADSUS centralizador como repositório e fonte de informações. A partir do centralizador, pode fazer consultas, tanto na base municipal, quanto na base de dados federal.

- **CADSUS municipal multiplataforma - mono** - Desconectado do CADSUS centralizador, o CADSUS Municipal Mono destina-se ao cadastramento de usuários e domicílios. Dado a sua característica *standalone*, desconectado, periodicamente é necessário que os dados nele contidos, sejam exportados por meio de alguma mídia para a base de dados do CADSUS centralizador, para, se necessário, sofrerem algum tipo de manutenção.

2.2 Deduplicação

O objetivo do processo da deduplicação é identificar registros similares[4], ou seja, dois ou mais registros que potencialmente se referem a um único objeto no mundo real.

É interessante notar o uso da palavra similar ao invés da palavra igual, pois, dada uma abordagem determinística de comparação de registros que utilize um identificador unívoco ou busque por registros idênticos, as igualdades são facilmente detectáveis, porém, o mesmo não acontece para a identificação de registros similares.

Existem alguns estudos, que visam detectar duplicidades, baseados na similaridade entre registros. Geralmente o processo de detecção de duplicidades pode ser dividido em

[4]Dado peso 1 para registros iguais, 0 para diferentes, a similaridade pode ser entendida como um peso entre 0 e 1 indicando que os registros apresentam algum nível de igualdade.

dois grupos, abordagem supervisionada e abordagem não supervisionada [10]. A abordagem não supervisionada identifica pares ou grupos de registros como iguais baseando-se somente na similaridade entre eles, não considera qualquer outra informação que indique se os registros são iguais ou não. Já a abordagem supervisionada requer o treinamento de uma amostra de dados que seja significativa em relação ao universo de dados a serem analisados. A amostra deve conter pares similares ou não, identificados como tais, para que o código de aprendizagem utilize-os como base de conhecimento e assim a amostra seja utilizada posteriormente em comparações de registros na base principal.

Dado os objetivos deste trabalho, não é escopo descrever uma análise detalhada das propostas de deduplicação, no entanto, os fundamentos e o detalhamento dos métodos utilizados neste documento são descritos a seguir.

Um dos primeiros trabalhos visando identificar registros duplicados foi proposto por Newcombe et al [10]. Utilizando uma abordagem probabilística, a proposta era fazer uma avaliação fonética dos nomes e/ou sobrenomes, calculando um peso para cada atributo, comparando-os para definir se os registros seriam classificados como similares ou não similares. Ainda que inovadora para a época, a proposta foi classificada como intuitiva[22], por falhar na apresentação dos fundamentos estatísticos.

Com base na visão de Newcombe, os pesquisadores Ivan Fellegi e Alan Sunter, propuseram um processo de identificação de duplicidades fortemente sedimentado e que até o presente é utilizado como base para diversos outros [10]. Basicamente, o método consiste em comparar os dados fazendo a classificação dos registros e gerando como resultado três conjuntos de dados: A1 que contém os registros considerados similares, A2 com pares possivelmente similares ou duvidosos e A3 com os registros não similares.

Inicialmente, o modelo de Fellegi e Sunter, estabelece que os pares de registros obtidos a partir do produto cartesiano de dois conjuntos de dados A e B geram dois novos conjuntos de dados diferentes: os conjuntos M e U:

$M = \{(a, b) \in A \times B \mid a = b\}$

$U = \{(a, b) \in A \times B \mid a \neq b\}$

O primeiro, M, está relacionado a qualidade dos dados ou seja, a probabilidade de

dois atributos a e b indicarem o mesmo objeto real, porém diferindo entre si na forma como foram registrados [40].

O segundo, U, indica o conjunto dos pares que representam objetos distintos, está relacionado à frequência relativa de um atributo no conjunto de dados.

A comparação acontece da seguinte forma: dados dois registros **r**, estes são divididos em **i** atributos e cada par de atributos, um de A e outro de B é comparado entre si, utilizando uma regra de decisão que atribuirá um peso parcial p_i de similaridade m_i ou não similaridade u_i para o par. O cálculo do peso parcial baseia-se no uso de quatro probabilidades condicionais:

- Probabilidade de dois atributos iguais dado que referem-se a um único objeto ou $m_i = Prob[(a_i = b_i, a \in A, b \in B \mid r \in \mathbf{M}])$
- Probabilidade de dois atributos iguais dado que referem-se a objetos distintos ou $u_i = Prob[(a_i = b_i, a \in A, b \in B \mid r \in \mathbf{U}])$
- Probabilidade de dois atributos diferentes dado que referem-se a um único objeto ou $(1 - m_i)$
- Probabilidade de dois atributos diferentes dado que referem-se a objetos distintos ou $(1 - u_i)$

Como exemplo de aplicação destas probabilidades, para o cálculo de M, dado duas bases de dados A e B, que possuam um atributo mês de aniversário, com doze valores possíveis, isto é, de janeiro a dezembro. Neste exemplo, considere que exista uma taxa de erro conhecida de 3%, exemplo: erros de digitação, em ambas as bases de dados. Logo, a probabilidade dos dois registros, $(a \in A \; e \; b \in B)$ serem iguais, isto é, terem o mesmo mês de aniversário e se referirem a um único objeto $((a, b) \in M)$ é de 97%. Desta forma $m_i = 0,97$, e a probabilidade de dois atributos serem diferentes e referirem-se a um único objeto é de 3% ou $(1 - m_i) = 0,03$.

Ainda, para o cálculo de U, dois registros, $(a \in A \; e \; b \in B)$ que são iguais mas referem-se a objetos distintos $((a, b) \in U)$ a probabilidade do mês de aniversário ser o

mesmo é $(\frac{1}{12}) = 0,083$, ou 8.3% de probabilidade que dois indivíduos diferentes tenham o mesmo mês de aniversário, assim $u_i = 8,3\%$, e a probabilidade de dois atributos diferentes referirem-se a dois objetos distintos é $(\frac{11}{12}) = 0,917$ ou $(1 - u_i) = 91,7\%$.

A partir do uso destas probabilidades, podem ser gerados dois pesos parciais[5]:

- Peso da similaridade $= log_2 \left(\frac{m_i}{u_i}\right)$ ou
- Peso da não similaridade $= log_2 \left(\frac{(1-m_i)}{(1-u_i)}\right)$

O cálculo do peso final P para a classificação dos registros é feito da seguinte forma: cada par de atributos i dos registros receberá somente um peso p_i, que será o peso da similaridade ou o peso da não similaridade. A escolha do peso será em função da comparação dos atributos, se os atributos dos registros forem iguais, será atribuído o peso da similaridade, caso sejam diferentes, o peso da não similaridade. Sequencialmente é feito a soma dos pesos parciais de todos atributos, gerando o peso final P, de outro modo: $P = \sum_{i=1}^{n} p_i$.

Por fim, o peso final P é comparado com dois valores de limiar, definidos previamente, α e β. Se $P \geq \beta$ o par de registros é inserido em A1, similares, se $\alpha < P < \beta$, eles serão inclusos no grupo A2, possivelmente similares e se $P \leq \alpha$ o par ficará no grupo A3, diferentes.

Em sua pesquisa, Peter Christen [10], apresenta o processo de deduplicação que utiliza conceitos do método de Fellegi e Sunter. Trata-se de uma proposta atual, madura, o modelo é utilizado para atender a demanda de deduplicação de cadastros da área da saúde. Porém, pode ser facilmente adaptável para a detecção de duplicidades em qualquer base de dados similar. A Figura 2.2 apresenta uma abstração dos estágios do processo.

A etapa inicial do modelo de Christen é o pré-processamento, apresentado na seção 2.2.1, nesta é feita a padronização dos dados a serem deduplicados. A seguir a etapa de blocagem descrita na seção 2.2.3, quando é feita a separação dos registros em blocos, para viabilizar o processamento das comparações entre registros para bases de dados com muitos registros. Na seção 2.2.4, a etapa de comparação apresenta técnicas de comparação e atribuição de pesos de similaridade para os registros comparados. Os pesos gerados

[5]O log de base 2 é utilizado para auxiliar na soma dos pesos parciais, gerando o peso final P

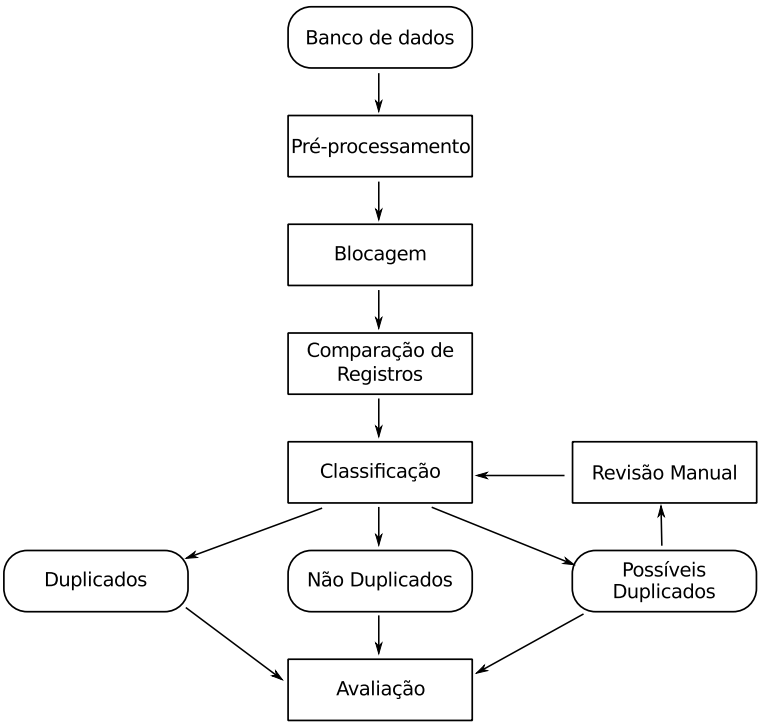

Figura 2.2: Visão geral - Processo de Deduplicação [10]

na etapa de comparação são utilizados como fonte para a etapa de classificação, apresentada na seção 2.2.5, quando os registros são separados em função do peso final, gerando conjuntos de registros duplicados, não duplicados e possíveis duplicados. A seção 2.2.6, aborda métricas que podem ser utilizadas para medir a qualidade da deduplicação.

2.2.1 Pré-processamento

A primeira etapa do processo de deduplicação de registros é a padronização. Registros iguais, mas com diferenças na forma do registrado tem pouca chance de serem identificados como similares. Por exemplo, o Quadro 2.1 mostra três registros iguais, porém se comparados computacionalmente, sem nenhum tipo de pré-processamento ou mesmo aproximação, serão classificados como diferentes.

Quadro 2.1: Registros não padronizados

nome	endereco	bairro	cidade	nascimento
Osvaldo Marcio Cavalieri	Rua João Machado 1561 Centro	-	Curitiba	04.10.1974
osvaldo marcio cavalieri	João Machado 1561	centro	Curitiba	04/10/74
OSVALDO MARCIO CAVALIERI	R.Joaao Machado 1561	Centro	Curitiba	04/10/1974

A etapa de pré-processamento consiste no trabalho de converter dados para um formato padronizado. Seu resultado irá refletir diretamente na qualidade do processo de identificação de duplicidades. Basicamente a etapa de pré-processamento consiste em quatro atividades: substituição de caracteres e/ou exclusão de palavras desnecessárias (seção 2.2.1.1); "tokenização" e padronização dos atributos de cada registro (seção 2.2.1.2); segmentação dos atributos (seção 2.2.1.3) que é a divisão de campos texto em atributos identificados; e a verificação do resultado da padronização, seção (2.2.1.4).

2.2.1.1 Substituição de caracteres e palavras desnecessárias.

Nesta atividade é feita a busca e remoção ou substituição de caracteres ou palavras que são desnecessárias ao processo. Por exemplo, caracteres acentuados podem ser substituídos por seu igual sem acento, palavras escritas numericamente poderão ser substituídas por sua forma em extenso, caracteres como interrogação, hífen ou mesmo palavras fora do contexto da comparação podem ser removidos. Exemplo "s/n".

Além de converter todos os caracteres para minúsculas e retirar todos espaços em branco excedentes, para esta atividade o FEBRL oferece as listas de correção. Uma lista de correção é um arquivo texto, cada linha do arquivo contém um valor substituto, seguido de uma lista de valores substituíveis, o procedimento ocorre da seguinte forma: caso um dos valores na lista de substituíveis seja encontrado no registro avaliado para padronização, o valor encontrado, caractere ou palavra, no registro, será substituído pelo valor substituto, o Quadro 2.2 mostra uma parte de uma lista de correção.

2.2.1.2 Tokenização e Padronização

Em compiladores, especificamente na análise léxica, dado uma cadeia de caracteres, "tokenizar" implica em atomizar, ou seja, quebrar a cadeia em sequências de caracteres que possuam um significado em conjunto [1], por exemplo palavras. Cada sequência de caracteres resultante é chamada de *token*.

Para o processo de "tokenização", por exemplo, dado um nome, este é dividido em *tokens* e cada *token* é procurado em uma ou mais tabelas de pesquisa, chamadas de *look-*

Quadro 2.2: Lista de correção

```
# Remover caracteres e palavras da entrada
        ' ' := '.', '̃','', '_', ' ' ' ', '̂', '̄', ' na ', '*', ' n/a ', ' n.a. ', ' ! '
# Substituir acentos, cedilha
        'a' := 'ã', 'â', 'á', 'à', 'ä', 'Ã', 'Â', 'Á', 'À', 'Ä'
        'o' := 'ó', 'õ', 'ô','ö', 'Ó', 'Õ', 'Ô', 'Ö'
        'u' := 'ú', 'ü', 'Ú', 'Ü'
        'c' := 'ç', 'Ç'
        'e' := 'é', 'ê','ë', 'É', 'Ê', 'Ë'
        'i' := 'í','ï', 'Í', 'Ï'
# Substituir números
        ' um ' := ' 1 ', ' 01 '
        ' dois ' := ' 2 ', ' 02 '
        ' tres ' := ' 3 ', ' 03 '
```

up tables. As tabelas *look-up* contém um conjunto de palavras corretas e cada uma destas palavras corretas pode ser seguida por variações dela mesma. São exemplos de variações: erros tipográficos comuns, abreviações, apelidos, etc.

No processo de padronização, cada *token* é confrontado com as variações da palavra e caso seja igual a uma das variações, será substituído pela palavra correspondente correta. Exemplificando: utilizando o campo endereço da terceira linha do Quadro 2.1, tokenizado para "R.", "joaao machado","1561", considerando o *token*, "joaao machado"[6], este será procurado nas tabelas de *look-up* e será encontrado no arquivo de *look-up* endereço (Quadro 2.3), sendo substituído pelo valor correto "joao machado". Caso, não seja encontrado, será procurado *token* por *token* separadamente, ou seja, "joaao", depois "machado", e assim, o processo é repetido até serem procurados todos os *tokens*. Caso algum *token* não seja encontrado, a palavra será mantida na sua forma original.

As tabelas *look-up*, são arquivos texto, podem conter palavras para diversos segmentos, exemplo: nomes masculinos e femininos; endereços; cidades; tipos de logradouro; etc. O Quadro 2.3 apresenta trechos de quatro tabelas *look-up*, a primeira e segunda para nomes masculinos e femininos, a terceira para sobrenomes e a quarta, para logradouros.

Durante o processo de tokenização, para cada *token* é atribuído uma ou mais *tags*. Que são utilizadas para auxiliar e agilizar o processo de segmentação (seção 2.2.1.3), pois,

[6]inicialmente procura pela cadeia completa de caracteres.

Quadro 2.3: Tabelas *look-up*

tag=<GM> # Tag para nomes masculinos
angelo : angeloo,angwlo,angrlo,anelo,angeelo,aangelo,angelp,angwlo,angel,angeelo,angelp,angleo
vicente : vicwnte,vicentr,viente,viceente,vicentw,vicene,vicentee,vicnete,viicente,vicrnte
tag=<GF> # Tag para nomes femininos
adriana : adriaana,adiana,adrianaa,adriaa,adrisna,adraina,adriiana,adrians
glaucia : glaaucia,glucia,glauciaa,glauucia,glauca,glsucia,glacuia,glauciia,glaucis,glaycia
tag=<SN> # Tag for sobrenomes
demetri : deetri,demwtri,demeti,demeetri,demteri,demetrii,demrtri
deodoro : deodoroo,drodedep, dwodeodoo,deeodeodorp,deoodro,deoodoro
tag=<WN> # Tag para nomes de logradouros
isidoro mikosz : isidoro mikoosz,isiidoro mikosz,isidoro mikpsz,isoidoro mikosz,isiodro mikosz,isdoro mikosz
joao machado : joaao machado,joao machsdo,joso machado,joa omachado,joap machado,joaomachado

no processamento da segmentação, ao invés de avaliar a palavra em si, avalia-se a *tag*.

2.2.1.3 Segmentação

A terceira atividade do pré-processamento é a segmentação e gravação dos *tokens*, ou atributos, separadamente. A segmentação, é necessária para otimizar e obter melhores resultados na etapa de comparação, quando, ao invés de comparar campos inteiros de texto, serão comparados partes menores, ou seja os *tokens*. Os Quadros 2.4 e 2.5, apresentam dois exemplos, de segmentação para nome e endereço.

Quadro 2.4: Exemplo tokenização nome

José	Antonio	Paula
jose	antonio	paula
GM	GM	GF,SN

Quadro 2.5: Exemplo tokenização endereço

Rua	Joãao Machado	1561	Centro
rua	joao machado	1561	centro
WT	WN	N4	LQ

Na segmentação, para cada *token* é atribuída uma ou mais *tags*. O objetivo da atribuição é acelerar o processo de segmentação da cadeia de caracteres, já que ao invés de analisar todas as variações de *tokens* existentes, isto é, todas as palavras, e então rotulá-las, será avaliado a *tag*, que é genérica para uma classe de palavras, e segmentado

em função da *tag*. Um token pode estar em uma ou mais tabelas de *look-up* ou pertencer a uma regra codificada (Quadro 2.7), gerando desta forma, uma ou mais *tags* por *token*. Por exemplo, o nome Paula pode ser um nome feminino, *tag* GF, ou sobrenome, *tag* SN. O Quadro 2.6[7] apresenta as *tags* utilizadas pelo FEBRL.

Quadro 2.6: Lista de tags utilizadas no módulo de pré-processamento do FEBRL

Tag	Descrição	Componente	Baseado em
LQ	Localidade - bairro	endereço	tabela look-up
LN	Localidade - cidade	endereço	tabela look-up
TR	Território - unidade federação	endereço	tabela look-up
CR	Território - país	endereço	tabela look-up
IT	Tipo de instituição	endereço	tabela look-up
IN	Nome de instituição	endereço	tabela look-up
PA	Tipo de código postal	endereço	tabela look-up
PC	Código postal	endereço	tabela look-up
N4	Números com 4 dígitos	endereço	regra codificada
UT	Tipo de unidade - complemento, apartamento, bloco	endereço	tabela look-up
WN	Nome do logradouro	endereço	tabela look-up
WT	Tipo do logradouro	endereço	tabela look-up
TI	Titulação	nome	tabela look-up
SN	Sobrenome	nome	tabela look-up
GF	Nome feminino	nome	tabela look-up
GM	Nome masculino	nome	tabela look-up
PR	Prefixo do nome - “de, da”	nome	tabela look-up
SP	Separador para nomes - “e”	nome	tabela look-up
BO	Filho de, filho, neto	nome	tabela look-up
NE	nome de solteira	nome	tabela look-up
II	Abreviatura - somente inicial	nome	regra codificada
ST	Nome de santos	endereço/nome	tabela look-up
CO	Vírgula, ponto e vírgula, dois pontos	endereço/nome	regra codificada
SL	Barra e contra barra	endereço/nome	regra codificada
NU	Outros números	endereço/nome	regra codificada
AN	Tokens alfanuméricos	endereço/nome	regra codificada
VB	Colchetes, chaves, aspas	endereço/nome	regra codificada
HY	hífen	endereço/nome	regra codificada
RU	Lixo - para *tokens* a serem descartados	endereço/nome	tabela look-up
UN	Desconhecido - nenhum dos itens acima	endereço/nome	regra codificada

Fonte: Adaptado de Data Matching [10]

A segmentação tem alguns desafios, nem sempre todos os campos possuem valor para ser atribuído uma *tag*, ou mesmo, quando um nome se enquadra em duas categorias. Exemplo: nome feminino e sobrenome. Tal fato dificulta sua classificação como nome ou

[7]Colunas Descrição, Componente e Baseado em, foram traduzidas do original.

sobrenome e por consequência, cria um impasse para o processo de segmentação.

Visando resolver estes e outros impasses, existem diferentes técnicas de segmentação para os diferentes tipos de dados: técnicas para nomes, técnicas para endereços, técnicas para nomes comerciais, etc. Duas são as principais [10]: a técnica baseada em regras e a abordagem estatística, que serão abordadas na seção 2.2.2.

2.2.1.4 Verificação

Opcionalmente, pode ocorrer uma quarta atividade. A verificação dos resultados. Esta pode ser feita confrontando-se um ou mais atributos de saída obtidos a partir da segmentação com um conjunto de dados confiável. Por exemplo, comparar cidade ou nome do logradouro com estes mesmos dados obtidos de uma base de dados dos correios. No caso de erro, este pode ser corrigido manualmente, ou, ser corrigido automaticamente, mas a correção automática pode gerar novos erros, que devem ser verificados [10].

2.2.2 Técnicas de segmentação

As principais técnicas de segmentação, são a técnica baseada em regras(seção 2.2.2.1), e a abordagem probabilística (seção 2.2.2.2) [10].

2.2.2.1 Segmentação baseada em regras

A segmentação baseada em regras é a submissão da sequência de *tokens* a uma análise de regras para separar os atributos. Basicamente a técnica consiste numa verificação condicional, codificada da seguinte forma: "se condição então ação" [34]. O Quadro 2.7 apresenta um exemplo de código para segmentação de nomes, $t[n]$ representa a tag, $o[n]$ representa o token associado a tag e L representa o número de *tokens* do nome.

Quadro 2.7: Segmentação baseada em regras

```
se t[n] = 'GM' e t[n + 1]= 'SN' entao nome:= o[n] e sobrenome := o[n + 1]
se t[n] = 'GF' e t[n + 1]= 'SN' entao nome:= o[n] e sobrenome := o[n + 1]
se t[n] = 'SN' e t[n + 1]= 'GM' entao nome:= o[n + 1] e sobrenome := o[n]
se t[n] = 'SN' e t[n + 1]= 'GF' entao nome:= o[n + 1] e sobrenome := o[n]
```

Trecho de código para segmentação baseada em regras para extrair nome e sobrenome de uma sequência de *tokens*.

Para desenvolver um sistema baseado em regras, inicialmente são desenvolvidos regras básicas, em seguida, o conjunto de dados que serão segmentados são submetidos a avaliação daquelas. A medida que forem detectados padrões não previstos nas regras básicas, estes serão adicionados ao conjunto de regras.

Dado a complexidade das regras para longas cadeias de *tokens*, gerar manualmente um conjunto compreensivo de regras é bastante trabalhoso e consome tempo [34]. Ainda, um problema significativo para a abordagem baseado em regras é a dificuldade de ter e manter um conjunto de regras que possa ser aplicado a qualquer conjunto de dados, pois, caso o conjunto de dados possua algum padrão não mapeado, este deverá ser avaliado e incluído nas regras. Caso contrário, a segmentação do novo padrão não será realizada.

Visando transpor a rigidez do modelo baseado em regras é proposto o modelo de segmentação baseado em estatística, abordado a seguir.

2.2.2.2 Segmentação Estatística

A abordagem estatística faz a segmentação dos campos baseando-se em distribuições de probabilidade. Nesta abordagem, o processo de segmentação de *tokens* em campos de saída pode ser visto como um processo de classificação, que, dado um modelo estatístico, cada token é avaliado e associado ao mais provável atributo de saída [36]. Neste processo a classificação atual depende de uma classificação anterior, isto é, a classificação anterior já foi aprendida e incluída no modelo.

O FEBRL, utiliza a segmentação baseada em regras para campos com até dois *tokens*. Para campos que possuam três ou mais *tokens*, devido a ampla utilização e aos bons resultados da segmentação de texto para registros estruturados, [12, 4, 10], a ferramenta utiliza a segmentação estatística usando o modelo escondido de Markov - HMM.

2.2.2.3 Modelo escondido de Markov - HMM

Teoria matemática, desenvolvida na década de 60, o HMM [35], é amplamente utilizado para reconhecimento de voz e processamento de linguagem natural e também pode

ser utilizado em outras áreas, por exemplo, na padronização e segmentação de registros.

O HMM pode ser visto como uma máquina probabilística de estados finitos, composta de um conjunto de estados escondidos (nodos) com ligações de transição (arestas) entre os estados e um conjunto de símbolos de saída ou observações. Cada ligação entre os estados tem uma probabilidade maior que zero, e cada estado tem uma distribuição probabilística de símbolos de saída. As probabilidades de transição e dos símbolos de saída podem ser representadas em matrizes, conforme apresenta as Tabelas 2.1 e 2.2 respectivamente, ou em forma de grafo como mostra a Figura 2.3 que é a representação gráfica da Tabela 2.1.

De maneira ampla, uma ideia geral do modelo escondido de Markov é a visão de um processo duplamente estocástico, onde além do processo principal, ocorre um processo estocástico subjacente, não observável[35][8], ou seja, a observação é indireta. Exemplo, dado um modelo climático com três estados: ensolarado, nublado e chuvoso[9]. E, considerando um estudante, que em função da previsão do tempo, vai à escola de bicicleta em dias ensolarados ou de ônibus em dias nublados/chuvosos. Neste cenário, a observação indireta do estado do aluno, que utilizou a bicicleta, indica uma probabilidade maior de ser um dia ensolarado, porém, ainda existe a possibilidade de ser um dia nublado ou chuvoso.

Tabela 2.1: Matriz de transição de estados

estado "de"	estado "para"						
	Número	Logradouro	Tipo Logradouro	Cidade	Unidade Federação	CEP	Fim
Início	0,85	0,08	0,02	0,05	0,0	0,0	0,0
Número	0,03	0,93	0,0	0,02	0,0	0,02	0,0
Logradouro	0,02	0,03	0,88	0,07	0,0	0,0	0,0
Tipo Logradouro	0,0	0,03	0,0	0,9	0,04	0,03	0,0
Cidade	0,0	0,0	0,0	0,03	0,35	0,45	0,17
Unidade Federação	0,0	0,0	0,0	0,0	0,0	0,85	0,15
CEP	0,0	0,0	0,0	0,02	0,08	0,0	0,9

A tabela apresenta a matriz de probabilidades. O cruzamento linha, coluna indica a probabilidade de transição de um estado para o outro. Fonte: adaptado de [10]

As matrizes de estados e símbolos de saída são criadas a partir de um conjunto de treinamento extraído do conjunto de dados a ser padronizado.

[8]Mais detalhes sobre o modelo escondido de Markov é encontrado na bibliografia.

[9]Para fins de simplificação, não foi citado as probabilidades de transição de estados.

Tabela 2.2: Matriz de símbolos de saída

Símbolo de Saída	estado					
	Número	Logradouro	Tipo Logradouro	Cidade	Unidade Federação	CEP
NU	0,9	0,01	0,01	0,01	0,01	0,05
WN	0,01	0,5	0,01	0,1	0,01	0,01
WT	0,01	0,01	0,92	0,01	0,01	0,01
LN	0,01	0,1	0,01	0,8	0,01	0,01
TR	0,01	0,06	0,01	0,01	0,93	0,01
PC	0,03	0,01	0,01	0,01	0,01	0,8
N4	0,02	0,01	0,01	0,01	0,01	0,1
UN	0,01	0,31	0,02	0,05	0,01	0,01

A matriz de símbolos de saída correlaciona os símbolos observados, linhas, com os estados,colunas. apresentando a frequência que o símbolo aparece em um estado. Fonte: adaptado de [10]

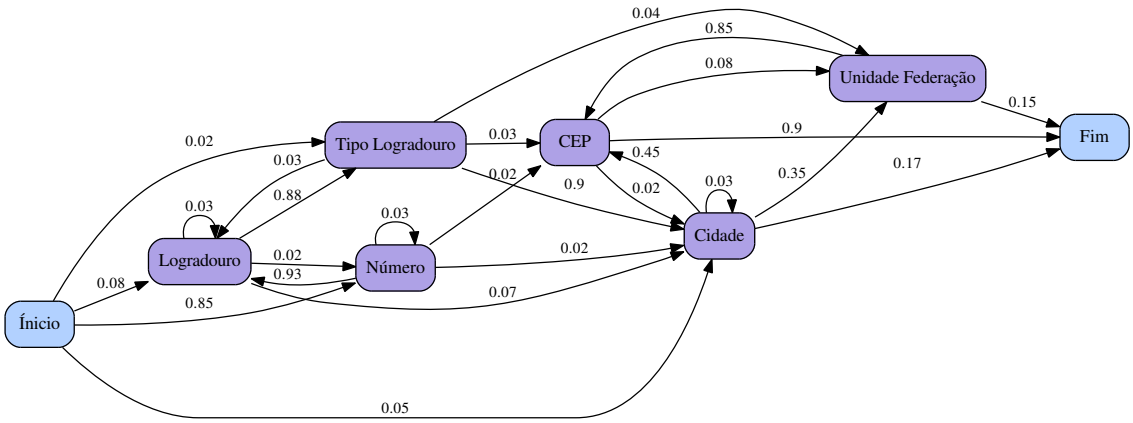

Figura 2.3: HMM de endereço

Representação gráfica das transições entre estados, elaborada a partir da Tabela 2.1. Fonte: adaptado de [10]

Quadro 2.8: Conjunto treinamento

```
# Entrada: Rua Joãao Machado 1326, Centro
# Lista de tokens: ['rua', 'joao_machado', '1326', 'centro']
WT:street_type , WN:street_name1 , N4:number_last , LQ:locality_name1

# Entrada: JMachado rua 328, Centro
# Lista de tokens: ['jmachado','rua', '328', 'centro']
 UN:street_name1 , WT:street_type , NU:number_last , LQ:locality_name1

# Entrada: Rua 326, Joãao Machado
# Lista de tokens: ['rua', '326', 'joao_machado', 'centro']
 WT:street_type , NU:number_last , WN:street_name1, LQ:locality_name1

# Entrada: 222, Av. Larsnjeiras, centro
# Lista de tokens: ['222', 'avenida', 'larjeiras', 'centro']
 NU:number_last , WT:street_type , UN:street_name1 , LQ:locality_name1
```

Como exemplo, considere o conjunto treinamento apresentado no Quadro 2.8. Neste caso, o método para criação das matrizes HMM é o seguinte :

- O cálculo da transição é feito a partir do total de transições observadas de um estado i para um estado j, dividido pelo total de transições i para outros estados [12]. Por exemplo, no Quadro 2.8 do estado início para o estado *street_ type*, a probabilidade é de 50%. Já, de *street_ name1* para *number_ last*, a probabilidade é de 25%.

- Para o cálculo da matriz de símbolos de saída, dado um estado j e um símbolo k a probabilidade é encontrada pelo total de símbolos k emitidos por j, dividido pelo total de símbolos emitidos por j[12]. Assim no Quadro 2.8, a probabilidade do símbolo de saída NU para o estado *number_ last* é de 75%.

Para exemplificar, considere o uso das matrizes das Tabelas 2.1 e 2.2 para realizar a segmentação do endereço "Av. Larsnjeiras 222, curitiba, 83650-470" (propositalmente errado) e considerando já concluídas a etapa substituição de caracteres e palavras desnecessárias (subseção 2.2.1.1), e a etapa da tokenização e padronização (subseção 2.2.1.2), gerando o seguinte resultado:

['avenida', 'laranjeiras', '222', 'curitiba', '83650470']

['WT', 'WN', 'NU', 'LN', 'PC']

Dado estes cinco símbolos, *tags*, e considerando os oito estados da Tabela 2.1, existem $8^5 = 32.768$ possibilidades de combinações de estados escondidos.

Dentre as possibilidades, para encontrar qual a sequência de estados escondidos é a mais provável, é utilizado um algoritmo de programação dinâmica, o algoritmo de Viterbi, que a partir das observações dos estados, indica a sequência mais provável [35, 24].

Para calcular a probabilidade de cada sequência de estados, utiliza-se ambas as matrizes. Combinando a matriz de transição de estados e a matriz de símbolos de saída para a sequência do exemplo acima, uma possibilidade de estado escondido é:

Início ⇒ Tipo logradouro (WT) ⇒ Logradouro(WN) ⇒ Número(NU) ⇒ CEP (LN) ⇒ Cidade(PC) ⇒ Fim

Procurando o par "Início ⇒ Tipo logradouro" na matriz de transição (Tabela 2.1),

resulta em 0,2, em seguida procurando o par "Tipo logradouro ⇒ WT" na matriz de símbolos (Tabela 2.2), resulta 0,92, e assim, sucessivamente, encontra-se a probabilidade desta sequência:

$$0,2 \times 0,92 \times 0,03 \times 0,5 \times 0,02 \times 0,9 \times 0,02 \times 0,01 \times 0,02 \times 0,01 \times 0,17 = 3,38 \times 10^{-14}$$

Uma outra possibilidade pode ser: Início ⇒ Tipo logradouro (WT) ⇒ Logradouro(WN) ⇒ Número(NU) ⇒ Cidade (LN) ⇒ CEP(PC) ⇒ Fim

$$0,02 \times 0,92 \times 0,03 \times 0,5 \times 0,02 \times 0,9 \times 0,02 \times 0,8 \times 0,45 \times 0,8 \times 0,9 = 2,58 \times 10^{-8}$$

Comparando-se as probabilidades das duas sequências, nota-se que a segunda $2,58 \times 10^{-8}$ tem maior chance de ser a correta. Assim, quando da segmentação do campo endereço, os atributos serão alocados em "sub campos" conforme a sequência que apresente maior probabilidade de ser a correta.

2.2.3 Blocagem ou indexação

Após o pré-processamento, a próxima etapa é a blocagem/indexação. Os registros devem ser separados em blocos de similares. O procedimento é necessário para diminuir e inclusive tornar viável a próxima etapa: comparação.

Dado dois conjuntos de registros m e n, sem blocagem, o número de comparações para encontrar os registros similares é dado por $m * n$ comparações, suponha 20.000 registros em cada conjunto, serão necessários 400.000.000 de comparações, mesmo para a deduplicação de um único conjunto n é necessário $n * (n-1)/2$ [10] comparações, ou seja, dado os 20.000 registros r1,r2,r3,...r20.000 serão necessários 199.990.000 comparações, exemplo: (r1,r2),(r1,r3),(r1,...),(r1,r20.000),(r2,r3),(r2,...),(r2,r20.000),(r3,...) e (r3,r20.000)... para identificar os registros similares.

O procedimento de blocagem pode ser visto, como uma etapa de procura e distribuição, pois consiste em procurar registros similares e distribuí-los em um ou mais blocos. Uma abordagem geral para a blocagem consiste na escolha de uma chave de bloco (seção 2.2.3.1), seguido da codificação da chave (seção 2.2.3.2), encerrando com as técnicas de blocagem (seção 2.2.3.3), conforme será descrito a seguir.

2.2.3.1 Chave de bloco

No processo de blocagem, a escolha da chave de bloco (atributo identificador) é uma das atividades mais importantes [11], pois a chave, aliada a técnica de blocagem empregada, será responsável pelo sucesso do processo, que é a distribuição dos registros similares em seus blocos.

A escolha da chave de bloco é feita utilizando os atributos disponíveis no conjunto para deduplicação, mas, antes de eleger este ou aquele atributo, é necessário considerar:

- Qualidade dos dados do atributo - a qualidade influência diretamente na segmentação dos blocos. Se um atributo possui muitos valores vazios, ou não informado, a chave criará um bloco com muitos registros ditos similares, mas que na prática não o são. Outro ponto é a qualidade do dado em si, uma chave com muitos erros pode fazer que registros similares sejam distribuídos em blocos distintos.

- Frequência dos valores do atributo - Dado um conjunto de valores para um atributo, a distribuição da frequência destes valores influência diretamente no número de comparações, por exemplo, para um conjunto de dados com 100.000 registros, utilizar um atributo primeiro nome como chave, mas se considerado que 10% dos valores desta chave sejam Maria, será criado um bloco com 10.000 registros, que gerará 49.995.000 de comparações, para um único bloco, logo, apesar de haver ganho, melhor seria escolher uma chave com uma distribuição dos valores, o mais equilibrada possível.

- Relação entre o número e tamanho dos blocos - Chaves muito genéricas podem gerar um baixo número de blocos, aumentando a probabilidade de localização de registros similares no bloco, porém aumentando o tempo necessário para o processamento, por exemplo, utilizar o campo sexo como chave de bloco, gerará apenas dois blocos, por outro lado, escolher uma chave muito especifica gerará uma grande quantidade de blocos, com poucos registros e com tempo de comparação reduzido, porém registros similares poderão não ser comparados por estarem em blocos distintos. O equilíbrio pode ser encontrado a partir da composição de uma chave com mais de um atributo,

e, com a escolha da técnica de blocagem, pois, algumas técnicas podem distribuir um único registro similar, em mais de um bloco.

2.2.3.2 Funções de codificação da chave

O desafio para identificação de iguais são as diferenças que os valores dos atributos podem apresentar, idealmente, os valores da chave também podem apresentar valores diferentes, que por consequência distribuam registros iguais para blocos distintos. Para minimizar essa situação, a chave pode ser submetida a uma função de codificação, que transformará valores similares em um código igual, (exemplo: uma função fonética tal como *Soundex,Phonix, NYSIIS*), irá converter a chave para um código que refletirá seu equivalente sonoro, exemplos, Quadro 2.9, isto é, valores diferentes na escrita, porém com sonoridade igual, terão um mesmo código e, consequentemente, serão inseridos no mesmo bloco.

Quadro 2.9: Funções fonéticas

Nome	Soundex	Phonex	Phonix	NYSIIS	Double metaphone	Fuzzy Soundex
peter	p360	b360	p300	pata	ptr	p360
pete	p300	b300	p300	pat	pt	p300
pedro	p360	b360	p360	padr	ptr	p360
stephen	s315	s315	s375	staf	stfn	s315
steve	s310	s310	s370	staf	stf	s310
smith	s530	s530	s530	snat	sm0,xmt	s530
smythe	s530	s530	s530	snat	sm0,xmt	s530
gail	g400	g400	g400	gal	kl	g400
gayle	g400	g400	g400	gal	kl	g400
christine	c623	c623	k683	chra	krst	k693
christina	c623	c623	k683	chra	krst	k693
kristina	k623	c623	k683	cras	krst	k693

Fonte: [10]

Via de regra, as funções fonéticas tem o mesmo principio, isto é, analisar as chaves e gerar um código, caso as chaves sejam similares, o código gerado deverá ser igual, por exemplo, no Quadro 2.9 o nome Gail e Gayle, após a execução de uma função, geraram um mesmo código.

Um dos mais antigos e mais utilizados [27] algoritmos fonéticos, baseado na pronúncia do idioma norte americano, o *Soundex*, funciona da seguinte forma: a partir da palavra

analisada é gerado um código com um caractere inicial seguido de números. O primeiro caractere do código é o caractere inicial da palavra, os números são gerados a partir dos demais caracteres da palavra, os caracteres são substituídos por números encontrados numa tabela de conversão, todos os 0, que correspondem a "a-e-i-o-h-w-y", são excluídos, em seguida os números repetidos são reduzidos a uma única ocorrência do algarismo, por fim, caso o número de algarismos seja inferior a três, serão adicionados zeros até atingir três. Exemplo, gerado o código "p0330111", intermediariamente, "p31" e por fim "p310".

Na etapa de blocagem, o FEBRL oferece todas as funções fonéticas do Quadro 2.9, inclusive a função *substring*, que é a escolha de um pedaço da chave, e pode ser utilizada para campos numéricos, por exemplo, para escolher os 4 dígitos do ano de uma data. O Apêndice A traz uma breve descrição de cada uma das funções fonéticas do Quadro 2.9.

2.2.3.3 Técnicas de blocagem

A técnica de blocagem é responsável pelo gerenciamento da maneira como os blocos são criados. Em outras palavras, como os registros, em função da chave de bloco, são distribuídos nos blocos. Um exemplo de técnica é a indexação q-gram [29]:

Dado um conjunto de dados com muitos erros, ou mesmo, muitas variações de nomes, algumas técnicas de blocagem podem não detectar registros similares, consequentemente inserindo-os em blocos distintos. A indexação q-gram visa aumentar a probabilidade de que registros similares estejam no mesmo bloco. A partir da chave de bloco, o algoritmo gera variações de *substrings* de tamanho q e a partir destas, insere um mesmo registro em diferentes blocos, exemplo, os sobrenomes Muller e Miller, com $q = 2$, geram a mesma chave "llleer", Quadro 2.10 e inserem os registros r1 e r2 no mesmo bloco.

Uma outra técnica de blocagem, parecida com a indexação q-gram é a *suffix array index*. O método consiste em gerar variações do valor da chave [10], por exemplo, Marcio, os sufixos, serão arcio, rcio, cio,io e o.

Para evitar que esta técnica gere uma quantidade de blocos e ou de registros por bloco, que inviabilize o processamento, existem dois parâmetros, o primeiro é o tamanho mínimo do sufixo. Exemplo, um sufixo tamanho 4 para Marcio, permitiria somente os

Quadro 2.10: Indexação Q-gram

Reg ID	Chave - Sobrenome	Bigram variações	Valores de chave
r1	miller	[mi,il,ll,le,er], [il,ll,le,er],[mi,ll,le,er],[mi,il,le,er], [mi,il,ll,er],[mi,il,ll,le], [ll,le,er],[il,le,er],[il,ll,er], [il,ll,le],[mi,le,er],[mi,ll,er], [mi,ll,le],[mi,il,er],[mi,il,le],	'miilllleer', 'illlleer', 'millleer', 'miilleer', 'miilller', 'miilllle', **'llleer'**, 'illeer', 'illler' 'miller', 'illlle', 'mileer' 'millle', 'miiler', 'miille',
r2	muller	[mu,ul,ll,le,er], [ul,ll,le,er],[mu,ll,le,er],[mu,ul,le,er], [mu,ul,ll,er],[mu,ul,ll,le], [ll,le,er],[ul,le,er],[ul,ll,er], [ul,ll,le],[mu,le,er],[mu,ll,er], [mu,ll,le],[mu,ul,er],[mu,ul,le],	'muulllleer', 'ulllleer', 'mullleer', 'muulleer', 'muulller', 'muulllle', **'llleer'**, 'ulleer', 'ulller' 'muller', 'ulllle', 'muleer' 'mullle', 'muuler', 'muulle',

Fonte: [10]

sufixos arcio e rcio. O segundo parâmetro, tamanho máximo de registros por bloco, visa evitar que sufixos muito comuns gerem blocos com muitos registros. Exemplo, um sufixo tamanho 2, para o nome Neide, geraria os sufixos eide, ide e de, neste caso, muitos nomes com o sufixo "de" poderia gerar um bloco com muitos registros.

Além destas, O FEBRL oferece outras técnicas, listadas no Apêndice B.

2.2.4 Comparação

Terminada a distribuição dos registros em blocos, a etapa seguinte, bloco a bloco, é comparar os registros entre si para identificar quão similar são.

Para cada par de registros do bloco, cada par de atributos são submetidos a uma função de similaridade. Em linhas gerais, assumindo $s = sim(a_i, a_j)$ como uma função de similaridade que resulta um número s e indica o grau de similaridade entre os atributos[10] a_i e a_j, e considerando s um resultado ajustado entre $0 \leq s \leq 1$, são resultados possíveis para a função:

- $sim(a_i, a_j) = 1$: atributos completamente iguais.
- $sim(a_i, a_j) = 0$: atributos completamente diferentes.
- $0 < sim(a_i, a_j) < 1$: grau de similaridade entre os atributos.

[10] a_i e a_j podem representar cadeias de caracteres, números, tempo, datas, anos, etc.

Para calcular a similaridade entre atributos, o FEBRL, possui 26 opções. Não é propósito descrever detalhadamente o funcionamento dos diversos algoritmos. Na sequência, segue descrição dos algoritmos utilizados neste trabalho, e no Apêndice C uma breve descrição dos algoritmos estudados para comparação, disponíveis no FEBRL [7].

- Comparação exata de *strings* - Compara duas *strings*, se são iguais ou não.
- Comparação de datas. A entrada da função são dois valores em dias, utilizados como tolerância para diferenças entre as duas datas: caso a primeira data seja menor que a segunda data. O primeiro valor é a tolerância de dias que a primeira data pode ter antes da segunda data. Caso a primeira data seja maior que a segunda data, o segundo valor é o número máximo de dias que a primeira data pode ter após a segunda data. Além deste caso, são calculados pesos de similaridade diferenciados quando entre as duas datas existir a possibilidade de troca de posição entre dia e mês. Por exemplo 10/07/2013 e 07/10/2013, e quando o dia e o ano forem iguais, exemplo: 10/01/2014 e 10/07/2014. Caso não se encaixe em nenhuma destas situações será retorna um peso de diferença definido pelo usuário.
- Distância Jaro - A função Jaro [38], utiliza como base, o número de caracteres de cada *string*, a contagem de caracteres iguais, na mesma posição da *string* e a contagem de transposições de caracteres, subsequentes, mas com ordem trocada, exemplo: rc e cr, para calcular o valor da similaridade.
- Distância Jaro-Winkler - Variação da técnica Jaro a função Winkler, para o cálculo da similaridade também considera o número de igualdades nos quatro primeiros caracteres de cada *string*.

2.2.5 Classificação

Após os registros serem separados em blocos, os atributos comparados par a par, recebendo um peso de similaridade, a próxima etapa é a classificação. Nesta etapa, de maneira geral, é calculado o peso final de similaridade, que será comparado a limiares de

corte, para que os registros sejam identificados como iguais, diferentes e possivelmente iguais.

As abordagens de classificação podem ser divididas em supervisionada e não supervisionada. Como visto na seção 2.2 a abordagem não supervisionada identifica pares ou grupos de registros como iguais, baseando-se somente na similaridade entre eles e a abordagem supervisionada precisa de um conjunto de dados de treinamento, que inequivocamente, identifique as duplicidades, ou não duplicidades, como tal. Este conjunto treinamento, será utilizado como referência para identificar as duplicidades na base de dados.

Além do método de classificação de Fellegi e Sunter, descrito em detalhes na seção 2.2, o FEBRL, oferece alguns outros métodos para classificação, descritos no Apêndice C. Basicamente, a abordagem não supervisionada de Fellegi e Sunter, consiste em somar os pesos parciais, atribuídos aos pares de atributos dos registros comparados, gerando um peso final. Em seguida o peso final é posicionado entre dois valores de limiar informados pelo usuário, e com base no posicionamento, os registros são classificados em iguais, diferentes ou possivelmente iguais.

2.2.6 Avaliação

Avaliar, implica julgar e emitir parecer. No FEBRL, se na etapa de classificação for possível aplicar algum método supervisionado, a ferramenta apresenta alguns indicadores de qualidade comumente utilizados em mineração de dados, aprendizado de máquina, recuperação da informação, como *accuracy, precision, recall* e *f-measure*[10].

Já, quando não for possível aplicar um método supervisionado, pode ser feita uma avaliação manual, ou melhor, uma revisão manual, como segue.

2.2.6.1 Revisão manual

Dado os métodos triviais de classificação, que geram três conjuntos de saída, os duplicados, não duplicados e possivelmente duplicados. A revisão manual consiste na avaliação manual, dos resultados, verificando manualmente a classificação realizada.

No entanto, realizar a revisão manual pode ser um procedimento difícil. Fatores como a quantidade de registros e a falta de conhecimento do revisor sobre domínio dos dados, são elementos que podem influenciar negativamente o resultado da revisão manual. Mas, ainda que a revisão manual seja de difícil execução, seu resultado pode ser utilizado para refinar a etapa de classificação, por exemplo, refinar os valores de limiar para a classificação.

2.2.7 Aplicativos de Deduplicação

Dada a demanda de soluções para integração de grandes bases de dados, inteligência nos negócios, gerenciamento de relacionamentos com clientes, entre outras, a deduplicação é bastante pesquisada, tanto no meio acadêmico quanto no comercial, em consequência disto, diversos são os aplicativos desenvolvidos para tratar o assunto. A opção deste trabalho não é a de desenvolver um novo aplicativo de deduplicação, mas sim utilizar um aplicativo já existente e aplicá-lo na deduplicação de dados relacionados a saúde.

Desta forma, para subsidiar a escolha do aplicativo de deduplicação a ser usado, foi feito um apanhado geral das características de alguns aplicativos de deduplicação. Como linha de corte inicial, foram selecionadas algumas restrições aos aplicativos candidatos: não deve ter custo de aquisição, ter código aberto e estar vinculado a um contexto de utilização semelhante a esta pesquisa, ou seja, área da saúde ou deduplicação de grandes bases de dados.

Os aplicativos que atendem a estas restrições são os seguintes: Big Match na seção 2.2.7.1, DuDe na seção 2.2.7.2, Fril na seção 2.2.7.3 e o Febrl na seção 2.2.7.4.

2.2.7.1 BigMatch

O aplicativo BigMatch [39], desenvolvido em C, distribuído livremente, é utilizado pelo censo norte americano para avaliar grandes bases de dados. É um aplicativo utilizado para procurar por possíveis registros duplicados em bases de dados muito grandes, que, devido o tamanho, de outra forma não poderiam ser processadas.

Em essência, o BigMatch confronta duas bases de dados, a primeira ou principal,

que é muito grande, chamada *record* é lida sequencialmente e nunca é ordenada. A segunda contém os critérios de blocagem e é denominada *memory* pois deve ser alocada em memória. Conforme os registros do *record* são lidos, eles são comparados com os critérios definidos na *memory*, gerando um peso para cada comparação. Em função do peso, os dados são organizados em grupos de registros similares e cada grupo é gravado separadamente, para posteriormente serem avaliados individualmente.

2.2.7.2 DuDe

Desenvolvido na Universidade de Potsdam, DuDe - *Duplicate Detection* [21] é um conjunto de ferramentas, desenvolvido em linguagem java, livre (*GNU General Public License*[11]), possui módulos com funcionalidades específicas para detectar duplicidades.

Visando a extração de dados, o módulo *Data Extractors* pode extrair dados de banco de dados relacionais, como: Oracle e PostgreSQL, também pode processar arquivos CSV[12], documentos XML[13] e arquivos de bibliografia Bibtex[14]. O módulo *PreProcessor* é opcional, utilizado para colher dados estatísticos, como exemplo, o número de registro ou valores distintos para um campo, durante o processo de extração dos dados. Os algoritmos de particionamento, *Partitioning Algorithms* possuem uma série de técnicas de blocagem para gerar pares de registros candidatos. Já o módulo de comparação *Comparators* possui funções de cálculo de similaridade. É utilizado para comparar os registros candidatos calculando a similaridade entre eles. *Postprocessor* recebe os registros com os pesos calculados, permite o cálculo de dados estatísticos como: tempo de processamento, número de pares duplicados, etc. Por fim, o módulo *Output* escreve os resultados da deduplicação.

Sem uma interface gráfica, a estrutura modular do DuDe possibilita a inclusão de código desenvolvido por terceiros e alterações também são possíveis. Bem documentado, com exemplos de uso, o conjunto de ferramentas funciona como um *plugin* para outros aplicativos desenvolvidos em linguagem java que necessitem de suas funcionalidades.

[11] http://www.gnu.org/copyleft/gpl.html
[12] *Comma Separated Values*
[13] *eXtensible Markup Language*
[14] http://www.bibtex.org/

2.2.7.3 FRIL

Fine-Grained Records Integration and Linkage - FRIL [28] foi desenvolvido como parte de projeto conjunto entre a Universidade Emory e o Centro de controle de Doenças em Atlanta, Estados Unidos. De distribuição livre, sob licença *Mozilla Public License*, desenvolvido em linguagem java, o aplicativo oferece um extenso conjunto de parâmetros que podem ser configurados pelo usuário, permite a combinação de esquemas de dados, assim como a localização de registros similares, possui métodos de blocagem e uma boa variedade de funções de comparação. Possui interface gráfica e pode executar em sistemas multi-core.

A ferramenta FRIL traz funcionalidades como o pré-processamento, com o uso de expressões regulares oferece capacidade de padronização dos dados, permite separar ou juntar atributos antes desdes serem avaliados. O usuário pode selecionar parâmetros, tais como: campos a serem comparados, funções de comparação, pesos, funções de blocagem e métricas de avaliação. O paralelismo multi-core é transparente ao usuário, ou seja, a ferramenta detecta o ambiente e passa a processar paralelamente. É possível avaliar a lista dos parâmetros utilizados e ter uma prévia dos resultados finais.

2.2.7.4 FEBRL

Fruto da colaboração entre a Universidade Nacional Australiana e o departamento de saúde do estado de New South Wales - Austrália, desenvolvido a partir 2003, distribuído livremente *ANUOS licence*, desenvolvido em python[15], e em constante evolução, o *Freely Extensible Biomedical Record Linkage - FEBRL* [10] é um aplicativo que possui um conjunto de funcionalidades para detecção de duplicidades.

Ainda que o FEBRL possa ser aplicado a outros domínios, o projeto de pesquisa objetiva desenvolver técnicas para aperfeiçoar os processos de detecção de duplicidades em bases de dados da saúde.

Com interface visual, entrada de dados via arquivos texto, de código aberto, modular, além de permitir consultas ou alterações no código, a ferramenta permite a inclusão

[15]http://www.python.org.br

de novos módulos aos existentes.

Como exemplo da quantidade de técnicas disponíveis no FEBRL, a Figura 2.4 apresenta uma fotografia tirada deste aplicativo, especificamente o módulo de comparação que possui vinte e seis funções para comparações.

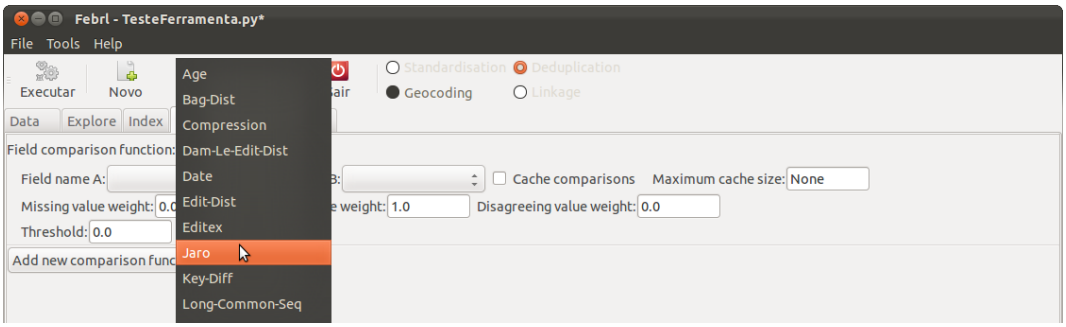

Figura 2.4: Interface FEBRL

Fonte: Ferramenta FEBRL

O aplicativo possui nativamente os seguintes módulos: pré-processamento, pode padronizar dados como nome, endereço, telefone e datas. Módulo blocagem, disponibiliza sete métodos de blocagem e oito técnicas de codificação fonética. Módulo comparação, possui vinte e seis funções de comparação que podem comparar "strings", números, datas, hora e posições geográficas. O módulo classificação possui seis diferentes métodos de classificação, e por fim o módulo de avaliação com dados estatísticos dos processos.

2.3 Qualidade da Informação

O termo qualidade pode ser aplicado em vários contextos. É senso comum relacionar o conceito de qualidade ao que o usuário espera dela, ou seja, ainda que o produto não seja o melhor, se atender a expectativa ou critério do usuário, ele terá qualidade. Devido as diferentes expectativas, pode ser difícil decidir se uma informação tem qualidade ou não, pois diferentes pessoas podem ter diferentes expectativas. Desta forma, para a avaliação da qualidade é importante utilizar critérios comuns [2].

Dentre as definições para a qualidade de dados, destaca-se duas [23]:

- Qualidade inerente: o dado é preciso, exato, corresponde fidedignamente ao objeto real que foi mapeado.

- Qualidade pragmática: o valor que dados precisos tem para o usuário. Não basta que o dado seja exato, tem que ser útil.

Frente a estas definições e considerando o senso comum sobre qualidade dos dados, existe uma tendência em limitar a qualidade apenas ao critério de precisão dos dados [3], porém, existem outros critérios ou dimensões tão ou mais importantes que a precisão: atualidade; completeza; consistência; relevância; clareza; utilidade, entre outros [37], são exemplos de critérios da qualidade de dados.

No contexto de deduplicação, os mais relevantes [10] critérios de qualidade são:

- Acurácia / precisão - O dado deve representar exatamente o objeto real que mapeia.
- Completeza - Todos os dados, atributos, necessários para identificar um objeto real devem estar gravados na base de dados.
- Consistência - Estabilidade dos dados, tipo e conteúdo devem ser mantidos ao longo do tempo.
- Atualidade - O dado registrado em um ponto no tempo deve condizer com o estado atual do objeto real.
- Acessibilidade - Além de gravados, os dados devem estar disponíveis ao acesso.
- Credibilidade - os dados devem ser reais, possíveis e livres de alterações indevidas.

No contexto da deduplicação, os principais desafios giram em torno dos métodos para detectar defeitos que comprometem os critérios de precisão e consistência[10]. Dados imprecisos e inconsistentes são fontes de problemas e impulsionam os esforços de desenvolvimento das técnicas de deduplicação. São exemplos de problemas [23] que causam defeitos nos dados:

- **Redundância de valores do mesmo domínio:** A redundância ocorre quanto um único objeto do mundo real é mapeado para o virtual e é nomeado de duas ou mais maneiras distintas no mesmo domínio, por exemplo, para definir o sexo masculino, utilizar ora M, ora Masc, ora Masculino ou ainda o número 1;

- **Dados incompletos:** Dados que deveriam possuir um valor mas não o tem, por exemplo, um valor nulo para um campo que obrigatoriamente deva possuir valor;

- **Dados incorretos:** Dados que deveriam ser informados corretamente, mas que por motivos tais como: erro de digitação, falha de entendimento do dado informado, ou campo obrigatório e desconhecido pelo responsável, são informados incorretamente;

- **Dados armazenados não atomicamente:** Dados que deveriam ser armazenados de forma atômica, são armazenados em um único campo, por exemplo, armazenar o nome da rua e o nome do bairro em um único campo;

- **Duplicidades:** Registros que representam um único objeto do mundo real;

- **Inconsistências nos valores dos dados:** Dados armazenados em bases redundantes geralmente sofrem diferentes atualizações e podem ficar inconsistentes entre si, por exemplo, uma arquitetura visando disponibilidade e bases de dados redundante, ou seja, duas ou mais bases de dados que deveriam ser iguais em diferentes servidores e supondo uma atualização no cadastro de um cliente de uma destas bases, porém sem replicação para as outras.

Tais defeitos podem ser minimizados ou até evitados. Para a maioria deles, basta validar a entrada de dados. Quando se trata de registros duplicados, mesmo com soluções de busca ou de detecção de registros já existentes na base, utilizando um identificador único, as duplicidades ainda podem acontecer. Raramente acontecem falhas nos métodos de busca para verificar se o registro já existe, na maioria das vezes, a falha ocorre devido ao fator humano. Erros de digitação, preenchimento incorreto acidental, ou proposital, visando contornar campos únicos e obrigatórios, meta na produtividade e não na qualidade, são exemplos de falhas humanas.

Um dos principais erros de preenchimento incorreto acidental é o erro de digitação, que pode ser dividido em três classes [16]: erros linguísticos, de transmissão e tipográficos. Os erros linguísticos são aqueles que ocorrem devido a imperícia ou falta de informação daquele que escreve, os erros de transmissão são aqueles que acontecem devido a erros

nos canais de transmissão, por exemplo a passagem de uma imagem para texto e os erros tipográficos estão relacionados com erros de ordem motora no ato de digitar.

Os erros tipográficos podem ser, adicionalmente divididos em dois tipos, simples e complexos, os complexos apresentam mais de um erro simples na mesma palavra e os erros simples [16] são:

- Inserção - A palavra tem uma letra a mais. Exemplo: peedro.
- Substituição - uma letra é trocada por uma outra letra qualquer. Exemplo: jpsé.
- Omissão - a palavra tem uma letra a menos. Exemplo: mrcio.
- Transposição - duas letras na palavra tem sua ordem trocada. Exemplo: asndra.

CAPÍTULO 3

JUSTIFICATIVA

Neste Capítulo é apresentada a problemática envolvida, assim como as consequências da existência de dados duplicados no aplicativo CADSUS-multiplataforma, seguido de uma proposta de solução.

A seção 3.1 apresenta evidências da existência de dados duplicados, bem como as implicações negativas deste fato. Na seção 3.2 é descrita sucintamente a proposta de pesquisa deste trabalho para detectar as duplicidades.

3.1 Problema

Conforme descrito na seção 2.1.2, o CADSUS visa suportar o programa cartão nacional de saúde, que por sua vez, "tem como objetivo fundamental a identificação unívoca do usuário do SUS e o acompanhamento do conjunto de atendimentos realizados pelo sistema de saúde, onde quer que eles aconteçam..."[20]. Outro objetivo é organizar a estrutura fragmentada gerada pelos aplicativos desenvolvidos para atender programas específicos e eliminar os registros duplicados.

O objetivo de evitar duplicidades, utilizando o CADSUS, aparentemente não foi alcançado. Prova disto, pode ser obtida quando comparados o número de cadastros CADSUS do estado do Paraná e o número da população deste estado. Elaborada a partir de dados obtidos dos sítios do Ministério da Saúde e do IBGE[1], a Tabela 3.1, apresenta evidências das duplicidades.

A primeira linha da Tabela 3.1, datada de 03/01/2010 mostra o número total de definitivos e provisórios[2] dos cadastros do CADSUS naquela data, seguido do número de habitantes do estado do Paraná no ano de 2010, dados do IBGE. A segunda linha

[1]IBGE - Instituto Brasileiro de Geografia e Estatística

[2]Independente de definitivo ou provisório cada usuário deve ter apenas um cadastro.

traz as mesmas informações, mas, a data é 03/01/2012 e o número de habitantes é uma estimativa fornecida pelo IBGE.

Observa-se na última coluna da Tabela 3.1, os percentuais entre o número de cadastros do CADSUS e o número de habitantes do estado do Paraná, para o ano de 2010, o total de cadastros CADSUS, é 27,80% maior que a população recenseada pelo IBGE e em 2012, o número de cadastros aumentam para 35,12% acima da população.

Tais valores sugerem um alto grau de duplicidades, o número de cadastros no CADSUS é maior que o número total de habitantes do estado. Deve-se considerar que nem todos os cidadãos possuem cadastro no CADSUS; e ainda, nota-se o aumento do número de duplicados ao longo do tempo: de 2010 para 2012 o percentual passou de 27,80% para 35,12%.

Tabela 3.1: Paraná - Cadastros CADSUS x População

	Cadastros CADSUS				
Data	Definitivos	Provisórios	Total	IBGE	%
13/01/2010	6.340.264	7.008.574	13.348.838	10.444.526	127,80
03/01/2012	6.436.095	7.856.807	14.292.902	10.577.755	135,12

Fonte: MS/Datasus e IBGE

A Tabela 3.1, sugere as duplicidades e este cenário é conhecido pelo SUS. No portal de cadastros nacionais DataSUS[18] destaca-se a observação que o número de pacientes cadastrados para um município não corresponde ao número de pacientes do SUS daquele município, seguido da justificativa para tal: "Apesar de utilizarem (município) os aplicativos CADSUS, em alguns momentos não fizeram consulta na base nacional para verificar se o usuário já estava cadastrado ou realizaram procedimentos de cadastramento diferente do recomendado" [18]. Cadastrar os pacientes que visitou, consultando a base municipal e nacional para verificar sua existência, via de regra, é uma das funções do Agente Comunitário de Saúde - ACS. Os dados devem ser inseridos ou atualizados no CADSUS, mas em alguns casos, a inserção pode acontecer mais de uma vez para o mesmo paciente, gerando assim uma, ou mais, duplicidades. Conforme Figura 3.1, algumas situações aumentam o risco da duplicidade:

- a cada visita na UBS, o paciente faz um novo cadastro;

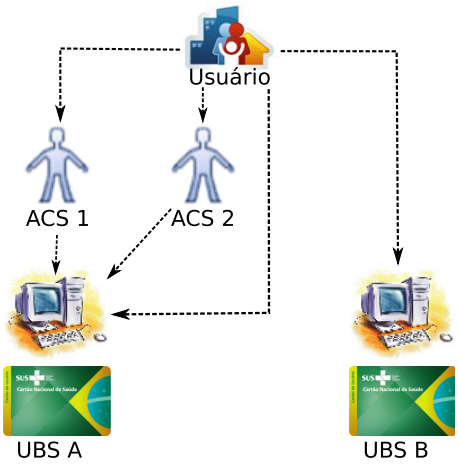

Figura 3.1: Situações de Risco - Duplicidades

- quando muda de bairro ou cidade o paciente faz um novo cadastro;
- um paciente recebe a visita de dois e cada ACS preenche um cadastro distinto;

O sistema CADSUS tem funcionalidades para evitar as situações de risco, por exemplo, a busca por pacientes já cadastrados pode ser feita foneticamente, ou seja, se feito uma busca por um paciente com o nome Osvaldo, caso exista, o resultado também retornará pacientes com o nome Oswaldo, ou a busca combinada, que é composta do primeiro nome do paciente, o primeiro nome de sua mãe e a data de nascimento do paciente.

Ainda mais, mesmo com a infraestrutura assíncrona, no CADSUS mono, desconectado do CADSUS centralizador, quando é feita uma importação de dados pelo CADSUS centralizador é possível fazer uma crítica indicando possíveis duplicidades.

Mesmo assim, as duplicidades podem ocorrer, e em essência, o motivo pode ser a não realização da busca, tanto na base de dados local, quanto na base de dados federal. Algumas são as explicações para tal, entre elas:

- o grande volume de trabalho do responsável por incluir o cadastro no CADSUS;
- falha no treinamento dos responsáveis pela busca e cadastro;
- negligência.

Vale citar que o cidadão tem garantia de acesso aos serviços de saúde [20]. Ou seja, mesmo que não tenha ou não porte o cartão SUS, o cidadão possui o direito a receber o

atendimento desvinculando-o da obrigação de manter um cadastro único.

As implicações de um cadastro duplicado podem ser várias, por exemplo, acompanhar o histórico clínico de um paciente que possua mais de um cadastro pode ser inviável. Provavelmente, os registros médicos estarão fragmentados em diferentes cadastros.

Algumas outras situações, contemplam a facilidade para que aconteçam erros, ou mesmo fraudes, por exemplo, um paciente com cadastros diferentes em duas UBS,(Figura 3.1) poderia solicitar e obter receitas e remédios nas duas UBS simultaneamente. E mais, na esfera administrativa, pode atrapalhar na requisição de financiamentos, comprometer a qualidade do planejamento de ações de saúde, e ainda, comprometer pesquisas epidemiológicas.

3.2 Proposta

Dada a atual infraestrutura do CADSUS-multiplataforma e considerando a evidência de cadastros duplicados nas bases de dados,(Tabela 3.1) a proposta deste trabalho é apresentar um método semi-automatizado para detecção de registros duplicados e possivelmente duplicados em bases de dados CADSUS municipais. Deste modo oferecer ao gestor da UBS do município, um método complementar, que liste os cadastros duplicados, para que em um segundo momento, não abordado neste projeto, seja possível realizar a sanitização e inativar os cadastros duplicados da base de dados.

CAPÍTULO 4

METODOLOGIA

Este Capítulo traz a proposta da metologia que foi empregada no processo de detecção de registros duplicados em uma base de dados CADSUS.

Na seção 4.1 é contextualizado os motivos da escolha da ferramenta FEBRL. A seção 4.2 apresenta particularidades do problema, justificando a escolha do método de pesquisa. O macro planejamento da execução e um breve resumo do que se obteve após a conclusão da pesquisa são apresentados nas seções 4.2.1 e 4.3.

4.1 Ferramentas

Existem algumas ferramentas que podem ser usadas para executar a análise e detecção de dados duplicados. Considerando o fator financeiro, ou seja, o custo de uma ferramenta para detecção de dados duplicados, este trabalho utilizou uma ferramenta sem custo financeiro. Diante isto, destacam-se as ferramentas descritas nas seções 2.2.7.2, 2.2.7.3 e 2.2.7.4, respectivamente: DuDe, FRIL e FEBRL.

Para a escolha da ferramenta que foi utilizada neste trabalho, foram considerados alguns pontos. O primeiro, é perspectiva de uso descentralizado por usuários que, em tese, desconhecem programação, habituados a trabalhar com interfaces gráficas. Também é importante que a ferramenta possua uma etapa de pré-processamento semi automatizada para limpeza e padronização dos dados e apresente um bom referencial bibliográfico [32] [10] para consulta em caso de dúvidas. Comparando as ferramenta citadas, enquanto a DuDe não possui uma interface gráfica, a FRIL possui, mas, Christen[10], sugere que esta apresenta limitações na etapa de pré-processamento. Logo, a ferramenta que mais se aproxima dos requisitos apontados como desejáveis e o propósito que se pretendia alcançar neste estudo, a ferramenta escolhida foi o FEBRL.

4.2 Método Aplicado

Utilizando a ferramenta FEBRL, uma base de dados real do CADSUS de um município, foi submetida as fases da Figura 2.2, quais sejam: pré-processamento; blocagem; comparação; classificação; revisão manual e avaliação dos resultados, conforme proposto por Christen [10].

Foram registradas e descritas as ações realizadas, assim como os recursos computacionais utilizados, os tempos de processamento, totais de registros duplicados e possivelmente duplicados. Encerrando, foram elaboradas as considerações, tanto dos procedimentos de execução quanto dos resultados alcançados.

4.2.1 Etapas da execução

A execução do trabalho foi realizada em 7 etapas, descritas a seguir.

- Etapa 1 - Aquisição da base de dados - Uma base de dados CADSUS-multiplataforma foi cedida pela Secretária Municipal de Saúde do município de Colombo Paraná, para a análise de deduplicação. Ainda, a secretária formalmente solicitou que a confidencialidade dos dados registrados na base deve ser mantida, ou seja, não deve ser divulgado nenhum tipo de informação que possa por em risco o usuário e sua privacidade. Vale citar que para a pesquisa, foram acessados e utilizados somente dados cadastrais dos usuários.

- Etapa 2 - Pré-processamento - O objetivo nesta etapa é selecionar e padronizar os atributos da base de dados para serem utilizados na deduplicação. Os dados padronizados foram utilizados como entrada para as etapas seguintes. Nesta etapa foram realizadas as seguintes ações:

  - Com base no dicionário de dados do sistema CADSUS, foi extraído da base de dados os atributos que serão deduplicados, convertendo-os para o formato CSV, suportado pelo FEBRL;

  - Foi selecionado e padronizado, quando necessário, um subconjunto de dados que foram comparados a procura de duplicidades.

- Etapa 3 - Blocagem/Indexação - Nesta etapa o objetivo foi selecionar um método de blocagem, seguido de uma chave de bloco e em função desta chave, distribuir os registros em blocos. Diminuindo o custo computacional da comparação, e, inclusive viabilizando o processo de deduplicação.

- Etapa 4 - Comparação - Na etapa de comparação, bloco a bloco, foi feita a comparação entre os registros pertencentes a cada bloco e para cada par de registros foi calculado o peso de similaridade, que indica o grau de similaridade entre eles.

- Etapa 5 - Classificação - Após a comparação dos registros, a próxima atividade foi classificá-los, para isto, foi definido os pesos limiares que foram utilizados como faixa de corte a fim de classificar os registros em duplicados, possivelmente duplicados e não duplicados.

- Etapa 6 - Avaliação e Revisão manual - Após todo o processo de deduplicação foi necessário apurar a qualidade dos resultados, nesta etapa, quando aplicáveis, foram apurados os indicadores de qualidade, seguido da revisão manual, ou seja, a verificação manual, do conjunto de possíveis duplicados, onde os pesos de limiar foram ajustados e adequados a base de dados analisada.

- Etapa 7 - Considerações e resultados - Por fim, foram descritos os valores e as impressões sobre os resultados da deduplicação.

4.3 Resultados

Foi apresentado um método semi-automatizado para detecção de cadastros duplicados e possivelmente duplicados em bases de dados CADSUS-multiplataforma de forma a possibilitar o uso descentralizado, pelos próprios operadores do sistema CADSUS. Para que estes tenham as mãos um processo complementar que os auxilie a detectar cadastros duplicados.

Subjetivamente, foi apresentado que este método pode ser aplicado a outras situações: ser utilizado como etapa inicial para um processo de sanitização da base de dados do município; auxiliar na detecção de dados duplicados em uma infraestrutura assíncrona, ou seja, uma base secundária desconectada da base principal, unidas por importação/exportação de dados; pré-filtro para a inserção de dados, indicando o grau de similaridade de um registro incluído com registros já existentes na base de dados; para detecção de dados duplicados em qualquer base de dados da saúde que tenha o mesmo padrão dos registros CADSUS;

CAPÍTULO 5

PROCEDIMENTO PARA DEDUPLICAÇÃO - CADSUS

Nos Capítulos anteriores foram apresentadas considerações sobre qualidade de dados, evidências de duplicidades em bases de dados CADSUS, conceitos de deduplicação, o método de deduplicação e como este é implementado na ferramenta FEBRL, entre outros. Este Capítulo descreve o uso das funcionalidades da ferramenta FEBRL, que é utilizada em todas as etapas do processo, para a deduplicação de uma base de dados CADSUS-Multiplataforma.

A primeira seção 5.1 apresenta informações sobre o ambiente e a instalação do FEBRL. Na seção 5.2 é descrito o processo de extração e seleção dos dados, atributos, utilizados para a deduplicação. O processo de padronização dos campos é apresentado na seção 5.3, a distribuição dos registros em blocos de similares é mostrado na seção 5.4. Em seguida, na seção 5.5, é apresentado o método de comparação entre registros. A seção 5.6 traz a etapa de classificação e as seções 5.7 e 5.8 tratam a avaliação da qualidade e o processo de revisão manual. Encerrando a seção 5.9 faz considerações sobre os resultados da deduplicação.

5.1 Instalação e execução FEBRL

Disponível para download em http://sourceforge.net/projects/febrl/ junto ao FEBRL está a documentação que traz os procedimentos de instalação passo a passo.

Quanto a execução, a ferramenta é multiplataforma, pode ser executado tanto em Linux[1] quanto em Windows[2]. A última versão, 0.4.2, disponibilizada em 14/12/2011, possui interface gráfica, facilitando o preenchimento dos parâmetros.

Inicialmente, a interface gráfica é um pouco confusa, pois, a maioria dos parâmetros estão distribuídos em abas ocultas, que se tornam visíveis a medida que os parâmetros

[1]http://br-linux.org/2008/01/faq-linux.html
[2]http://windows.microsoft.com/pt-br/windows/home

são preenchidos e os procedimentos são executados. A operação da interface gráfica pode ser vista em [8].

A interface gráfica possibilita salvar as configurações utilizadas em um arquivo texto, codificado em python, tornando possível a guarda da memória das configurações utilizadas para posterior execução do projeto por linha de comando. Desta forma, detalhes de preenchimento das telas não serão descritos, pois as configurações utilizadas para este projeto estão disponíveis no Apêndice E.

5.1.1 Ambiente

Para efetuar a deduplicação foi utilizado um computador com as configurações:

- **Hardware:**
- Intel(R) Core(TM) i5-2410M CPU @ 2.30GHz
- Memória RAM: 8 GiB SODIMM DDR3 Síncrono 1333 MHz
- **Software:**
- Sistema Operacional: Ubuntu[3] - v13.04 - 64 bits
- Python 2.7
- FEBRL 0.4.2
- PostgreSQL 9.1

5.2 Extração e seleção dos dados

A primeira ação foi analisar a estrutura da base de dados e a documentação da crítica dos dados [15] do CADSUS multiplataforma que descreve as tabelas e campos da base. Realizou-se uma pré-seleção dos atributos candidatos para a deduplicação, procurando por campos que contenham dados que possam identificar inequivocamente um usuário. O resultado desta análise são os atributos apresentados no Quadro 5.1.

[3]http://ubuntu-br.org/

Quadro 5.1: Atributos candidatos para análise da deduplicação

co_usuario	código único do usuário
no_usuario	nome do usuário
co_rg	rg do usuário
no_mae	nome da mãe
no_pai	nome do pai
dt_nascimento	data de nascimento
no_municipio_nas	município de nascimento
no_sg_uf_nas	sigla de uf de nascimento
nu_ddd_end	ddd telefone do endereço
nu_telefone_end	telefone do endereço)
nu_ddd	ddd telefone1 do usuário
nu_telefone	telefone1 do usuário
nu_ddd_2	ddd telefone2 do usuário
nu_telefone_2	telefone2 do usuário
co_sexo	sexo
ds_tipo_logradouro	tipo de logradouro
no_logradouro	nome do logradouro
nu_logradouro	numero do logradouro
no_compl_logradouro	complemento do logradouro
no_bairro	nome do bairro
co_cep	cep
no_municipio_res	município da residência
no_sg_uf_res	uf da residência

Os atributos do Quadro 5.1 foram classificados como candidatos, pois, nem todos estão aptos a serem utilizados na verificação de duplicidades. Como exemplo, considere atributos que possuam muitas repetições de valores, como “null”, “ignorado”, “não informado”, etc. Estes são inúteis para o processo, pois não indicam indícios de similaridades. Por exemplo, o atributo RG naturalmente parece ser apto para a identificação de registros duplicados, porém, neste conjunto de dados, por tratar-se de um atributo não obrigatório, possui muitos registros sem valor, logo, é inapto para a deduplicação.

Por esta razão, o próximo passo foi identificar quais campos podem e devem ser descartados. Para cada atributo do Quadro 5.1, foi verificada a sua representatividade na base de dados. Foram informados para a ferramenta os valores de atributos que devem ser considerados repetições ou ruins/faltantes: null, ignorado, ignorada e não informado.

Dado ao volume de registros, atributos quais os valores úteis, representassem 95% ou menos dos registros, foram desconsiderados. Desta maneira foi obtida o Quadro 5.2

que contém os atributos aptos para análise. A Figura 5.1 mostra um trecho do relatório, que, entre outras informações[4], traz uma avaliação final, classificando os atributos em aptos e inaptos para deduplicação.

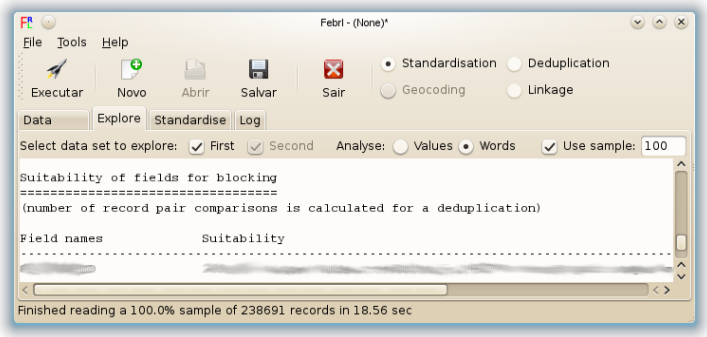

Figura 5.1: Aba explore

Fonte: FEBRL

Durante o processo de análise, observou-se que alguns atributos eram redundantes e portanto desnecessários para o procedimento, sendo estes: ds_tipo_logradouro, nu_logradouro, no_bairro e co_cep, que devido a relação com o atributo no_logradouro, podem ser substituídos por este.

Quadro 5.2: Atributos aptos para análise da deduplicação

co_usuario	código único do usuário
no_usuario	nome do usuário
no_mae	nome da mãe
dt_nascimento	data de nascimento
no_municipio_nas	município de nascimento
no_sg_uf_nas	sigla de uf de nascimento
co_sexo	sexo
no_logradouro	nome do logradouro
no_municipio_res	município da residência
no_sg_uf_res	uf da residência

5.3 Padronização dos dados

Após selecionar os atributos mais indicados para deduplicação (seção 5.2), a próxima ação é a padronização.

[4]Verificar manual do FEBRL para mais detalhes do relatório.

Inicialmente é necessário verificar quais os atributos necessitam ser padronizados e quais precisam ser segmentados. Atributos validados na entrada de dados ou valores selecionáveis, como sexo ou cidades, já estão padronizados, pois o usuário não digitará um valor errado. Quanto a segmentação, atributos atômicos, como município nascimento, não necessita ser segmentado pois já esta em sua menor forma. Assim atributos com estas características não necessitam padronização.

Os atributos aptos foram avaliados com base nos critérios citados no parágrafo anterior e no Quadro 5.3 estão apresentados aqueles que precisam ser padronizados e segmentados. Estes são atributos de livre digitação, logo, podem apresentar erros tipográficos e também não estão na sua forma atômica. Por exemplo, nome do usuário deve ser dividido em nome e sobrenome.

Quadro 5.3: Atributos a serem padronizados/segmentados

no_usuario	nome do usuário
no_mae	nome da mãe
no_logradouro	nome do logradouro
dt_nascimento	data de nascimento

Para os demais atributos é desnecessário a padronização. Exemplo, os atributos município da residência e uf da residência, estão segmentados e padronizados, isto é, no primeiro atributo, somente são armazenados os nomes dos municípios e no segundo somente as siglas dos estados. Quanto a validação, ambos foram validados no aplicativo CADSUS, pois, o usuário pode apenas selecionar um valor da lista (estados ou municípios).

Vale citar que o procedimento para padronização dos atributos do tipo nome e endereço é semelhante, logo, será descrito uma única vez, utilizando o atributo nome. Ainda o atributo data de nascimento será somente segmentado para a extração do ano de nascimento, necessário na etapa de blocagem (seção 5.4).

Nas subseções a seguir são apresentados as especificações dos pré-requisitos para a padronização dos dados. A subseção 5.3.1 trata as listas de correções, a subseção 5.3.2, o processo de criação das tabelas de *look-up*, já a subseção 5.3.3 aborda a criação do modelo HMM, por fim, a subseção 5.3.4 apresenta o resultado final da padronização: a segmentação dos campos em atributos.

5.3.1 Listas de correção

Apresentada na subseção 2.2.1.1, o objetivo de uma lista de correção é auxiliar no processo de remoção de caracteres/palavras desnecessárias (exemplo: abreviaturas), ou que podem gerar dúvida na comparação. Uma lista de correção é um arquivo texto. As linhas possuem valores substitutos, seguidos de valores substituíveis. Caso seja encontrada alguma correspondência entre os valores substituíveis e os valores do atributo, o valor encontrado será substituído pelo valor substituto.

Devido a existência de caracteres ora acentuados ora não, abreviaturas, caracteres como ponto, interrogação, entre outros na base, foi criada uma lista de correção. Baseando-se na lista de correção que acompanha a ferramenta FEBRL, uma cópia do arquivo foi gerada e a ela adicionada caracteres considerados desnecessários para deduplicação: ". , ; - ?", incluiu-se a substituição de caracteres acentuados, por seu correspondente sem acento e também a troca dos algarismos, do 1 até 21, pela sua forma textual. Um trecho da lista de correção pode ser visto no Quadro 2.2 (pág. 13).

5.3.2 Tabelas *look-up*

Mais completas que as listas de correções, as tabelas *look-up*, descritas em detalhes na subseção 2.2.1.2, são utilizadas para substituir possíveis palavras erradas. Mais que isso, armazenam a relação entre palavras e suas respectivas *tags*. As listas auxiliam tanto no treinamento do modelo HMM, quanto na segmentação de um atributo. Quanto mais completas e precisas, melhor será a qualidade dos resultados.

Uma tabela *look-up* contém uma ou mais *tag(s)* identificando um conjunto de palavras, a estrutura da tabela pode possuir uma *tag* por arquivo, ou várias *tags* em um único arquivo, separadas pelo conjunto de palavras que cada *tag* identifica. As *tags* são úteis para o processo de segmentação dos atributos e estão relacionadas aos *token*s, pois, durante a segmentação de um atributo, ao invês de se avaliar a sequência de *token*s, é avaliado a sequência de *tags* (Quadro 2.5 - pág. 14). Já as palavras, estão dispostas na forma de lista de palavras, as substitutas que estão corretas seguidas das substituíveis, ou incorretas, que são as utilizadas para a correção de erros comuns.

Para este trabalho, dado os atributos do Quadro 5.3, que podem ter erros de digitação e a necessidade de segmentá-los, e, considerando o domínio dos atributos nomes e endereços, foram criadas 14 tabelas *look-up*, conforme amostra no Quadro 5.4, sendo, 6 tabelas para atributos referentes a nomes e 8 para endereços.

Quadro 5.4: Amostra das tabelas *look-up* geradas para este trabalho

Nomes femininos	tag=<GF> # Tag para nomes femininos glaucia : glaaucia,glauucia,glucia,glauciaa,glauucia,...
Nomes masculinos	tag=<GM> # Tag para nomes masculinos angelo : angeloo,angwlo,angrlo,anelo,angeelo,aangelo,...
Sobrenomes	tag=<SN> # Tag para sobrenomes demetri : demwtri,demrtri,deetri,demeetri,demwtri,demeti,...
Diversos	tag=<BO> # Tag para sequenciais similiares a "filho" filho: filho de tag=<SP> # Tag para separadores e :
Prefixo	tag=<PR> # Tag para prefixos de :
Títulos	tag=<TI> # Tag para títulos doutor : dr
Tipo Logradouro	tag=<WT> # *tag* para tipo de logradouro avenida : av.,av,avinida,avenidaa,avwnida,avenids,aveenida,...
Logradouros	tag=<WN> # *tag* para nomes de logradouros isidoro mikosz : isidoro mikoosz,isiidoro mikosz,...
Complemento	tag=<UT> # *tag* para informações complementares conjunto : cj, conj, conjunto habitacional,...
Bairros	tag=<LQ> # *tag* para nomes de bairros atuba : atubaa,atuuba,atubs,atubs,atyba,...
Instituições	tag=<IT> # *tag* identificadores de instituição cooperativa : coperativa,coopreativa, coopeativa...
cep	tag=<PC> # *tag* para ceps 83401002 :
Cidades	tag=<LN> # *tag* para nomes de cidades colombo :
Território	tag=<TR> # *tag* para unidades da federação pr :

As primeiras linhas do Quadro 5.4 apresentam trechos de tabelas *look-up* relacionadas com nomes. Para o preenchimento destas, utilizou-se sítios de nomes na internet. Para nomes femininos e nomes masculinos foram utilizados sítios que publicam significados de nomes para bebês, exemplo: http://www.visionvox.com.br/biblioteca/n.htm. A tabela sobrenomes, foi extraída de sítios de genealogia, exemplo: http://www.fidelis-

soares.com/surnames-oneletter.php?firstchar%3D. As tabelas diversos, prefixos, títulos e instituições foram reaproveitadas do trabalho de [32]. As tabelas tipo logradouro, logradouros, complemento, bairros, cep, cidades e território foram populadas a partir de informações dos sítios dos correios, prefeituras de Colombo e Curitiba e do IBGE.

As tabelas logradouros, bairros e cep foram populadas exclusivamente com dados das cidades de Colombo e Curitiba, pois os dados de endereço de residência destas duas cidades representam juntos 99,35% de todos os endereços do conjunto de dados a deduplicar, sendo o restante, 0,65% distribuídos entre outros 472 municípios brasileiros.

Quanto ao número de entradas das tabelas de *look-up*, as mais significativas são: nomes femininos com 2.067 nomes, nomes masculinos com 5.158 nomes, sobrenomes com 10.028 sobrenomes, a tabela de logradouros com 10.658 nomes de ruas e a tabela de cidades com 5.283 nomes de cidades.

Para a criação dos valores substituíveis, incorretos, dado a quantidade de palavras e a impossibilidade de se criar ou prever manualmente erros comuns para todas as situações, tomou-se como base a conceituação dos principais erros tipográficos, descritos no final da seção 2.3 (pág. 34). Assim, utilizando as entradas consideradas corretas das tabelas de *look-up*, foram criadas variações[5]. Que podem ser consideradas incorretas, isto é, com possíveis erros de digitação, e para gerá-las foram utilizadas as seguintes regras:

- Inserção: para cada palavra foi gerado cinco outras, duplicando as vogais a e i o u.
- Substituição: para cada palavra foi criado cinco outras, substituindo as vogais **a** por **s**, **e** por **w**, **e** por **r**, **u** por **y** e **o** por **p**.
- Omissão: excluir um caractere a partir do terceiro caractere da palavra.
- Transposição: trocar a ordem de dois caracteres, o terceiro e quarto da palavra.

Devido a grande quantidade de palavras, a geração de valores substituíveis não foi realizada exaustivamente. Por exemplo, após aplicação das regras acima descritas, no universo de 17.253 nomes femininos, masculinos e sobrenomes, foram gerados aproximadamente 200.000 variações de palavras com possíveis erros tipográficos.

[5]Exceto as tabelas *look-up* cidades e territórios, pois não possuem erros no CADSUS.

5.3.3 Criação do modelo escondido de Markov - HMM

Para realizar a segmentação utilizado o FEBRL, para qualquer atributo com mais de três *token*s é necessário utilizar a segmentação estatística, vista na seção 2.2.2.2, porém, para executar o procedimento é necessário ter um modelo escondido de Markov, ou seja, as matrizes de transição de estado e símbolos de saída, apresentadas na seção 2.2.2.3. Como não foi encontrado um modelo escondido de Markov de um projeto semelhante, a próxima ação foi a criação de um modelo para o projeto, conforme apresentado a seguir.

Criar um HMM, é um procedimento demorado, pois, para alimentar as matrizes de transição de estados e símbolos de saída (Tabelas 2.1 e 2.2), é necessário treinar o modelo.

O FEBRL oferece a funcionalidade para criação do HMM, porém este processo é feito em parte na interface visual e em parte na linha de comando, com a utilização do programa *trainhmm.py* que acompanha a instalação da ferramenta[6].

Para treinar o modelo HMM, utilizamos a abordagem descrita em [12], pois, a mesma é utilizada para treinamento de um HMM de uma base semelhante e também foram aproveitadas as funcionalidades oferecidas pelo pacote FEBRL. Os passos para treinar e criar este modelo foram:

1. Selecionar aleatoriamente 100 registros do conjunto a deduplicar.

2. Com o FEBRL processar os 100 registros, selecionando os atributos que serão segmentados, informando a lista de correção e tabelas de *look-up* e gerando um primeiro arquivo de treinamento. Exemplo: Quadro 2.8 - pág. 19.

3. Ajustar o arquivo de treinamento, verificando para cada sequência de *token*s, se cada estado corresponde ao *token* e, neste primeiro conjunto de treinamento, incluir manualmente, após o estado, o símbolo de saída (Quadro 2.8).

4. Executar o programa *trainhmm.py* para processar o conjunto treinamento ajustado e gerar um HMM, chamado HMM100.

5. Selecionar aleatoriamente 1000 registros;

[6] O funcionamento do programa *trainhmm.py* é detalhado nas linhas iniciais do próprio código.

6. Com o FEBRL processar os 1000 registros, utilizar o HMM100 da etapa anterior e gerar um novo arquivo de treinamento.

7. Ajustar o novo arquivo de treinamento, verificando para cada sequência de *token*s, se cada estado e símbolo corresponde ao *token*. Caso algum símbolo não seja encontrado, incluí-lo.

8. Executar o programa *trainhmm.py* para processar o novo conjunto treinamento ajustado e gerar um HMM, chamado HMM1000.

9. Selecionar aleatoriamente 10.000 registros

10. Com o FEBRL processar os 10.000 registros, utilizar o HMM1000 e gerar um arquivo de probabilidades.

11. Analisar o arquivo de probabilidades procurando por ocorrências de registros que estejam fora do padrão e por padrões de registros ainda não processados nas amostras anteriores. Editar o conjunto de treinamento e incluir os padrões encontrados.

12. Executar novamente programa *trainhmm.py* para processar conjunto treinamento editado e gerar novo HMM.

13. Refazer o processo com novos 10.000 registros aleatórios mais duas vezes a fim de refinar a matriz de transição de estados, exemplo: Tabela 2.1 e matriz de símbolos de saída. Exemplo: Tabela 2.2.

Ao final da execução do procedimento, foi criado um conjunto treinamento com 1.796 registros, que após processado pelo programa *trainhmm.py* gerou dados para composição da Figura 5.2 e as Tabelas 5.1 e 5.2.

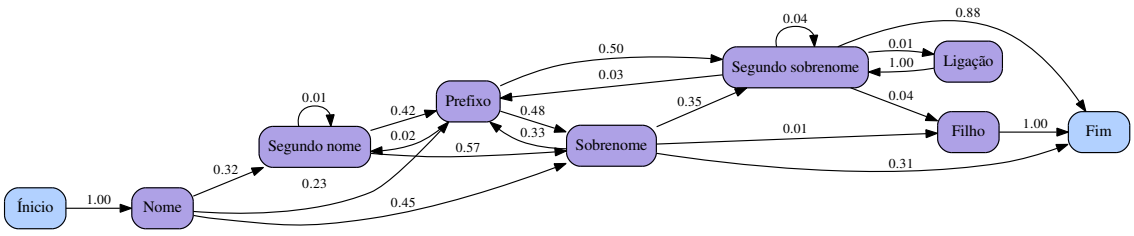

Figura 5.2: HMM de nome

Fonte: autor

Tabela 5.1: Matriz de transição de estados

estado "de"	estado - "para"							
	nome	segundo Nome	sobrenome	segundo Sobrenome	prefixo	ligação	filho "de"	fim
início	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
nome	0.0	0.32	0.45	0.0	0.23	0.0	0.0	0.0
segundo nome	0.0	0.01	0.57	0.0	0.42	0.0	0.0	0.0
sobrenome	0.0	0.0	0.0	0.35	0.33	0.0	0.01	0.31
segundo sobrenome	0.0	0.0	0.0	0.04	0.03	0.01	0.04	0.88
prefixo	0.0	0.02	0.48	0.50	0.0	0.0	0.0	0.0
ligação	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
filho (de)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

Fonte: autor

Tabela 5.2: Matriz de símbolos de saída

símbolo de saída	estado						
	nome	segundo nome	sobrenome	segundo sobrenome	filho (de)	prefixo	ligação
PR	0.01	0.01	0.01	0.01	0.01	0.92	0.01
SP	0.01	0.01	0.01	0.01	0.01	0.01	0.93
BO	0.01	0.01	0.01	0.01	0.93	0.01	0.01
II	0.01	0.01	0.01	0.01	0.01	0.01	0.01
GF	0.18	0.31	0.01	0.02	0.01	0.01	0.01
UN	0.39	0.12	0.22	0.11	0.01	0.01	0.01
SN	0.01	0.01	0.67	0.65	0.01	0.01	0.01
GM	0.38	0.52	0.06	0.18	0.01	0.02	0.01

Fonte: autor

5.3.4 Segmentação

Cumpridos os pré-requisitos da padronização, foram escolhidos para padronização e segmentação os atributos nome do usuário, nome da mãe, data de nascimento e nome do logradouro, pois, os três primeiros são utilizados combinados no aplicativo CADSUS para localizar cadastros existentes e o atributo, nome do logradouro, não é validado na entrada de dados do CADSUS.

Ainda, para compor a base a ser deduplicada, também foram selecionados os atributos já padronizados na entrada dos dados código único do usuário, data de nascimento[7], município de nascimento, sigla de uf de nascimento, município da residência e uf da resi-

[7]O atributo data de nascimento foi utilizado duas vezes, segmentado em dia, mês e ano para auxiliar na etapa de blocagem.

dência.

O processo de segmentação resultou em uma cópia da base de dados, com os atributos: nome do usuário, sobrenome do usuário, nome da mãe, sobrenome da mãe, ano de nascimento, nome do logradouro, data de nascimento, município de nascimento, sigla de uf de nascimento, município da residência, uf da residência e código único do usuário padronizados e preparados para as próximas etapas.

5.4 Blocagem

Uma das mais importantes decisões no processo da deduplicação é a escolha do(s) atributo(s) chave de bloco, pois em função da chave de bloco, serão criados os blocos de registros, para que as comparações entre registros sejam efetuadas somente entre os registros integrantes de cada bloco.

Para este projeto, foram escolhidos os atributos sobrenome do usuário e ano de nascimento para formarem uma chave de bloco composta. Dentre os atributos disponíveis, estes foram escolhidos, pois organizam os blocos em sobrenomes similares distribuídos por ano de nascimento. Como característica principal, a chave de bloco escolhida gera um maior número de blocos, porém, com o uso do ano, os registros são distribuídos de forma mais uniforme e precisa.

Para a codificação fonética da chave de bloco, foi utilizado o algoritmo *Soundex*, descrito na seção 2.2.3.2. Quando comparado com os disponíveis no FEBRL, é o mais genérico e apresenta poucas regras de codificação, lembrando que as regras de codificação dos outros algoritmos atendem regras fonéticas de idiomas que não o idioma brasileiro.

Dado as técnicas de blocagem listadas na seção 2.2.3.3 e o estudo comparativo entre elas, apresentado em [10]. Este estudo compara a aplicação das técnicas em três diferentes bases de dados, utilizando como parâmetros o tempo de execução, o número de pares de registros candidatos, os possíveis duplicados, e a memória necessária. Ainda indica quais técnicas geram mais pares de registros candidatos, porém, que são lentas e consomem mais mémoria: *string map index, indexação q-gram* e *canopy clustering*, seguida das técnicas mais ágeis, porém com menor número de pares candidatos e menor consumo de memória:

blocagem padrão, *sorted neighbourhood* e *suffix array index*. Considera a boa relação entre os parâmetros tempo, pares candidatos e memória utilizada, apresentada no estudo pela técnica *suffix array index*, e ainda que esta é uma espécie de q-gram e que entre as mais ágeis é a que mais gera pares candidatos. Essas características justificam a sua escolha para ser utilizada neste trabalho.

Descrita na seção 2.2.3.3 a técnica *suffix array index*, permite limitar o número de pares de registros candidatos, possibilita que seja definido um tamanho mínimo de sufixo. Assim, visando obter melhores resultados, foi feito a blocagem com 4 tamanhos de sufixo, conforme resultados da etapa de blocagem apresentados na Tabela 5.3.

Tabela 5.3: Resultados Blocagem

Tamanho	Tempo		Quantidade	
Sufixo	Codificação	Indexação	Blocos	Pares registros
8	1,69s	6,38s	70,326	4,10 milhões
7	3,71s	6,75s	89,043	6,75 milhões
6	6,99s	6,93s	93,241	16,65 milhões
5	18,55s	7,24s	94,007	59,05 milhões

Fonte: autor

5.5 Comparação

Bloco a bloco, na etapa de comparação os atributos dos registros são comparados par a par e para cada comparação é calculado um peso de similaridade, em seguida, na etapa de classificação. Baseando-se no peso de similaridade de cada par de atributos é calculado um peso de similaridade entre os registros.

Para a comparação foram selecionados seis atributos: nome do usuário, sobrenome do usuário, nome da mãe, sobrenome da mãe, data de nascimento e sexo. Estes foram escolhidos pois, exceto sexo, são utilizados no aplicativo CADSUS na busca de cadastros existentes.

Os demais atributos, ainda que preparados, não foram utilizados na deduplicação, pois ao invés de ajudar no processo, podem criar distorções no peso de similaridade para a etapa de classificação. Por exemplo, se acrescentado o atributo município de nascimento e feito a comparação para dois pares de registros, o primeiro par composto por dois registros

de irmãos gêmeos, com nome de usuário diferente, e sobrenome de usuário, nome da mãe, sobrenome da mãe, data nascimento, sexo e município de nascimento iguais. Neste cenário é calculado um peso de similaridade entre os registros de 5,94. Já um segundo par de registros, realmente iguais, somente com o nome da mãe com um único caractere de diferença e município de nascimento diferente, é calculado um peso de 5,8.

Neste caso, utilizando o atributo município de nascimento, os registros efetivamente mais similares, o segundo par (peso 5,8), ficou em desvantagem frente aos registros diferentes, gêmeos (peso 5,94), ao passo, que se desconsiderado o atributo município de nascimento, o primeiro par de registros receberia um peso de 4,94, e o segundo manteria o 5,8, assim refletindo precisamente a realidade.

Ainda, para comparar os atributos foi necessário selecionar o tipo de algoritmo de comparação. Para os primeiros quatro atributos, por serem de livre digitação, do tipo *string*, para eles foi utilizado o algoritmo de comparação Winkler, pois dentre os algoritmos de comparação de strings (seção 2.2.4), ele apresenta melhores resultados [10].

A entrada de dados do sexo no aplicativo CADSUS é controlada, assim, foi escolhido o algoritmo de comparação *exact string comparison*, pois não existe erro de digitação, logo, ou os valores do atributo são iguais ou diferentes. Por fim, para o atributo data nascimento, do tipo data, foi selecionado o algoritmo *date comparison*.

Selecionado os parâmetros desta etapa, foi realizada a comparação dos atributos conforme resultados apresentados na Tabela 5.4.

Tabela 5.4: Resultados Comparação

Tamanho Sufixo	Pares de Registros	Tempo Processamento	Mémoria Utilizada
8	4.10 milhões	12 min	1.91 GB
7	6.75 milhões	20 min.	3.04 GB
6	16.65 milhões	51 min.	6.91 GB
5	59.05 milhões	720 min.[7]	N. Disponível[8]

Fonte: autor

[7]Aproximadamente 12 horas.

[8]Esgotou as memórias virtual e swap, e foi necessário dividir o arquivo e processar em partes menores.

5.6 Classificação

Na etapa de classificação os registros são separados em duplicados, não duplicados e possíveis duplicados. Para esta etapa foi selecionado o método de classificação de Fellegi e Sunter, apresentado na seção 2.2. Ainda que atualmente a ferramenta FEBRL possua outros métodos de classificação, o método de Fellegi e Sunter é um método amplamente utilizado e testado na deduplicação de dados. Presente no FEBRL desde sua criação e dado sua simplicidade, mostra-se adequado a este projeto, pois possibilita ao usuário analisar os resultados e definir os valores de limiar que serão a faixa de corte para classificação.

Na primeira execução da classificação foi arbitrado dois limiares de corte, 4,8 e 5,3, e após a revisão manual (seção 5.8), este valores foram ajustados para 4,44 e 4,70, gerando os resultados apresentados na Tabela 5.5.

Tabela 5.5: Resultados Classificação

Tamanho Sufixo	Possivelmente Duplicados	Duplicados	Tempo Processamento
8	881	32.645	$\approx 2min$
7	904	32.965	$\approx 2min$
6	921	33.047	$\approx 3min$
5	973	33.368	$\approx 8min$

Fonte: autor

5.7 Avaliação

O processo de avaliação visa verificar a qualidade da deduplicação realizada, no entanto, para que seja calculado os indicadores de qualidade, é pré-requisito que um conjunto de registros sejam identificados inequivocamente, como duplicados ou não (seção 2.2.6).

O FEBRL possibilita o cálculo de métricas para a avaliação, porém, necessita que ao menos um atributo possa identificar inequivocamente situações de duplicidade. Mas, na base de dados CADSUS, nenhum dos atributos oferece esta característica, assim não foi possível calcular os indicadores de qualidade presentes no FEBRL.

Ainda assim, visando avaliar a qualidade, ou seja, o veracidade dos registros identificados como duplicados, foi realizado a revisão manual (seção 5.8), que apurou distorções no uso do atributo município de nascimento e no limiar de corte escolhido inicialmente. Ainda detectou características comuns ao grupo dos possivelmente duplicados, como apresentado na próxima seção.

5.8 Revisão Manual

Devido ao grande número de registros potencialmente duplicados encontrados na primeira faixa de limiar 4,8 a 5,3, ao todo 4.795 para o sufixo tamanho 8, foi necessário inserir os resultados em um banco de dados (Postgres), e com o auxílio de consultas *Structured Query Langage (SQL)*, utilizando o atributo código único do usuário como chave primária foi relacionado, listado e ordenado os registros apontados como iguais, desta forma aumentando a velocidade e execução da revisão manual.

Como já citado, inicialmente foi definido uma faixa de limiar entre 4,8 e 5,3, que se mostrou imprecisa, pois, analisando-se os registros com peso de classificação igual ou próximo a 4,8, constatou-se que todos os registros nesta faixa indicavam verdadeiras duplicidades. Assim, os limiares foram diminuídos, até encontrar um conjunto de possíveis duplicidades em que fosse necessário a avaliação humana, para indicar se eram ou não duplicidades.

Assim, a faixa de limiar para este trabalho foi definida entre 4,44 a 4,70. Os registros classificados até 4,44 são os não duplicados, a partir de 4,70, duplicados e para aqueles registros posicionados entre 4,44 e 4,70, foram detectados casos que geram dúvidas sobre a duplicidade ou não dos registros. Esses casos devem ser revisados manualmente para indicar se são ou não duplos. A seguir têm-se uma lista das principais situações apuradas:

- Nomes, sobrenomes, datas de nascimento e sexos iguais, mas um registro apresenta o nome e sobrenome da mãe, e o outro registro o nome e sobrenome do pai.

- Nomes, sobrenomes, datas de nascimento e sexos iguais, nomes e sobrenomes da mãe diferentes, ou vazios, ou não informado.

- irmãos gêmeos, primeiro nome diferente, sobrenomes, datas de nascimento e dados mães iguais, sexos iguais.
- irmãos gêmeos, primeiro nome diferente, sobrenomes, datas de nascimento e dados mães iguais, sexos diferentes.
- Irmãos de mesmo sexo, com o primeiro nome diferente, mesma mãe e datas de nascimento diferente.
- Irmãos de sexo diferente, com o primeiro nome diferente, mesma mãe e datas de nascimento diferente.
- Primeiro nome diferente, primeiro nome da mãe diferente, sobrenome usuário, sobrenome mãe, datas de nascimento e sexo iguais.
- Primeiro nome e datas de nascimento diferentes, mãe, sobrenomes e sexo iguais.
- Primeiro nome diferente, sobrenomes, datas de nascimento e sexo iguais, sobrenome da mãe de solteira diferente do sobrenome de casada.

5.9 Considerações sobre os resultados

Na etapa de blocagem, a Tabela 5.3 mostra, que independente do tamanho do sufixo, o tempo para a geração dos blocos, no maior caso, aproximadamente 30 segundos, estão otimizados e executam o processamento em tempo aceitável. Ainda na blocagem, a medida que se diminui o tamanho do sufixo ocorre um crescimento significativo na quantidade de pares candidatos. O mesmo não ocorre para o número de blocos. Por exemplo, a mudança do tamanho de sufixo de 6 para 5, refletiu em aumento de 254,65% para os pares candidatos, enquanto que os blocos, para os mesmos sufixos, aumentaram apenas 0,82%.

A Tabela 5.4 apresenta os resultados da etapa de comparação. Nesta Tabela, nota-se principalmente a relação entre o tempo de processamento e memória utilizada. Nas situações que a memória é suficiente para processar o número de pares de registros, o

tempo de processamento aumentava, quase proporcionalmente ao número de pares de registros. Porém, para o sufixo tamanho 5, que extrapolou os recursos de memória, o tempo estimado de 5 horas, saltou para 12 horas, sendo necessário dividir o arquivo em partes menores, 14 ao todo, e executar o procedimento de comparação, arquivo a arquivo.

O resultado final da deduplicação é mostrado na Tabela 5.5. Chama a atenção o grande número de registros duplicados para o sufixo 5, 33.368, representando 13,98% dos 238.691 cadastros. Outro ponto interessante, é o baixo número de cadastros possivelmente duplicados, que apresentou um percentual de 0,40% do total de cadastros da base, mostrando assim a precisão do processo de deduplicação e também apontando que mesmo que seja necessária a intervenção manual para a detecção de duplicados, se considerado o número de cadastros, esta será mínima.

CAPÍTULO 6

CONCLUSÃO

6.1 Conclusões e trabalhos futuros

Conforme sugestão da existência de cadastros duplicados nas bases de dados CADSUS apresentada na seção 3.1, esta dissertação apresentou um método semi-automatizado para que o município, por intermédio de seus gestores de saúde, possa utilizar o método e identificar cadastros duplicados e os possivelmente duplicados.

Inicialmente, foi feita uma introdução sobre a recente explosão da quantidade dos dados e suas consequências. O enfoque partiu da duplicidade dos dados globalmente, passando para um campo mais específico, a duplicação de dados na área da saúde. Ainda foi realizado uma breve abordagem sobre os aplicativos da saúde brasileiros.

Num segundo momento, foram analisados os principais tópicos ligados ao domínio do problema, foram descritos conceitos sobre a unidade básica de saúde, a infraestrutura do CADSUS multiplataforma, conceitos de qualidade dos dados ligados a deduplicação, alguns métodos de deduplicação e visão global sobre alguns sistemas de deduplicação. Na sequência, foi apresentada a problemática específica, comprovando a existência de duplicidades nas bases de dados do aplicativo CADSUS-multiplataforma. Ainda, foram expostos o planejamento para a execução deste trabalho, especificando a metodologia, ferramentas de uso, quais os resultados esperados e etapas de execução. Na sequência, foram descritas as etapas de deduplicação propostas por Christen[10], detalhando as técnicas, pré-processamento, blocagem, comparação, classificação, avaliação e revisão manual.

Por fim, realizou-se a deduplicação de uma base de dados CADSUS-multiplataforma com 238.691 cadastros de usuários. Segundo o método aplicado, o resultado da deduplicação, apresentado na seção 5.6, mostrou que 33.368 cadastros, 13,98% de toda a base foram classificados como duplicados, e que outros 973, 0,4% de toda a base, foram classificados como possivelmente duplicados.

Os 33.368 cadastros detectados como duplicados segundo o critério apresentado na seção 5.6, podem ser efetivamente rotulados como duplicidades[1]. Porém, devido a possibilidade de falsos positivos, estes cadastros devem passar pela análise individual do município, para conferência. Conforme apresentado na seção 5.8, os outros 973 cadastros acusados como possivelmente duplicados, representam registros duvidosos, que podem ou não ser duplicidades, por exemplo, cadastros de irmãos gêmeos, logo, também precisam ser verificados.

Baseando-se nos resultados e na revisão manual (seção 5.8), foi constatado, como afirma a literatura, que as etapas de pré-processamento e blocagem são as mais importantes, pois influenciam diretamente nos resultados. O alto percentual de registros duplicados, o baixo percentual de possíveis duplicados (seção 5.6), e os tipos de casos que geraram dúvidas sobre a duplicidade ou não dos cadastros (seção 5.8), denotam tal importância.

Visando trabalhos futuros, pode-se observar as seguintes situações. Na etapa de pré-processamento, seção 5.3.3 especificamente a segmentação, foi necessário treinar o modelo escondido de Markov - HMM. No FEBRL, para segmentar um atributo em mais de dois atributos é necessário utilizar o modelo HMM, porém, notou-se que o esforço realizado para o treinamento do HMM, poderia ser evitado. Poderia existir a possibilidade de utilizar a segmentação baseada em regras para casos que o banco de dados possua um modelo de entidade relacionamento bem estruturado, como o caso do CADSUS-multiplataforma, e que os valores dos atributos obedeçam um padrão local de escrita, por exemplo o campo nome, escrito numa única ordem em todo o território nacional. Desta forma, ferramenta FEBRL poderia ser adpatada para estes casos.

Outra situação, foi a escolha do algoritmo de codificação para a chave de bloco *soundex* descrito na seção 5.4, o FEBRL não possui nenhuma opção que utilize o idioma brasileiro, logo, poderia ser agregado na ferramenta, algum algoritmo que use para codificação a fonética brasileira.

Ainda, previsto na ferramenta FEBRL, mas não implementado, o acesso a banco de dados e possibilidade de gravação dos resultados diretamente na base, com disponibilidade

[1]Selecionamos aleatoriamente 200 pares de cadastros dos 33.368 e todos eram verdadeiras duplicidades.

da execução de consultas *SQL*, facilitaria sobre maneira a análise dos resultados e quando necessário a reexecução do procedimento de deduplicação.

Com outro enfoque, no nível estratégico, há de se pensar, na adaptação e inserção de funcionalidades do FEBRL ao aplicativo CADSUS-multiplataforma, dotando-o de mais um pré-filtro para o cadastro de novos pacientes, para que no momento do cadastro, indique cadastros potencialmente duplicados, minimizando, ainda mais, a possibilidade de inserção de duplicidades.

Por fim, os números acima mostram que o processo de deduplicação apresentado, pode ser utilizado como um método auxiliar, inicial, ao processo de sanitização de uma base de dados CADSUS-multiplataforma municipal. O resultado do processo, uma lista de duplicidades e potenciais duplicatas, pode ser vista pelo município como como uma única lista de possíveis cadastros duplicados, que o município deverá corrigir e unificar manualmente no CADSUS, um a um, todos os cadastros apontados como duplicados. Ainda, durante a correção dos cadastros é importante que seja registrado eventuais erros[2] da deduplicação, para que estes sejam analisados e se possível, utilizados para melhorar o processo.

[2]Registros identificados como duplicidades, que não o são.

APÊNDICE A

FUNÇÕES DE CODIFICAÇÃO DA CHAVE - FEBRL

Além do *soundex* apresentado na seção 2.2.3.2 o FEBRL também possibilita o uso das seguintes funções fonéticas:

- *Phonex* - Variação do algoritmo *soundex* [30]. Antes de aplicar o método *soundex*, faz um pré-processamento da palavra e baseado em regras de pronuncia norte americana, substitui alguns caracteres no início da palavra.

- *Phonix* - Amplia a ideia do algoritmo *phonex* [10], além das alterações no inicio, possui regras de substituições para cadeias de caracteres no meio da palavra.

- *NYSIIS* - Igualmente baseado em regras de substituição, atrás em uso somente do *soundex*, o *New York State Identification and Intelligence System* gera um código exclusivamente de letras, com no mínimo três e máximo seis caracteres [10].

- *Double metaphone* - Os algoritmos citados tem uma desvantagem, foram desenvolvidos para atender as regras de pronuncia do idioma norte americano, logo, não são totalmente adequados a outros idiomas. Desta forma, este algoritmo foi desenvolvido visando atender a nomes europeus e asiáticos [10]. Igual o *phonix* e *NYSIIS*, o double metaphone também possui um grande número de regras, o código gerado retorna letras e para nomes que tenham diferentes pronuncias, pode retornar até dois códigos, como apresentado no Quadro 2.9 nome: smith.

- *Fuzzy Soundex* - Este algoritmo combina uma etapa de pré-processamento baseada em q-gram (geração de *substrings* de tamanho q [29]), com uma segunda etapa de utilização da lógica *soundex*. No pré-processamento este algoritmo faz a substituição de q-grams de tamanho dois e três na palavra, para em seguida gerar um código baseado em *soundex*.

APÊNDICE B

TÉCNICAS DE BLOCAGEM - FEBRL

As técnicas de blocagem descritas aqui, estão presentes no FEBRL, que também oferece possibilidade da indexação total *full index*, em que todos os registros são comparados com todos os registros, logo, não é utilizado blocagem.

- Blocagem padrão - Nesta técnica os registros não são duplicados em diferentes blocos [10], cada registro, em função da correspondência com a chave de bloco será alocado em um único bloco.

- *Sorted Neighbourhood* - Invés de utilizar uma chave de bloco, este método utiliza uma chave de indexação, criada do mesmo modo que a chave de bloco, [10] para ordenar e distribuir os registros, e uma janela de deslizamento w maior que 1, janela com valor 1, equivale a blocagem padrão. A janela define o quanto um bloco deverá avançar em relação ao próximo, isto é, quantos registros marginais do bloco a estarão também no bloco b.

- *Canopy clustering* - Este modelo de indexação utiliza técnicas de clusterização para criar blocos *clusters* de registros [14]. Igualmente ao método q-gram, neste modelo um registro pode participar de diferentes blocos. Os blocos são criados aplicando o conceito de q-gram para geração das chaves de blocos. As medidas de similaridade para realizar a distribuição dos registros nos blocos podem ser a abordagem Jaccard ou a abordagem TF-IDF *Term-Frequency / Inverse Document Frequency* [14].

- *String map index* - A ideia central desta técnica é a conversão dos valores de chaves de bloco em objetos [33], mapeando-os, sem prejuízo para o grau de similaridade entre os valores, para um espaço multi-dimensional indexado, seguido da aplicação de um método similar ao *canopy clustering* que irá inserir objetos similares em *clusters* comuns.

A grosso modo, o processo de mapeamento funciona da seguinte maneira: após a definição de um número de dimensões d, para cada dimensão é escolhido uma *string* ou chave, aleatoriamente, em seguida, uma função é executada diversas vezes, procurando-se por uma segunda *string* que seja os mais distante, diferente, possível da primeira, na sequência, em função destas duas strings, é calculado as coordenadas de distância para todas as outras strings, este processo é repetido para todas as dimensões, gerando d objetos para cada *string*. Em seguida, similarmente ao método *canopy clustering*, será aplicado a avaliação da distância dos objetos nas diferentes dimensões e gerados *clusters*, blocos de objetos similares.

APÊNDICE C

ALGORITIMOS DE COMPARAÇÃO - FEBRL

Além, dos algoritimos *Exact string comparison*, *Date comparison*, *Jaro approximate string comparison* e *Winkler approximate string comparison*, descritos na seção 2.2.4 o FEBRL, também traz os seguintes algoritmos [9] para comparação entre atributos:

- *Contains string comparison* - Dado duas *strings* de tamanhos diferentes, verifica se a menor *string* esta contida dentro da maior.

- *Truncate string comparison* - Permite que seja estabelecido um tamanho máximo para o número de caracteres da *string* que será comparado.

- *Key difference comparison* - Para o cálculo do peso da similaridade, esta função compara duas sequências de caracteres (pode ser utilizada para números) e calcula a similaridade em função do número de caracteres diferentes entre as sequências, por exemplo 12357 e 12851, tem dois caracteres diferentes, os pares 3,8 e 7,1. É possível definir um número de tolerância de caracteres diferentes por comparação.

- *Numeric percentage comparison* - Utilizada para atributos numéricos, calcula um percentual entre os dois números, se o percentual calculado for igual, será atribuído o peso da similaridade, caso o percentual for menor que um percentual máximo pré-definido, será calculado o peso de similaridade proporcional, caso contrário, percentual calculado maior, assume o peso da diferença definido pelo usuário.

- *Numeric absolute comparison* - A função encontra a diferença absoluta entre dois atributos numéricos e compara com um valor absoluto máximo pré definido pelo usuário, se a diferença absoluta for menor ou igual ao valor pré definido, é calculado um peso de similaridade proporcional, caso contrário, retorna um peso de diferença definido pelo usuário.

- *Encoded string comparison* - Esta função utiliza funções fonéticas, ver seção 2.2.3.1, isto é, antes da comparação, aplica uma função de conversão às *strings*, em seguida compara os códigos gerados, caso sejam iguais é atribuído o peso de similaridade para iguais, caso diferentes, o peso para diferentes.

- *Age comparison* - A partir de um percentual de diferença informado pelo usuário, a função compara duas idades e atribui um peso de similaridade. Caso seja informado a idade no formato data, é utilizado a data atual para calcular a idade.

- Time comparison - Compara horas no formato HHMM ou HH:MM, utiliza a mesma lógica da função *Date comparison*.

- *Q-gram approximate string comparison* - A função calcula a similaridade considerando o número total de q-grams comuns em ambas *strings* dividido pelo número mínimo, ou máximo ou a média de q-grams de ambas *strings*.

- *Positional Q-gram approximate string comparison* - Esta função é semelhante a q-gram, porém, não são contados todos os q-grams iguais, são contados somente aqueles que estiverem posicionados abaixo do limite posicional definido, ou seja, se na *string 1* na posicao 0 existe um bigram (q=2) "er" e na *string 2* existe uma correspondência na posição 4 "er", e o valor posicional for 2, estes não serão computados para o cálculo da similaridade.

- *Skip-gram approximate string comparison*, baseado em bigrams, além de gerar os bigrams com dois caracteres adjacentes, também cria bigrams com caracteres distantes entre si. Exemplo: dado "clara" com "cl","la","ar","ra" e um *skip* defindo em 1, geraria também "ca", "lr" e "aa".

- *Edit-distance approximate string comparison* - Variação da função *Levenshtein*, o cálculo da similaridade é feito a partir do cálculo da distância entre as duas *strings*, para isso é calculado o número mínimo de deleções, inserções ou substituições de caracteres para tornar uma *string* igual a outra.

- *Damerau-Levenshtein distance approximate string comparison* - Semelhante a função de *edit-distance* esta função adiciona a operação de transposição entre caracteres, ou seja, caso dois caracteres adjacentes das *strings* estejam transpostos, exemplo: “hl” e “lh”, estes serão contados como uma operação de ajuste (transpostos).

- *Bag distance approximate string comparison* - Podendo ser vista como uma simplificação da função *edit-distance*, para calcular a similaridade, esta função considera o número de caracteres diferentes entre as duas *strings* ordenadas.

- *Smith-Waterman edit distance approximate string comparison* - Baseado na função *edit-distance*, permite atribuir *scores* para alguns tipos de situações entre os caracteres das suas *strings*, por exemplo um $score = 5$ caso exista a correspondência exata entre dois caracteres, ou um $score = 2$ para caracteres considerados similares (exemplo, v e w).

- *Syllable alignment distance approximate string comparison* - O principio da função é dividir as *strings* em silabas e utilizando técnicas de conversão fonética com o método *edit-distance*, calcular o número de edições necessárias para converter uma sequência de silabas na outra.

- *Sequence match approximate string comparison* - Utiliza como fonte a função *sequenceMatcher* do módulo *difflib* do python. A função basicamente compara recursivamente cadeias de caracteres iguais entre as duas *strings*, gerando uma similaridade para as igualdades encontradas.

- *Editex approximate string comparison* - *Editex*, combina a codificação fonética, para converter as *strings* em código, seguido da aplicação do cálculo *edit-distance* nos códigos gerados.

- *Longest common substring approximate string comparison* - Neste modelo, são localizadas as *substrings* iguais em cada *string*, a partir dai, é feito a soma do número de caracteres iguais em cada substring, por fim é dividido, pelo número mínimo, máximo ou a média de caracteres de ambas strings.

- *Ontology longest common substring approximate string comparison* - A função inicia o cálculo utilizando o mesmo método da função *Longest common substring approximate string comparison*, em seguida calcula uma medida de diferença considerando os caracteres diferentes entre as *strings*, por fim faz um ajuste nos pesos, é uma variação da técnica Winkler, considerando as igualdades dos caracteres iniciais das duas *strings*.

- *Compression based approximate string comparison* - Utiliza um algoritmo para compactar as strings. Para calcular a similaridade, primeiro compacta as strings separadamente, em seguida, concatena as strings e as compacta e por fim faz uma comparação entre o tamanho das *strings* na primeira compactação e o tamanho da *string* resultante após a segunda compactação.

- *Token-set approximate string comparison* - São retiradas de ambas as *strings* todas palavras que estejam numa lista chamada *stop word list*, em seguida são totalizadas o número de palavras iguais nas *strings*, por fim é dividido, pelo número mínimo, máximo ou a média de palavras, somente as que restaram.

APÊNDICE D

MÉTODOS DE CLASSIFICAÇÃO - FEBRL

O FEBRL, disponibiliza algumas abordagens para classificação de registros, além da abordagem de Fellegi e Sunter, também traz as seguintes:

- O método *Optimal Threshold* é uma abordagem supervisionada, que automaticamente calcula os valores de limiar para classificação dos registros [9]. Para tanto, deve ser atendido uma condição inicial: a existência de, ao menos, um atributo que possa indicar com certeza alguma duplicidade ou diferença entre os registros. Utilizando os pesos de igualdade e diferença dos registros identificados como duplicados, a partir do atributo indicado, são calculados os melhores valores de limiar *optimal threshold*, e estes serão utilizados como entrada para o método de Fellegi e Sunter que será aplicado para a classificação do conjunto de dados.

- As abordagens não supervisionadas *KMeans* [31] e *FarthestFirst* [26, 17] são baseadas em clusterização e em função dos pesos de comparação que podem agrupar os registros em *clusters* de iguais e diferentes. As abordagens disponibilizam alguns métodos para seleção dos centroides e diferentes opções para o cálculo das distâncias, também possibilita a seleção de uma região *fuzzy*, entre os *clusters* de iguais e diferentes.

- O método *SuppVecMachine*, Máquina de vetores de suporte - SVM, é um método supervisionado e a partir de um conjunto treinamento, com indicações de iguais e diferentes, gera um modelo espacial com classes bem definidas, que são utilizadas para fazer a classificação dos registros. A implementação deste método no FEBRL utiliza a biblioteca *libsvm* [6] e da mesma forma que o método *Optimal Threshold* o método *SuppVecMachine* necessita de um atributo que identifique duplicidades e não duplicidades para criar o conjunto de treinamento.

- Por último, o método *TwoStep* é um método de classificação não supervisionado. Primeiramente é informado os pesos de pares de registros com alta probabilidade de serem iguais e os com alta probabilidade de serem diferentes. Num segundo momento, utilizando como conjunto treinamento os registros que se enquadrem dentro da faixa de valores informados, é utilizado o método *Kmeans* ou *SVM* para realizar a classificação.

APÊNDICE E

CONFIGURAÇÕES DEDUPLICAÇÃO - FEBRL

```
# ==========================================================
# AUSTRALIAN NATIONAL UNIVERSITY OPEN SOURCE LICENSE (ANUOS LICENSE)
# VERSION 1.3
#
# The contents of this file are subject to the ANUOS License Version 1.2
# (the "License"); you may not use this file except in compliance with
# the License. You may obtain a copy of the License at:
#   http://datamining.anu.edu.au/linkage.html
# Software distributed under the License is distributed on an "AS IS"
# basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See
# the License for the specific language governing rights and limitations
# under the License.
# The Original Software is: "Amostra01.py"
# The Initial Developers of the Original Software are:
#   Peter Christen
# Copyright (C) 2002 - 2011 the Australian National University and
# others. All Rights Reserved.
# Contributors:
# Alternatively, the contents of this file may be used under the terms
# of the GNU General Public License Version 2 or later (the "GPL"), in
# which case the provisions of the GPL are applicable instead of those
# above. The GPL is available at the following URL: http://www.gnu.org/
# If you wish to allow use of your version of this file only under the
# terms of the GPL, and not to allow others to use your version of this
# file under the terms of the ANUOS License, indicate your decision by
# deleting the provisions above and replace them with the notice and
# other provisions required by the GPL. If you do not delete the
# provisions above, a recipient may use your version of this file under
# the terms of any one of the ANUOS License or the GPL.
```

```
# ============================================================
# Start of Febrl project module: "Amostra01.py"
# Generated using "guiFebrl.py" on Tue Dec 10 18:08:35 2013
# ============================================================
# Import necessary modules (Python standard modules first, then Febrl
    modules)

import logging
import classification
import comparison
import dataset
import encode
import indexing
import measurements
import mymath
import output
import stringcmp
# ------------------------------------------------------------
# Intialise a logger, set level to info oe warning
log_level = logging.INFO # logging.WARNING
my_logger = logging.getLogger()
my_logger.setLevel(log_level)
# ------------------------------------------------------------
# Febrl project type: Deduplicate
# ------------------------------------------------------------
# Define input data set A:
data_set_a = dataset.DataSetCSV(description="Data set generated by Febrl
    GUI",
                                access_mode="read",
                                strip_fields=False,
                                miss_val=[''],
                                rec_ident="co_usuario",
                                file_name="/home/osvaldomc/ajuda/base/
                                    dbcnsVIRG-ajustado-padronizadoFEBRL.csv"
                                    ,
```

```
                                    header_line=True,
                                    delimiter=",",
                                    field_list = [("dia",0),
                                                  ("mes",1),
                                                  ("ano",2),
                                                  ("Nascimento",3),
                                                  ("nome_usuario",4),
                                                  ("sobrenome_usuario",5),
                                                  ("nome_mae",6),
                                                  ("sobrenome_mae",7),
                                                  ("nome_rua",8),
                                                  ("co_usuario",9),
                                                  ("no_usuario",10),
                                                  ("no_mae",11),
                                                  ("no_municipio_nas",12),
                                                  ("no_sg_uf_nas",13),
                                                  ("co_sexo",14),
                                                  ("no_bairro",15),
                                                  ("no_municipio_res",16),
                                                  ("no_sg_uf_res",17)])

# ------------------------------------------------------------------------
# Define field comparison functions
fc_funct_1 = comparison.FieldComparatorWinkler(agree_weight = 1.0,
                                               description = "Winkler-
                                                   nome_usuario-nome_usuario
                                                   ",
                                               disagree_weight = 0.0,
                                               missing_weight = 0.0,
                                               threshold = 0.5,
                                               check_sim = True,
                                               check_init = True,
                                               check_long = True)

fc_funct_2 = comparison.FieldComparatorWinkler(agree_weight = 1.0,
```

```
                                                    description = "Winkler-
                                                        sobrenome_usuario-
                                                        sobrenome_usuario",
                                                    disagree_weight = 0.0,
                                                    missing_weight = 0.0,
                                                    threshold = 0.5,
                                                    check_sim = True,
                                                    check_init = True,
                                                    check_long = True)

fc_funct_3 = comparison.FieldComparatorWinkler(agree_weight = 1.0,
                                                    description = "Winkler-
                                                        nome_mae-nome_mae",
                                                    disagree_weight = 0.0,
                                                    missing_weight = 0.0,
                                                    threshold = 0.5,
                                                    check_sim = True,
                                                    check_init = True,
                                                    check_long = True)

fc_funct_4 = comparison.FieldComparatorWinkler(agree_weight = 1.0,
                                                    description = "Winkler-
                                                        sobrenome_mae-
                                                        sobrenome_mae",
                                                    disagree_weight = 0.0,
                                                    missing_weight = 0.0,
                                                    threshold = 0.5,
                                                    check_sim = True,
                                                    check_init = True,
                                                    check_long = True)

fc_funct_5 = comparison.FieldComparatorExactString(agree_weight = 1.0,
                                                        description = "Str-Exact
                                                            -no_municipio_nas-
                                                            no_municipio_nas",
```

```
                                                    disagree_weight = 0.0,
                                                    missing_weight = 0.0)

fc_funct_6 = comparison.FieldComparatorDate(agree_weight = 1.0,
                                                description = "Date-Nascimento-
                                                    Nascimento",
                                                disagree_weight = 0.0,
                                                missing_weight = 0.0,
                                                max_day1_before_day2 = 1,
                                                max_day2_before_day1 = 1,
                                                date_format = "ddmmyyyy")

fc_funct_7 = comparison.FieldComparatorExactString(agree_weight = 1.0,
                                                    description = "Str-Exact
                                                        -co_sexo-co_sexo",
                                                    disagree_weight = 0.0,
                                                    missing_weight = 0.0)

field_comp_list = [(fc_funct_1, "nome_usuario", "nome_usuario"),
                   (fc_funct_2, "sobrenome_usuario", "sobrenome_usuario"),
                   (fc_funct_3, "nome_mae", "nome_mae"),
                   (fc_funct_4, "sobrenome_mae", "sobrenome_mae"),
                   (fc_funct_5, "no_municipio_nas", "no_municipio_nas"),
                   (fc_funct_6, "Nascimento", "Nascimento"),
                   (fc_funct_7, "co_sexo", "co_sexo")]

rec_comp = comparison.RecordComparator(data_set_a, data_set_a,
    field_comp_list)

#--------------------------------------------------------------------------
# Define indices for "blocking"
index_def_1 = [["sobrenome_usuario", "sobrenome_usuario", False, False, 7,
    [encode.soundex]],
               ["ano", "ano", False, False, None, []]]
```

```
index = indexing.SuffixArrayIndex(dataset1 = data_set_a,
                                  dataset2 = data_set_a,
                                  progress_report = 1,
                                  rec_comparator = rec_comp,
                                  index_sep_str = "",
                                  skip_missing = True,
                                  index_def = [index_def_1],
                                  suffix_method = "suffixonly",
                                  block_method = (6, 999), #define o
                                      tamanho de bloco
                                  padded = False)
# Build and compact index
#
index.build()
index.compact()

# Do record pair comparisons
[field_names_list, w_vec_dict] = index.run()
#---------------------------------------------------------------------
# Define weight vector (record pair) classifier
#
classifier = classification.FellegiSunter(lower_threshold = 4.0,
                                          upper_threshold = 4.6)
# Unsupervised training of classifier
#
class_w_vec_dict = w_vec_dict  # Use orignal weight vector dictionary
classifier.train(class_w_vec_dict, set(), set())
# Classify all weight vectors
#
[m_set, nm_set, pm_set] = classifier.classify(class_w_vec_dict)
#---------------------------------------------------------------------
# Define output file options
histo_str_list = output.GenerateHistogram(class_w_vec_dict, 1.0, "/home/
    osvaldomc/ajuda/prj/Deduplicados/histograma.csv")
```

```
for line in histo_str_list:
  print line
output.SaveMatchStatusFile(class_w_vec_dict, m_set, "/home/osvaldomc/ajuda/
    prj/Deduplicados/statusdopeso.csv")

output.SaveMatchDataSet(m_set, data_set_a, "match_id", "/home/osvaldomc/
    ajuda/prj/Deduplicados/dbcnsVIRG-ajustado-padronizadoFEBRL-match.csv")

# ============================================================================
# End of Febrl project module: "amostra02.py"
# ============================================================================
```

BIBLIOGRAFIA

[1] Alfred V. Aho, Ravi Sethi, e Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[2] Glenda Carla Moura Amaral. Aquaware: Um ambiente de suporte a qualidade de dados em data warehouse - dissertação de mestrado - ufrj. `http://teses2.ufrj.br/Teses/NCE_M/GlendaCarlaMouraAmaral.pdf`, 2003. [Acessado em 24-10-2013].

[3] Carlo Batini e Monica Scannapieco. *Data Quality: Concepts, Methodologies and Techniques (Data-Centric Systems and Applications)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[4] Vinayak Borkar, Kaustubh Deshmukh, e Sunita Sarawagi. Automatic segmentation of text into structured records. *SIGMOD Rec.*, 30(2):175–186, maio de 2001.

[5] Greg Caressi e Jesse Sullivan. Managing data growth and acess: The security and access speed of on-site data combined with the efficiency of a cloud solution. `http://www.frost.com/prod/servlet/cio/264652260`, 2012. [Online; acessado em 19-07-2012].

[6] Chih-Chung Chang e Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, maio de 2011.

[7] Peter Christen. A comparison of personal name matching: Techniques and practical issues. *Proceedings of the Sixth IEEE International Conference on Data Mining - Workshops*, ICDMW '06, páginas 290–294, Washington, DC, USA, 2006. IEEE Computer Society.

[8] Peter Christen. Febrl: A freely available record linkage system with a graphical user interface. `http://crpit.com/confpapers/CRPITV80Christen.pdf`, 2008. [Acessado em 19-07-2012].

[9] Peter Christen. Febrl - freely extensible biomedical record linkage - manual - release 0.4.2, 2011.

[10] Peter Christen. *Data Matching*. Springer, 1 edition, 2012.

[11] Peter Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Trans. on Knowl. and Data Eng.*, 24(9):1537–1555, setembro de 2012.

[12] Tim Churches, Peter Christen, Kim Lim, e Justin Xi Zhu. Preparation of name and address data for record linkage using hidden markov models. `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC140019/`, 2002. [Acessado em 10-10-2013].

[13] Conselho Nacional de Secretários de Saúde. Brasília CONASS. *SUS - Sistema Único de Saúde*. Sistema Único de Saúde - Coleção para Entender a Gestão do SUS, 291 p., 1 edition, 2011.

[14] Tr cs Stephen, M Blackburn, Robin Garner, Chris Hoffmann, Asjad M Khan, Kathryn S Mckinley, Rotem Bentzur, Daniel Feinberg, Daniel Frampton, Samuel Z Guyer, Martin Hirzel, Antony Hosking, Maria Jump, Han Lee, Thomas Vandrunen, Daniel Von Dincklage, Peter Christen, e Peter Christen. Towards parameter-free blocking for scalable record linkage. Relatório técnico, 2007.

[15] Ministério da Saúde Departamento de informática do SUS. Cadsus multiplataforma - documentação da crítica dos dados. `http://cnes.datasus.gov.br/Critica_dos_dados.doc`, 2008. [Acessado em 14-10-2013].

[16] Fred J. Damerau. A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7(3):171–176, março de 1964.

[17] Sanjoy Dasgupta. Performance guarantees for hierarchical clustering. *Proceedings of the 15th Annual Conference on Computational Learning Theory*, COLT '02, páginas 351–363, London, UK, UK, 2002. Springer-Verlag.

[18] Portal de cadastros nacionais DataSUS. Observacoes quanto as informacoes de totais no cadastro no municipio. http://cartaonet.datasus.gov.br/infoEstatistica.php, 2012. [Online; acessado em 19-07-2012].

[19] Guia de Direitos. . http://www.guiadedireitos.org/index.php?option=com_content\&view=article\&id=14\&Itemid=33, 2011. [Acessado em 19-07-2012].

[20] Secretaria de Gestão Estratégica e Participativa. *Cartão Nacional de Saúde - Normas e Procedimentos de Uso*. Ministério da Saúde, 1 edition, 2011.

[21] Uwe Draisbach e Felix Naumann. Dude: The duplicate detection toolkit. http://www.hpi.uni-potsdam.de/fileadmin/hpi/FG_Naumann/publications/2010/DuDe_-_The_Duplicate_Detection_Toolkit_cr.pdf, 2010. [Acessado em 19-07-2012].

[22] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, e Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Trans. on Knowl. and Data Eng.*, 19(1):1–16, janeiro de 2007.

[23] Larry P. English. *Improving Data Warehouse and Business Information Quality: Methods for Reducing Costs and Increasing Profits*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[24] G. David Forney. The viterbi algorithm. *Proceedings of the IEEE, Vol., 61, No. 3*, páginas 268–278, 1973.

[25] John Gantz e David Reinsel. Digital universe study: Extracting value from chaos. http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf, 2011. [Online; acessado em 19-07-2012].

[26] Hochbaum e Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

[27] D. Holmes e M.C. McCabe. Improving precision and recall for soundex retrieval. *Information Technology: Coding and Computing, 2002. Proceedings. International Conference on*, páginas 22–26, 2002.

[28] Pawel Jurczyk, James J. Lu, Li Xiong, Janet D. Cragan, e Adolfo Correa. Fril: A tool for comparative record linkage. `http://fril.sourceforge.net/amia2008jurczyk.pdf`, 2002. [Acessado em 19-07-2012].

[29] Karen Kukich. Techniques for automatically correcting words in text. *ACM Comput. Surv.*, 24(4):377–439, dezembro de 1992.

[30] A. Lait e B. Randell. An assessment of name matching algorithms. Technical report, Department of Computing Science, University of Newcastle upon Tyne, 1993.

[31] James MacQueen. Some methods for classification and analysis of multivariate observations. *5th Berkeley Symposium on Mathematical Statistics and Probability*, páginas 351–363. Springer, 1967.

[32] Adriana Zanella Martinhago. Customização em ambientes de qualidade de dados - dissertação de mestrado - ufpr. `http://dspace.c3sl.ufpr.br/dspace/bitstream/1884/4797/1/dissertacao_adriana.pdf`, 2006. [Acessado em 19-07-2012].

[33] Andrew McCallum, Kamal Nigam, e Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, páginas 169–178, New York, NY, USA, 2000. ACM.

[34] K. Hima Prasad, Tanveer A. Faruquie, Sachindra Joshi, Snigdha Chaturvedi, L. Venkata Subramaniam, e Mukesh Mohania. Data cleansing techniques for large enterprise datasets. *Proceedings of the 2011 Annual SRII Global Conference*, SRII '11, páginas 135–144, Washington, DC, USA, 2011. IEEE Computer Society.

[35] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *PROCEEDINGS OF THE IEEE*, páginas 257–286. IEEE, 1989.

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

[37] Yair Wand e Richard Y. Wang. Anchoring data quality dimensions in ontological foundations. *Commun. ACM*, 39(11):86–95, novembro de 1996.

[38] William E. Yancey. Evaluating string comparator performance for record linkage. Relatório técnico, Bureau of the Census, 2005.

[39] Willian Yancey. BigMatch: A Program for Extracting Probable Matches from a Large File. `www.census.gov/srd/papers/pdf/rrc2007-01.pdf`, 2007. [Acessado em 19-07-2012].

[40] Statistics New Zealand. Data integration manual. `http://www.stats.govt.nz/~/media/Statistics/about-us/policies-protocols-guidelines/data-integration-further-technical-info/DataIntegrationManual.pdf`, 2006. [Acessado em 19-07-2012].