UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA

GABRIEL GARCIA DE ALMEIDA

# Um estudo sobre Aprendizado de Máquina aplicado a *Data Matching*

Prof. Geraldo Bonorino Xexéo, D.Sc.
Orientador

Rio de Janeiro, Junho de 2016

# Um estudo sobre Aprendizado de Máquina aplicado a *Data Matching*

**Gabriel Garcia de Almeida**

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto de Matemática da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para obtenção do grau de Bacharel em Informática.

Apresentado por:

_____

Gabriel Garcia de Almeida

Aprovado por:

_____

Prof. Geraldo Bonorino Xexéo, D.Sc.

_____

Prof. Marcelo Arêas Rodrigues da Silva, M.Sc.

_____

Prof. Maximiliano Martins de Faria, M.Sc.

RIO DE JANEIRO, RJ - BRASIL

Junho de 2016

# Agradecimentos

Primeiramente, devo agradecer a meus companheiros de trabalho, que merecem grande destaque e contribuíram ativamente para este projeto, tanto na idealização, quanto nas inúmeras análises e especialmente durante o processo de geração dos dados utilizados aqui. Em seguida, gostaria de agradecer a todos que me apoiaram nesta jornada da graduação, dentre familiares e amigos, além de meu orientador, que meu deu toda liberdade necessária para a efetivação desta monografia.

RESUMO

Um estudo sobre Aprendizado de Máquina aplicado a *Data Matching*

Gabriel Garcia de Almeida

Junho/2016

Orientador: Geraldo Bonorino Xexéo, D.Sc.

Este trabalho tem por objetivo desenvolver um *framework* de qualidade de dados focado em *data matching*, estudando métodos de aprendizado supervisionado para predizer a probabilidade de igualdade entre pares de registros, por meio do cálculo de diversas medidas de similaridade entre os atributos.

ABSTRACT

Um estudo sobre Aprendizado de Máquina aplicado a *Data Matching*

Gabriel Garcia de Almeida

Junho/2016

Advisor: Geraldo Bonorino Xexéo, D.Sc.

*The objective of this work is the development of a framework of data quality focused on data matching, studying methods of supervised learning to predit the probability of equality between pairs of records, using similarity metrics between the atributtes.*

# Lista de Figuras

# Lista de Tabelas

# Lista de Símbolos

| | |
|---|---|
| $\theta$ | vetor coluna de parâmetros |
| $\theta_i$ | i-ésimo elemento do vetor de parâmetros |
| $h_\theta(\mathbf{x})$ | função de hipótese com parâmetro $\theta$ |
| $m$ | número de observações disponíveis |
| $n$ | número de *features* disponíveis |
| $g(z)$ | função logística |
| $\mathbf{y}$ | vetor coluna de respostas, contendo $m$ respostas |
| $\mathbf{y}_i$ | resposta da i-ésima observação |
| $\mathbf{X}$ | matriz do conjunto de treinamento, contendo $m$ observações e $n$ *features* |
| $\mathbf{X}_i$ | vetor de *features* da i-ési ma observação do conjunto de treinamento |
| $\mathbf{X}_{i,j}$ | valor da j-ésima *feature* da i-ésima observação do conjunto de treinamento |
| $\alpha$ | coeficiente de aprendizado |
| $\lambda$ | coeficiente de regularização |

# Sumário

# Capítulo 1

# Introdução

Desde o surgimento dos sistemas computacionais, a quantidade de dados gerados e armazenados vem crescendo enormemente [19], especialmente com o advento da internet e o aumento do acesso às mídias de comunicação. Tais dados podem ter as mais diversas formas e os mais variados objetivos. Exemplos de uso vão desde a simples manutenção de cadastros como informações de *login* dos usuários de um sistema até características de produtos e quantidade em estoque de uma loja virtual.

Dados sem erros podem não ser apenas interessante do ponto de vista teórico, como podem ser críticos e evitar prejuízos. Por exemplo, se é desejado enviar uma cobrança, o endereço dos clientes devem estar corretos ou a correspondência pode não chegar. Certos tipos de erros podem ser muito difíceis de serem prevenidos, especialmente no que diz respeito a sua dimensão temporal, como uma mudança de endereço ou de nome em um casamento. Outra situação seria o caso em que o dado não está exatamente preciso ou está incompleto, como um nome que pode ser escrito de diversas formas (ex: Luiz e Luís), nomes do meio omitidos, erros de digitação e endereços com seu CEP errado ou neglicenciado. Tal situação envolvendo a acurácia do dado pode ser mais tratável que o caso anterior, mas também pode ser difícil de ser evitada, dependendo do contexto do sistema de informação, como em casos de validações falhas na inserção do dado ou até usuários despreparados para o uso do sistema.

Uma forma de se realizar a correção dos dados incorretos é fazendo o uso de uma base de referência, a qual acreditamos estar, pelo menos, mais correta que a base analisada. No caso do endereçamento de cobranças, a base DNE[1] dos Correios brasileiro pode ser usada como referência, já que esta contém todas as localidades e logradouros abrangidos pelo sistema de CEPs. Uma abordagem intuitiva seria fazer com que cada endereço da base avaliada em questão seja comparado com o endereço equivalente da base de referência, usando-se o CEP como referencial.

Nota-se ambos registros não são idênticos, podem haver pelo menos dois cenários: ou existe uma pequena variação, porém de fato os dois endereços representam o mesmo local, ou o CEP realmente está incorreto. Tendo isso em vista, apenas verificar binariamente se os endereços são iguais pode não ser interessante, porém ter uma forma de indicar o quão provável ambos os endereços se refiram ao mesmo local seria mais interessante, já que poderia-se definir um limite aceitável de erro para que os dados da base de dados sejam atualizados, enquanto dados considerados inconsistentes seriam encaminhados para uma análise mais profunda.

Fazer tal tarefa manualmente para uma base de dados de grandes ordens de grandeza é inviável, logo um método computacional automatizado e eficiente se faz necessário. A área de estudo de tais métodos comparativos é chamada de ***data matching*** ou ***record linkage*** e é o tema desta monografia.

## 1.1 Objetivo

Este trabalho tem por objetivo o desenvolvimento de um método de *data matching* que se utiliza de técnicas de **aprendizado de máquina**, mais especificamente, de **aprendizado supervisionado**, para se comparar automaticamente registros de diferentes bancos de dados, mensurando a similaridade entre os registros. Tal abordagem permite que as especificidades das bases e os critérios de classificação desejados sejam inferidos através de exemplos providos pelo analista, sem a necessidade de se definir diversos parâmetros abstratos manualmente para a classificação.

[1] `http://www.correios.com.br/para-voce/correios-de-a-a-z/dne`

Para se testar a eficácia do método proposto, uma *framework* foi desenvolvida em Java e testada com uma base real, calculando-se medidas de desempenho através da metodologia da **validação cruzada**, como usual no contexto de aprendizado de máquina[6].

## 1.2 Abordagem

Este projeto abordará o problema de *data matching* como proposto teoricamente por Fellegi. Seguem abaixo algumas das definições simplificadas vindas de seu trabalho e bem elucidadas por Bilenko [16] apud [5]:

- Pode-se representar cada par de registros comparados com um vetor de medidas que devem quantificar a similaridade entre eles nos diferentes âmbitos.
- O problema de identificar registros equivalentes pode ser encarado como um problema de classificar um dado vetor de medidas em três classes: equivalentes, não equivalentes e possivelmente equivalentes.
- Um sistema pode realizar a classificação de um par de registros calculando a probabilidade de seu vetor de medidas representar ou não um caso de equivalência. Dois limiares de classificação devem ser escolhidos para se categorizar a observação nas classes propostas, baseados nos níveis desejados de precisão e abrangência.

O trabalho de Fellegi é interessante devido a sua afinidade com a metodologia de aprendizado de máquina. De fato, o algoritmo de aprendizado escolhido, a **regressão logística**, tenta calcular a probabilidade de uma observação pertencer a classe de interesse, a partir de um vetor de *features*, como usualmente seria chamado o vetor de medidas no contexto de aprendizado de máquina. A regressão logística foi escolhida pelo fato de ser um classificador binário simples, que assume apenas a hipótese de que as classes são linearmente separáveis, ou seja, um plano (ou hiperplano) deve ser capaz de realizar a identificação da classe de interesse, além de

possuir uma forte interpretação estatística, como será explicado no Capítulo 2 deste trabalho. [23]

Foi escolhido representar a similaridade entre os pares de registros usando-se várias métricas diferentes, de forma que cada atributo relevante de um dado registro é comparado com seu atributo equivalente no registro candidato por meio de vários algoritmos conhecidos de similaridade de *string*, coletando-se inúmeras *features* para o algoritmo de aprendizado. Assume-se que as equivalências entre os esquemas das bases de dados foram mapeadas a priori da realização do processo de *data matching* para que a comparação fosse possível. A intuição por trás desta abordagem é prover ao algoritmo de aprendizado diferentes perspectivas sobre um atributo, de forma que ele seja capaz de ponderar a afinidade de cada métrica de similaridade em avaliar a equivalência dos diferentes atributos, além de suas importâncias relativas, com relação aos demais atributos.

## 1.3 Trabalhos similares

O estudo de *data matching* não é recente, podendo-se considerar que um dos primeiros trabalhos foi o desenvolvimento da chamada **distância de edição**, idealizada por Levenshtein em 1965 [21], agora também conhecida como **Distância de Levenshtein**. Esta compara sequências de caracteres e indica quantas operações devem ser realizadas para se transformar um elemento em outro. É possível indicar se dois registros são equivalentes apenas definindo-se um limite de tolerância para a quantidade de operações permitidas e comparando-os como se cada um fosse uma única sequência de caracteres. Outras medidas de similaridades para sequências de caracteres também foram desenvolvidas, como as medidas de Jaro, Jaro-Winkler e Monge-Elkan [11] [15], que serão mais detalhas a frente.

A utilização de aprendizado de máquina no contexto de *data matching* já foi feita antes e uma abordagem similar a deste trabalho vem da pesquisa de Bilenko et al.[5], que também utiliza diferentes combinações entre medidas de similaridade e métodos de agregação para os diferentes atributos dos registros. Dentre as técnicas,

é utilizado o algoritmo de aprendizado supervisionado SVM. [2]

Outro trabalho interessante é o de Camacho[8], que utiliza o algoritmo genético como uma forma de escolher o melhor método de similaridade e os diferentes pesos para se agregar a informação dos diversos campos em uma dada base de dados. Utiliza-se um conjunto de treino para calcular a qualidade de cada solução durante a execução do algoritmo, logo pode ser considerada uma abordagem supervisionada.

Ferramentas de código aberto com propostas similares à deste projeto estão disponíveis na internet, como o Duke[17] e o Dedupe[18]. O Duke é uma ferramenta feita na linguagem Java que permite ao usuário configurar bases de dados para se realizar o processo de desduplicação ou *data matching*, recebendo como parâmetros o algoritmo de similaridade a ser usado em cada atributo relevante da base, além das probabilidades usadas para calcular-se a chance final dos registros comparados serem iguais. Para facilitar a configuração, o autor disponibilizou um utilitário que auxilia na escolha dos parâmetros, usando algoritmo genético e aprendizado supervisionado ativo, para evoluir a configuração iterativamente, enquanto tenta questionar minimamente o usuário. A ferramenta Dedupe é desenvolvida em Python e tem um escopo parecido com a anterior, porém usa por padrão o algoritmo de *affine gap* como medida de similaridade para *strings* e a regressão logística regularizada como método de treino, também usando aprendizado ativo para obter a opinião do usuário a respeito de um dado par de registros.

## 1.4 Organização

Este projeto está organizado em quatro capítulos. O primeiro é esta introdução, na qual se apresenta o problema de *data matching*, sua motivação e princípios teóricos, além de alguns trabalhos existentes na área. O segundo capítulo elucidará mais os conceitos envolvidos, tanto das áreas de *data matching* como de aprendizado de máquina, usados durante a implementação do projeto. O terceiro capítulo explica os detalhes de implementação e apresenta os resultados obtidos na base de teste para

[2] Sigla em inglês para *Support Vector Machine*

diversos experimentos. Por fim, o quarto capítulo apresenta as conclusões e futuras linhas de continuação para este trabalho.

# Capítulo 2

# Conceitos e metodologia

## 2.1 Aprendizado supervisionado

O **aprendizado supervisionado** pode ser entendido intuitivamente como uma forma de aprendizado baseado em exemplos, na qual o algoritmo recebe diversos casos do problema que ele deve resolver e suas respostas esperadas, informação que será usada na tentativa de generalizar os casos apresentados a ele futuramente. [6][23]

Formalmente, o algoritmo de aprendizado supervisionado receberá o chamado **conjunto de treino**, que geralmente consiste em uma matriz, onde cada linha representa uma observação do problema e cada coluna é uma ***feature*** ou característica do problema, além de um vetor contendo as respostas esperadas pelo algoritmo em cada um dos casos. Quando as respostas esperadas são valores contínuos, o problema de aprendizado é chamado de **regressão** e quando são valores discretos, ele é um problema de **classificação**. Esta monografia utiliza classificação binária, ou seja, apenas duas respostas possíveis são aceitas. O processo de classificação binária tipicamente pode ser interpretado de duas formas: probabilisticamente, em que se prediz a chance de uma dada observação ser da classe de interesse, ou geometricamente, na qual um plano (ou hiperplano, dependendo do tamanho da dimensão) é usado para separar as classes. [23][6]

## 2.2 *Features*

Para que os algoritmos de aprendizado de máquina funcionem bem, precisa-se definir bons descritores do problema que se deseja resolver, tipicamente chamados de *features*. A escolha do que usar como *feature* deve ser a etapa mais essencial do aprendizado de máquina, pois é isso que irá representar o que o algoritmo terá de informação relevante para resolver o problema.

Em estatística, esse conceito é conhecido como **variáveis observáveis**, definidas como variáveis aleatórias que literalmente podem ser mensuradas. Um método de aprendizado de máquina pode ser encarado como um **modelo estatístico**, na qual uma **função de distribuição de probabilidade** [1] é assumida e se quer usar as finitas observações de um evento (treino) para estimar os parâmetros não conhecidos desta função, que posteriormente será usada para se predizer os resultados das futuras observações, dado as devidas variáveis observáveis. [13][24]

Operacionalmente, as *features* são comumente simples valores numéricos contínuos que ajudam a descrever o problema que se deseja atacar. Por exemplo, se queremos prever o preço de um imóvel numa localidade específica, podemos usar como *features* o tamanho do imóvel, quantidade de banheiros e quartos, número de vagas na garagem etc. Idealmente pode-se pensar que um especialista deve ser capaz de julgar a situação tratada apenas sabendo os valores das *features*. Mesmo informações não-numéricas podem ser codificada num vetor de *features*, como um valor binário, que é representado no conjunto de treino tipicamente como 0 e 1 (ou -1 e 1). Variáveis categóricas podem seguir uma lógica similar, onde cada categoria possível é tratada como uma variável binária individual.

## 2.3 Medidas de similaridade

Quando se refere em medidas ou **métricas de similaridade**, fala-se de funções que comparam duas sequências de caracteres [2] e retornam um valor numérico, que

[1] Ou uma função de densidade de probabilidade

[2] Também chamadas de *string*, no contexto de programação

indica o quão similar são elas. Os termos *métrica* ou *distância* podem ser confusos neste contexto, pois matematicamente ele implica que uma dada função $d(x, y)$ deve seguir as restrições[29]:

$$d(x, y) > 0$$

$$d(x, y) = 0 \iff x = y$$

$$d(x, y) = d(y, x)$$

$$d(x, z) \leq d(x, y) + d(y, z)$$

Onde $x$ e $y$ são o objeto de estudo, neste caso, *strings*. Nem todas as medidas de similaridade apresentadas aqui seguem estas restrições, em especial a última, chamada de **inequação do triângulo**. Como é matematicamente incorreto chamar todas as funções apresentadas aqui de métricas de similaridade, opta-se em usar o termo medida para se referir à essas funções comparativas. Alguns exemplos de medidas de similaridade de *string* são indicadas abaixo, todas baseadas apenas em caracteres individuais e sugeridas nas fontes bibliográficas. [5, 15, 11]

### 2.3.1 Distâncias de Edição

Baseada no trabalho de Levenshtein em [21], propõe-se que a distância entre duas *strings* seja dada pelo custo da menor sequência de operações de edição entre ambas *strings*: inserção, substituição e exclusão de caracteres. Cada operação pode ter um custo individual, porém se todos os custos são iguais a um, então a distância de edição é chamada de **distância de Levenshtein**. Por exemplo, comparando *Gabriel* e *Gabirel* usando a distância de Levenshtein, obtem-se distância 2: operação de substituição de $r$ por $i$ e outra substituição de $i$ com $r$, mais exemplos se encontram na Tabela 2.3.1. Distâncias de edição funcionam bem para se identificar erros tipográficos porém são tipicamente inefetivas para outros tipos de inconsistências. [15]

O caso genérico é chamado de **distância de Neddleman-Wunsch**, na qual pode-se definir um peso específico para toda combinação de caracteres por operação, por exemplo, usando-se a distância entre as teclas de um teclado padrão ABNT-2.

| ***String* 1** | ***String* 2** | **Distância de edição** | **Operações** |
|---|---|---|---|
| Gabriel | Gabriela | 1 | Inserção do “A” |
| Gabriel | Daniel | 3 | Substituição do “G” por “D”<br>Substituição do “b” por “n”<br>Exclusão do “r” |

Tabela 2.1: Exemplos da Distância de Levenshtein.

Uma distância de edição pode ser calculada eficientemente usando-se **programação dinâmica**, obtendo um tempo de execução proporcional ao produto dos tamanhos de ambas as *strings*.[5]

### 2.3.2 Jaro

Jaro em [20] propôs uma medida de similaridade idealizada para ser usada com *strings* pequenas, como primeiro e últimos nomes, e que é conhecida por ter bons resultados no contexto de *data matching* [5][11][15]. A medida se baseia no número e na ordem dos caracteres em comum entre as duas *strings* para indicar o grau de similaridade entre ambas. Winkler modificou a medida de Jaro para ponderar mais fortemente o prefixo da *string*[30], tal medida é conhecida pelo nome Jaro-Winkler. Alguns exemplos das medidas se encontram na Tabela 2.3.2.

| ***String* 1** | ***String* 2** | **Jaro** | **Jaro-Winkler** |
|---|---|---|---|
| Gabriel | Gabriela | 0,958 | 0,975 |
| Gabriel | Daniel | 0,746 | 0,746 |
| Gabriel Garcia | Gabriel de Almeida | 0,798 | 0,8793 |

Tabela 2.2: Exemplos das medidas de Jaro e Jaro-Winkler.

### 2.3.3 Smith-Waterman e variantes

O método de Smith-Waterman é uma extensão da distância da edição, que permite haver pesos diferentes para caracteres incorretos no início ou no fim das *strings*, geralmente menores que os demais. Tal característica faz desse método uma boa escolha para se realizar alinhamentos locais entre as *strings*, o que permitiria uma

melhor capacidade de se detectar *substrings* similares. [15] Por exemplo, os casos *Prof. John R. Smith, University of Calgary* e *John R. Smith, Prof.*, também vindos de [15], podem ser mais facilmente considerados equivalentes se o título de professor e o nome da universidade fossem removidos. O método de Smith-Waterman, basicamente vai ignorar prefixos e sufixos diferentes entre ambas *strings*.

Uma variante desse modelo, chamada em inglês de *affine gap*, pode assumir mais duas operações: a abertura e continuação de *gaps* (lacunas). Fazendo com que o custo de continuação de *gaps* seja menor do que o custo de abertura, este novo modelo se torna interessante para se detectar casos de abreviação ou truncamento, como em *John R. Smith* e *Jonathan Richard Smith*. [5, 15]

Alvaro Monge e Charles Elkan fizeram diversos experimentos com a variante *affine gap* do algoritmo de Smith-Waterman para se realizar detecções de duplicatas, com os pesos de cada operação bem ajustados para tal, como descrito em [22]. Esta variante, usando os parâmetros descritos por eles, será referenciada posteriormente como Monge-Elkan. A Tabela 2.3.3 possui exemplos desta medida.

| ***String* 1** | ***String* 1** | **Monge-Elkan** |
|---|---|---|
| Gabriel | Sr. Gabriel Almeida | 1,0 |
| Gabriel Almeida | Gabriel Garcia de Almeida | 0,813 |

Tabela 2.3: Exemplos da medida de Monge-Elkan.

## 2.4 Pré-processamento

O pré-processamento é uma etapa que tipicamente precede o processo de extração de *features*, encarregada de preparar os dados antes de serem utilizados, afim de remover, ao menos parcialmente, alguma incompatibilidade ou ruído. O pré-processamento, em geral, pode ser compreendido em termos de uma transformação em um processo de ETL[3], sendo necessário devido as formas heterogêneas nas quais os dados são armazenados nas diversas bases, que podem ter diferentes formatos e estruturas. Um exemplo de problema de formato seria quando uma base assume

[3]Sigla para Extract, Transform e Load

por convenção que todos os nomes devem ser em letras minúsculas, enquanto outra permite maiúsculas e minúsculas. Já o problema estrutural pode ser o caso em que o logradouro, tipo de logradouro e bairro de um cliente estão em um único campo, enquanto em outra base, eles apareceriam discriminados. Resolver tais problemas é interessante até do ponto de vista da eficácia dos algoritmos de similaridade, que serão usados para predizer a equivalência entre os diferentes registros, já que estes não são necessariamente idealizados considerando estes tipos de problemas.

Exemplos de passos de pré-processamento são:

- Conversão para letras maiúsculas ou minúsculas
- Remoção de espaçamentos extras
- Remoção de caracteres não alfanuméricos
- Remoção de acentos
- Discriminação dos diferentes campos, como no caso de endereços
- Padronização do formato de datas
- Remoção de palavras pouco relevantes, como preposições[4]
- Expansão de siglas e abreviações conhecidas
- Escrita de números por extenso
- Preenchimento de valores inválidos

Os passos de pré-processamento devem ser escolhidos de acordo com as necessidades de cada situação que se quer resolver. Por exemplo, no caso de se querer tratar um nome de pessoa física, pode não tem muito sentido conter um número nele, mas se fosse o caso de um nome de empresa, já poderia haver. [15]

[4]Essá técnica também é conhecida como remoção de *stop-words*

## 2.5 Regressão logística

Algoritmos de aprendizado supervisionado são bem conhecidos e estudados a bastante tempo, geralmente interpretados como problemas de otimização, em que se busca minimizar o erro de uma função $h_\theta(\mathbf{x})$ escolhida como hipótese para solução do problema desejado, usando o conjunto de treino como subsídio para tal. A função de hipótese deve receber um vetor $\mathbf{x}$ de medições (*features*) como entrada e retornar o valor da resposta esperada do problema. Ela será ajustada para minimizar o erro por meio da escolha de um vetor de parâmetros $\theta$.[6, 24, 23]

Pode-se interpretar o problema de classificação probabilisticamente, assumindo-se que uma variável aleatória binária $Y$ toma o valor 1 se a observação em questão faz parte da categoria de interesse ou 0 caso contrário. Também se assume que a **função de probabilidade conjunta** de Y pode ser aproximada pela função de hipótese e que esta distribuição de probabilidade depende das variáveis observáveis $X_1, X_2, ...$, resumidas no vetor $\mathbf{x}$, de forma que:

$$h_\theta(\mathbf{x}) = P(Y = 1|X_1 = x_1, X_2 = x_2, ...; \theta) = p(y|\mathbf{x}; \theta)$$

$$1 - h_\theta(\mathbf{x}) = P(Y = 0|X_1 = x_1, X_2 = x_2, ...; \theta) = 1 - p(y|\mathbf{x}; \theta)$$

Ou de forma compacta, como sugerido em [24]:

$$p(y|\mathbf{x}; \theta) = (h_\theta(\mathbf{x}))^y(1 - h_\theta(\mathbf{x}))^{1-y}$$

Um dos modelos de aprendizado de máquina mais simples para o problema de classificação é a chamada **regressão logística**, que tenta prever a probabilidade de uma dada observação ser pertinente a uma categoria de interesse. A observação então é classificada binariamente, baseada num limiar de decisão, tipicamente de 50%. [24] A função de hipótese $h_\theta(\mathbf{x})$ da regressão logística para $n$ *features* e $n + 1$ parâmetros no vetor $\theta$, é dada por :

$$h_\theta(\mathbf{x}) = g(\theta_0 + \mathbf{x}_1\theta_1 + \mathbf{x}_2\theta_2 + ... + \mathbf{x}_n\theta_n)$$

Ou, de forma vetorial:

$$h_\theta(\mathbf{x}) = g(\mathbf{x}^T\theta) = \frac{1}{1 + e^{-\mathbf{x}^T\theta}}$$

Onde $\mathbf{x}_i$ é o valor da i-ésima *feature* de uma determinada observação, $\theta_i$ é o i-ésimo parâmetro de $\theta$ e $g(z)$ é a função sigmoide[5], definida como $g(z) = \frac{1}{1+e^{-z}}$. Observa-se que para simplificar a notação, assume-se que a 0-ésima *feature* de $\mathbf{x}$ é invariante e igual a 1. Uma propriedade interessante da função sigmoide é tender a 0 quando $z \to -\infty$ e tender a 1 quando $z \to +\infty$, além de ser 0.5 quando $x = 0$, o que a torna uma boa forma de garantir que os valores retornados pela regressão logística possam ser interpretados como probabilidades, variando os resultados entre 0 e 1.

Para calcular uma estimativa dos parâmetros $\theta$ de $h_\theta(\mathbf{x})$, podemos utilizar um estimador de máxima verossimilhança como função objetivo. Considerando que temos $m$ observações no conjunto de treino $\mathbf{X}$ com suas respostas $\mathbf{y}$ e que estas observação foram feitas independes entre si, temos a função de verossimilhança:

$$\begin{aligned}\mathcal{L}(\theta) &= \prod_{i=1}^{m} P(Y = \mathbf{y}_i | X_1 = \mathbf{X}_{i,1}, X_2 = \mathbf{X}_{i,2}, ...; \theta) \\ &= \prod_{i=1}^{m} p(\mathbf{y}_i | \mathbf{X}_i; \theta) = \prod_{i=1}^{m} h_\theta(\mathbf{X}_i)^{\mathbf{y}_i}(1 - h_\theta(\mathbf{X}_i))^{1-\mathbf{y}_i}\end{aligned}$$

Onde denotou-se $\mathbf{X}_{i,1}$ como sendo o valor da primeira *feature* da i-ésima observação do conjunto de treino, $\mathbf{X}_i$ como sendo o vetor de *features* da i-ésima observação e $\mathbf{y}_i$ como sendo a resposta esperada da i-ésima observação. Como usual, podemos aplicar a função logaritmo na equação para simplificar passos futuros:

$$\log(\mathcal{L}(\theta)) = l(\theta) = \sum_{i=0}^{m} \mathbf{y}_i \log(h_\theta(\mathbf{X}_i)) + (1 - \mathbf{y}_i) \log(1 - h_\theta(\mathbf{X}_i))$$

Para maximizar a função de verossimilhança para este caso de múltiplos parâmetros, podemos usar um método iterativo que será referenciado como **método**

[5]Também chamada de função logística.

**dos gradientes**, conhecido em inglês com *gradient descent* ou *steepest descent* [28], que intuitivamente otimiza uma dada função fazendo pequenos deslocamentos destes parâmetros na direção do vetor gradiente deles[6]. Para calcular o vetor gradiente é necessário saber as derivadas parciais em função de cada um dos parâmetros $\theta_j$, para um $j = 0, 1, ..., n$:

$$\frac{\partial l(\theta)}{\partial \theta_j} = \sum_{i=0}^{m} \mathbf{y}_i \frac{1}{h_\theta(\mathbf{X}_i)} \frac{\partial h_\theta(\mathbf{X}_i)}{\partial \theta_j} + (1 - \mathbf{y}_i) \frac{1}{1 - h_\theta(\mathbf{X}_i)} \frac{\partial (1 - h_\theta(\mathbf{X}_i))}{\partial \theta_j}$$

$$= \sum_{i=0}^{m} \Big( \mathbf{y}_i \frac{1}{h_\theta(\mathbf{X}_i)} - (1 - \mathbf{y}_i) \frac{1}{1 - h_\theta(\mathbf{X}_i)} \Big) \frac{\partial (h_\theta(\mathbf{X}_i))}{\partial \theta_j}$$

Lembrando que $g(z)$ é a função sigmoide, obtém-se pela regra da cadeia que a derivada parcial de $h_\theta(\mathbf{x})$ em função de $\theta_j$ é :

$$\frac{\partial (h_\theta(\mathbf{x}))}{\partial \theta_j} = \frac{\partial g(\mathbf{x}^T \theta)}{\partial \theta_j} = g'(\mathbf{x}^T \theta) \frac{\partial (\mathbf{x}_1 \theta_1 + \mathbf{x}_2 \theta_2 + ...)}{\partial \theta_j} = g'(\mathbf{x}^T \theta) \mathbf{x}_j$$

E calculando a derivada de $g(z) = \frac{1}{1+e^{-z}}$, obtem-se:

$$g'(z) = \frac{1}{(1 + e^{-z})^2} (e^{-z}) = \frac{1}{(1 + e^{-z})} \Big( 1 - \frac{1}{1 + e^{-z}} \Big) = g(z)(1 - g(z))$$

Por fim, substituindo na equação da função de verossimilhança:

$$\frac{\partial l(\theta)}{\partial \theta_j} = \sum_{i=0}^{m} \Big( \mathbf{y}_i \frac{1}{h_\theta(\mathbf{X}_i)} - (1 - \mathbf{y}_i) \frac{1}{1 - h_\theta(\mathbf{X}_i)} \Big) h_\theta(\mathbf{X}_i)(1 - h_\theta(\mathbf{X}_i)) \mathbf{X}_{i,j}$$

$$= \sum_{i=0}^{m} (\mathbf{y}_i (1 - h_\theta(\mathbf{X}_i)) - (1 - \mathbf{y}_i) h_\theta(\mathbf{X}_i)) \mathbf{X}_{i,j}$$

$$= \sum_{i=0}^{m} (\mathbf{y}_i - h_\theta(\mathbf{X}_i) \mathbf{y}_i - h_\theta(\mathbf{X}_i) + h_\theta(\mathbf{X}_i) \mathbf{y}_i) \mathbf{X}_{i,j}$$

$$\frac{\partial (h_\theta(\mathbf{X}_i))}{\partial \theta_j} = \sum_{i=0}^{m} (\mathbf{y}_i - h_\theta(\mathbf{X}_i)) \mathbf{X}_{i,j}$$

[6] Ou contra o vetor gradiente, caso se deseje minimizar a função.

Com esta derivada parcial, podemos descrever a iteração do método do gradiente para cada parâmetro proveniente do vetor $\theta$ como :

$$\theta_j := \theta_j + \alpha(\sum_{i=0}^{m}(\mathbf{y}_i - h_\theta(\mathbf{X}_i))\mathbf{X}_{i,j})$$

As condições de parada deste algoritmo podem ser várias, parando quando a primeira condição ocorrer. Exemplos de condições de parada são: limitar o número máximo de iterações, definir um erro mínimo aceitável para o modelo e definir uma variação mínima para a melhora do erro entre as iterações.

O método dos gradientes irá convergir para um ótimo local desde que o passo em direção ao ótimo local seja suficientemente pequeno para que não se extrapole demais o ponto, logo o coeficiente $\alpha$, chamado de **coeficientes de aprendizado** serve para garantir tal critério, quando $0 < \alpha \leq 1$. Um problema do método do gradiente é a sua demora na convergência [28], que pode ser contornada usando alguma heurística que regule o coeficiente de aprendizado, afim de que passos maiores sejam dados enquanto se está indo na direção correta, porém passos menores devem ser realizados conforme a taxa de erro não melhore. Uma heurística possível é o ***bold driver***[14], que vai aumentando gradativamente o coeficiente de aprendizado conforme a taxa de erro melhora, porém diminui o coeficiente bruscamente quando a taxa de erro piora.

Por fim, como Bishop explica em seu livro[6], algoritmos de aprendizado de máquina podem sofrer com o chamado ***over-fitting***, condição em que um modelo se adapta demais ao conjunto de treino, perdendo sua capacidade de generalização, quando aplicado fora do conjunto de treino. Um exemplo disto seria se uma das medidas de similaridade de um dos campos fosse considerada muito relevante durante o treinamento, ficando com seu peso muito forte e praticamente ignorando as demais *features*, que poderiam ser decisivas em situações não vistas. Uma técnica geralmente usada para vencer tal dificuldade é o uso da **regularização**, que consiste na adição de um termo de penalizador na função objetivo, a verossimilhança neste caso, que irá evitar que os coeficientes se tornem muito grandes. Tal termo pode ser simplesmente a norma euclidiana ao quadrado do vetor $\theta$, como sugerido em [6], tornando a função objetivo otimizada no método dos gradientes em:

$$l(\theta) - \lambda\|\theta\|_2^2$$

Onde $\lambda$ é um coeficiente extra, chamado de coeficiente de regularização, que irá mensurar a importância dada a regularização em relação a função objetivo, e $\|\theta\|_2$ é a norma euclidiana do vetor $\theta$. Novamente, para se aplicar o método dos gradientes, é necessário saber a derivada parcial da função objetivo, mas devido a linearidade da derivação, basta derivar seu novo termo e reutilizar o resultado anterior:

$$\frac{\partial\|\theta\|_2^2}{\partial\theta_j} = \frac{\partial\sum_{i=0}^{n}\theta_i^2}{\partial\theta_j} = 2\theta_j$$

Como $\lambda$ é ajustado manualmente, pode-se descartar o fator 2 e é obtida finalmente a nova regra de atualização de $\theta$, usada durante as iterações:

$$\theta_j := \theta_j + \alpha(\sum_{i=0}^{m}(\mathbf{y}_i - h_\theta(\mathbf{X}_i))\mathbf{X}_{i,j}) - \lambda\theta_i$$

## 2.6 *Blocking*

Antes de se realizar as comparações do processo de *data matching*, é necessário localizar os possíveis pares de candidatos a equivalência. Idealmente esta etapa pode ser praticamente ignorada caso exista identificadores comuns nas bases de dados, fazendo com que esta comparação seja realizada de forma ótima, de um-para-um. Tal abordagem teria um custo de tempo computacional assintótico da ordem de, pelo menos, $O(min(m,n))$, onde $n$ e $m$ são as quantidades de registros em cada base de dados. O procedimento seria basicamente varrer a menor base e usar uma estrutura de dados eficiente para se obter o registro equivalente da outra base, como uma **tabela de *hash***, que realiza consultas em um tempo médio da ordem de $O(1)$, em algumas condições razoáveis. [12]

Caso identificadores comuns não estejam disponíveis, uma comparação exaustiva seria a abordagem mais simples para se localizar possíveis pares de registros equivalentes entre as bases, o que teria um custo de tempo computacional da ordem

de $O(nm)$, sendo impraticável quando as bases de dados assumem altas ordens de grandeza. Logo é desejável que exista uma forma de, dado um registro em uma das bases, se seleciona uma lista de bons candidatos na outra base. Tal procedimento deve ser eficiente e abrangente, funcionando como uma triagem, afim de eliminar comparações entre pares de registros muito discrepantes, porém mantendo bons candidatos para uma inspeção mais profunda. Como descrito em [11, 15], estes métodos são geralmente conhecidos como ***blocking*** ou ***pruning***[7] e tipicamente usam campos altamente discriminativos, como nomes, para se gerar um índice comparativo. Este método funciona de forma similar a uma tabela de *hash* que retorna um conjunto de registro similares, onde a função de *hashing* seria uma função de agregação arbitrária ou até uma codificação especial do atributo escolhido, de forma a juntar, num mesmo conjunto, os diversos registros parecidos. **Codificações fonéticas**, como o ***Metaphone***, são fortes candidatos nesta etapa, pois generalizam um dado texto para uma representação de seus fonemas mais relevantes. [15]

## 2.7 Avaliação do desempenho

Para se garantir que um algoritmo de aprendizado obtenha bons resultados, mesmo em situações não previstas durante o treino, é necessário definir uma metodologia para avalia-los. O método mais comum para modelos estatísticos e algoritmos de aprendizado supervisionado é a chamada **validação cruzada**[1], na qual uma parte do conjunto de dados é deixada de fora do treinamento e usada para avaliar o desempenho do algoritmo. Desta forma, tenta-se medir a performance e a capacidade de generalização do modelo gerado pelo algoritmo, para então decidir se é viável aceita-lo como uma solução do problema ou se será necessário criar um modelo diferente para tal.

A forma mais simples de se aplicar a validação cruzada é separar aleatoriamente o conjunto de dados em duas porções, de por exemplo 30% e 70% do total de dados, na qual a maior porção será o conjunto de treino e a menor será o **conjunto de teste**. Pode-se então realizar o procedimento de aprendizado adequado, como a

[7] poda, em inglês

regressão logística, e usar o modelo resultante para classificar o conjunto de teste e então calcular métricas de avaliação.

No caso do problema de classificação binária, podemos definir as seguintes métricas, também amplamente utilizadas no contexto de recuperação da informação[26]:

$$\text{Acurácia} = \frac{\text{Verdadeiros Positivos} + \text{Verdadeiros Negativos}}{\text{Tamanho total}}$$

$$\text{Precisão} = \frac{\text{Verdadeiros Positivos}}{\text{Verdadeiros Positivos} + \text{Falsos Positivos}}$$

$$\text{Abrangência} = \frac{\text{Verdadeiros Positivos}}{\text{Verdadeiros Positivos} + \text{Falsos Negativos}}$$

Onde o valor de **verdadeiros positivos** indica o número de casos em que o classificador indicou corretamente que a observação é pertencente à categoria de interesse e o valor de **falsos positivos** indica o número de casos erroneamente ditos desta categoria. Verdadeiros negativos e falsos negativos seguem analogamente.

A métrica de acurácia é uma simples taxa de acerto, intuitivamente indicando a qualidade geral do modelo, enquanto as métricas de precisão e abrangência[8] são mais específicas, porém podem facilmente ser compreendidas num contexto de motor de busca: a precisão do algoritmo irá indicar qual fração dos documentos retornadas é realmente relevante à consulta feita, enquanto a abrangência dirá se realmente todas os documentos relevantes foram retornadas. [9]

Seguindo a lógica do motor de buscas, pode-se maximizar a medida de abrangência fazendo com que o motor de busca sempre retorne todos os documentos, ou, no caso da classificação binária, sempre dizendo que a observação é da classe de interesse, sacrificando a medida de precisão. Ao passo de que para maximizar a precisão, apenas necessitaria-se classificar uma observação como da classe de interesse quando existe uma confiança muito alta a respeito dessa classificação. Para evitar tais casos extremos, utiliza-se outra medida, também importante no contexto de recuperação de informação, chamada de **medida F1**, em inglês, *F1 measure* ou *F1 score*, que é a simples média harmônica entre precisão e a abrangência. [3] Tal métrica é interessante pelo fato de agregar igualitariamente os indicativos de precisão

[8] Também chamada, em inglês, de *recall*

e o abrangência e também por penalizar o valor final se houver alguma discrepância entre ambos. Por exemplo, se temos um abrangência perfeita de 100% e uma precisão de apenas 50%, a medida F1 seria de aproximadamente $0,66$, enquanto se estes valores fossem de 90% e 60%, respectivamente, a medida F1 seria de cerca de $0,72$. A fórmula da medida F1 é dada a seguir:

$$F_1 = 2\frac{\text{precisão} \times \text{abrangência}}{\text{precisão} + \text{abrangência}}$$

Para validar a usabilidade de um modelo treinado, as métricas de avaliação devem ser ponderadas de forma condizente com o problema. Por exemplo, se o objetivo do classificador é indicar que dados cadastrais estão corretos ou não, com o intuito de diminuir a intervenção humana durante uma possível rotina de correção cadastral, então a métrica de precisão poderia ser mais relevante, já que um falso positivo pode significar não atualizar um cadastro problemático.

# Capítulo 3

# Desenvolvimento e experimentos

## 3.1 Arquitetura utilizada

A arquitetura lógica do *framework* desenvolvido se baseia numa sequencia de passos, todos detalhados no capítulo anterior. Resumidamente, são eles:

1. **Pareamento de esquemas**: Nesta etapa são mapeados quais atributos de cada base de dados deverão ser comparados entre si. Esta configuração é feita manualmente antes da execução do algoritmo;

2. **Extração de dados**: Os dados são obtidos da base analisada e da base de referência;

3. **Pré processamento**: Passo de remoção de ruídos e padronização estrutural dos dados, afim de se facilitar as comparações;

4. ***Blocking***: Etapa de otimização do processo de *data matching*, responsável por evitar uma comparação exaustiva entre ambas as bases, removendo os casos muito discrepantes da comparação. No caso deste projeto, a etapa de *blocking* não foi crítica devido a presença de um identificador comum em ambas as bases;

5. **Extração de *features***: Dado um indivíduo da base de referência e um da

base analisada, o extrator de *features* retorna um vetor de valores numéricos contendo as medidas de similaridade escolhidas;

6. **Classificador binário**: Após treinado com inúmeros exemplos, o classificador binário, neste caso a regressão logística, irá dar o veredito final do processo de *data matching*, indicando se um par de registros são de fato equivalentes.

A Figura 3.1 tem por objetivo elucidar o processo descrito.

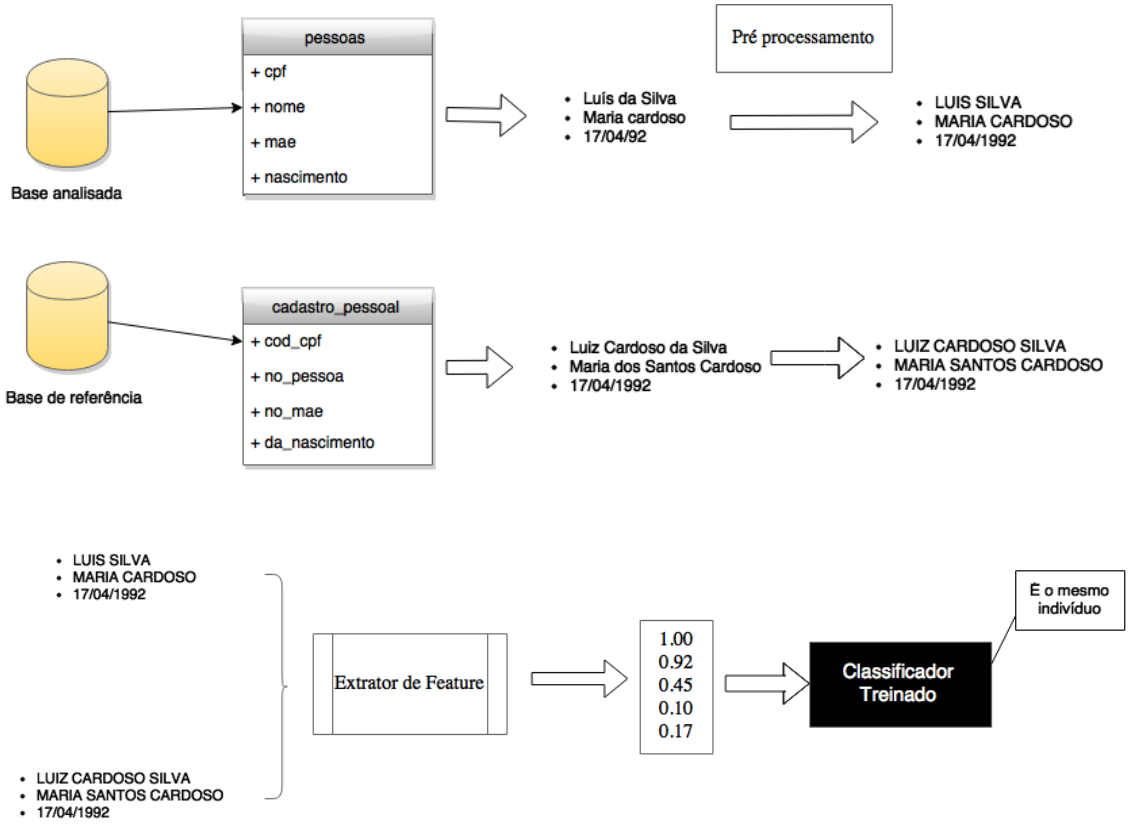


Figura 3.1: Visão geral do processo utilizado. Todos os dados apresentados na figura são apenas ilustrativos.

## 3.2 Detalhes da base

As técnicas descritas neste projeto foram aplicadas num subconjunto de um banco de dados privado real, contendo cerca de 250 mil registros de pessoas, na

qual se utilizou informações de nome completo, nome da mãe e data de nascimento, comparadas com uma base de pessoas físicas, usando o CPF como uma forma de cruzamento dos dados.

Uma amostra aleatória do conjunto de dados, que será usada como conjunto de treinamento, contendo 899 pares de pessoas físicas com o nome não idêntico, foi coletada e classificada manualmente por 3 pessoas, usando voto de maioria como critério de escolha da resposta final. A proporção final de cada uma das duas categorias do conjunto de treinamento (*registros equivalentes* ou *registros diferentes*) se encontra na Figura 3.2. Observa-se uma certa discrepância entre as proporções das classes: na amostra, os registros equivalentes são responsáveis por cerca de 72% dos dados, enquanto os registros diferentes foram quase 28%. Tal desbalanceamento é esperado no contexto deste estudo, já que acredita-se que a maioria dos pares de registros analisados sejam equivalentes, especialmente considerando-se o fato de que estes pares foram obtidos pelo uso de uma chave comum entre as bases: o CPF. O impacto deste desbalanceamento no algoritmo de aprendizado poderá ser analisado no experimento usando-se métricas como a taxa de falsos positivos e a taxa de falsos negativos, que podem indicar um enviesamento da classificação.

Importante observar também algumas outras características da base de dados que podem influenciar com o processo de aprendizado. Uma análise da completude dos dados de ambas as bases pode ser encontrada de forma resumida nas tabelas 3.1 e 3.2. Nota-se que apenas cerca de 10% da base analisada possui informação de data de nascimento ou nome da mãe, ao passo que na base de referência existem cerca de apenas 13% que não têm somente o nome da mãe. As consequências desta baixa completude no processo de aprendizado podem ser analisadas através dos parâmetros em $\theta$, que terão maiores valores absolutos para as *features* que o algoritmo considerar mais informativas para a tarefa de classificar a amostra.


Figura 3.2: Proporção das categorias do conjunto de dados

Tabela 3.1: Resumo da completude da base analisada

| Completude da base analisada | | |
|---|---|---|
| Data nascimento está preenchida? | Nome da mãe está preenchido? | Frequência |
| Sim | Sim | 60 |
| Sim | Não | 35 |
| Não | Sim | 0 |
| Não | Não | 804 |

Tabela 3.2: Resumo da completude da base de referência

| Completude da base de referência | | |
|---|---|---|
| Data nascimento está preenchida? | Nome da mãe está preenchido? | Frequência |
| Sim | Sim | 784 |
| Sim | Não | 115 |
| Não | Sim | 0 |
| Não | Não | 0 |

## 3.3 Tecnologias utilizadas

Para a implementação do experimento do *framework* proposto neste projeto foram utilizados diversos recursos. A linguagem de programação escolhida foi Java [2] em sua versão 1.8 devido suas novas características interessantes para o projeto como o dito suporte a *stream* de dados, facilitando enormemente a implementação de um processamento de dados em paralelo, o que agiliza o tempo de execução de certas rotinas proporcionalmente ao número de núcleos do computador utilizado.

Em conjunto com a linguagem Java, foram usados duas bibliotecas: jBLAS [7] e Second String [10]. A biblioteca jBLAS é responsável por prover operações matriciais otimizadas, importantes durante o passo de treinamento via método dos gradientes, que é um algoritmo tipicamente sequencial, não sendo trivialmente paralelizável. A biblioteca Second String provém as funções de medida de similaridade de *strings*, que são complexas de se implementar de forma eficiente. A biblioteca é o resultado de um estudo comparativo feito por Cohen em [11].

## 3.4 Detalhes do experimento

As etapas de pré-processamento para o nome da pessoa física e nome da mãe foram: conversão para letras maiúsculas e remoção de caracteres não alfanuméricos, acentos, preposições (de, das, da, do, dos) e espaçamentos extras. No caso das datas de nascimento, estas apenas tiveram seus formatos padronizadas e valores inválidos foram representados como nulo.

As *features* utilizadas pelo algoritmo foram a distância de Levenshtein, a similaridade de Jaro-Winkler e a similaridade de Monge-Elkan para nome da pessoa física e nome da mãe, assumindo valor zero para todos os casos quando algum campo fosse nulo ou vazio, já que este valor representa a não existência de qualquer similaridade. A distância de Levenshtein, por sua vez, foi normalizada para o intervalo de 0 a 1, com o intuito de tornar a distância equivalente a uma medida de similaridade e fazer com que *strings* de diferentes registros sejam comparáveis, onde o valor 1

indica *strings* idênticas e 0 indica que são totalmente diferentes. Isto foi feito usando a fórmula abaixo:

$$\text{Levenshtein normalizado}(a, b) = \frac{1 - Levenshtein(a, b)}{max(|a|, |b|)}$$

Onde $|a|$ e $|b|$ são os comprimentos das *strings* $a$ e $b$ em questão. Para datas de nascimento, usou-se também esta forma normalizada da distância de Levenshtein, com o intuito de obter erros de digitação.

O algoritmo de treinamento da regressão logística usado, o método dos gradientes, foi implementado conforme descrito no capítulo anterior, usando o logaritmo da verossimilhança como função objetivo e a heurística do *bold driver* para ajustar o coeficiente de aprendizado, conforme o algoritmo progride durante a otimização, retornando um passo atrás caso o erro não melhore. A configuração dos treinamentos realizados tiveram as seguintes especificações:

- Número máximo de iterações toleradas: 1000;
- Variação mínima tolerada da função objetivo: $10^{-3}$;
- Parâmetros iniciais de $\theta$: Todos com o valor 1;
- Taxa de aprendizado inicial: $0,001$;
- Incremento da taxa de aprendizado pelo *bold driver*: 10% ;
- Decremento da taxa de aprendizado pelo *bold driver*: 50% ;
- Taxa de aprendizado máxima: $0,1$;
- Taxa de aprendizado mínima: $0,0001$;

A fim de se analisar a qualidade da generalização dos modelos criados pelo algoritmo de aprendizado de máquina, dividiu-se o conjunto de dados totais em um conjunto de treino e um conjunto de teste. A proporção usada foi de 70% dos dados (629 registros) para treino e os restantes 30% (270 registros) para teste[4].

## 3.5 Resultados obtidos

Para se analisar o método proposto, primeiramente, irá se analisar um exemplo de caso individual de modelo obtido por meio do algoritmo, com os parâmetros finais apresentados na Tabela 3.3. Tal modelo, quando analisado em seu conjunto de teste, obteve os resultados quantitativos descritos nas Tabelas 3.4 e 3.5, sendo analisado com um limiar de decisão de 50%. Em geral, ele demonstra uma boa generalização, como resumido pelas estatísticas da Tabela 3.4, com valores de acurácia, precisão e abrangência consistentemente altos, onde todos estão acima de 90%, o que é compatível com os resultados obtidos em outras bases de dados nos trabalhos referenciados [5][8]. Na **matriz de confusão** da Tabela 3.5, pode-se observar que o modelo obteve resultados um pouco desequilibrados, proporcionalmente, entre falsos positivos e falsos negativos, com a taxa de falsos positivos mais alta, possivelmente devido ao desbalanceamento da amostra de dados. Ter uma taxa maior de falsos positivos pode ser indesejado, dependendo de certos critérios de aplicação do sistema, como já discutido anteriormente, porém isto pode ser regulado via um aumento no limiar de decisão, que também será analisado.

| **Atributo/Medida** | **Jaro-Winkler** | **Monge-Elkan** | **Levenstein** |
|---|---|---|---|
| Nome | 2,376870 | 4,340361 | 3,674466 |
| Nome da mãe | 0,757770 | 0,670388 | 0,645577 |
| Data de nascimento | - | - | 0,036718 |
| *Viés* | -6,538859 | | |

Tabela 3.3: Pesos de um modelo de regressão logística sem regularização.

| **Medida de avaliação** | **Valor** |
|---|---|
| Acurácia | 93,70% |
| Precisão | 92,96% |
| Abrangência | 98,40% |
| Medida F1 | 0.9560 |

Tabela 3.4: Medidas de avaliação de desempenho para o modelo apresentado na tabela 3.3

| **Classe real** | **Classe predita** | |
|---|---|---|
| - | Verdadeiro | Falso |
| Verdadeiro | 185 (68,51%) | 3 (1,11%) |
| Falso | 14 (5,19%) | 68 (25,19%) |

Tabela 3.5: Matriz de confusão do modelo apresentado na tabela 3.3

Observa-se que os parâmetros relativos ao nome da pessoa comparada possuem mais importância que os demais parâmetros, algo já esperado quando se analisou a completude da base de dados. Importante observar que uma *feature* é considerada mais relevante que as demais, no caso da regressão logística, quando seu valor absoluto é maior que os outros: valores negativos no peso de uma *feature* fazem com que conforme aquela *feature* aumente para uma observação, diminui-se a chance a daquela observação ser da classe de interesse. Neste caso, o nome da pessoa é cerca de cinco vezes mais relevantes que o nome da mãe e 300 vezes a mais que a data de nascimento, se o valor absoluto dos pesos de cada um destes atributos forem somados. Com o viés de $-6,538859$, este modelo apenas requer que os nomes sejam idênticos, depois de ocorrido o pré-processamento, para considerar que dois pares de registros se refiram a mesma pessoa, adotando um limiar de 50%, já, que neste caso, todas as medidas de similaridade seriam 1, somando pelo menos $10,39$, e fazendo com que a função sigmoide $g(z) = \frac{1}{1+e^{-z}}$ retorne $0,999969315$, ou seja, o modelo supõe que os pares de registros analisados têm cerca de 99,99% de chance de serem relativos a mesma pessoa, no caso específico deste base. Tal comportamento pode ser contraintuitivo, já que o peso todo da classificação recai sobre o nome, porém é razoável se considerando que, a priori, os pares de registros sempre possuem um mesmo identificador, o CPF.

Afim de se analisar a progressão do algoritmo de aprendizado, um gráfico da variação do erro em função do número de iterações do treinamento deste modelo é exibido na Figura 3.3, onde o erro é calculo tanto para o conjunto de treino, quanto para o conjunto de teste. A função de erro usada foi a raiz quadrada do erro médio quadrático [13], definida como:

$$\text{Erro Médio Quadrático} = \frac{1}{m}\sum_{i=0}^{m}(y_i - h_\theta(\mathbf{X}_i))^2$$

Onde se considera que existe um conjunto de teste com $m$ observações, $y_i$ representa a resposta da i-ésima observação, $X_i$ são as *features* da i-ésima observação e $h_\theta(x)$ é a função de hipótese. O erro médio quadrático é usado como uma segunda perspectiva à função de verossimilhança e também é mais intuitivo, sendo uma análise mais geral do classificador: casos dúbios, que seriam classificados de formas diferentes conforme o limiar do classificador variasse, serão serão levados em conta e ponderados no valor final. Como $y_i$ e a função de hipótese podem assumir apenas valores entre 0 e 1, a função de erro também será de 0 a 1 e a raiz quadrada tornará pequenas variações mais aparentes.

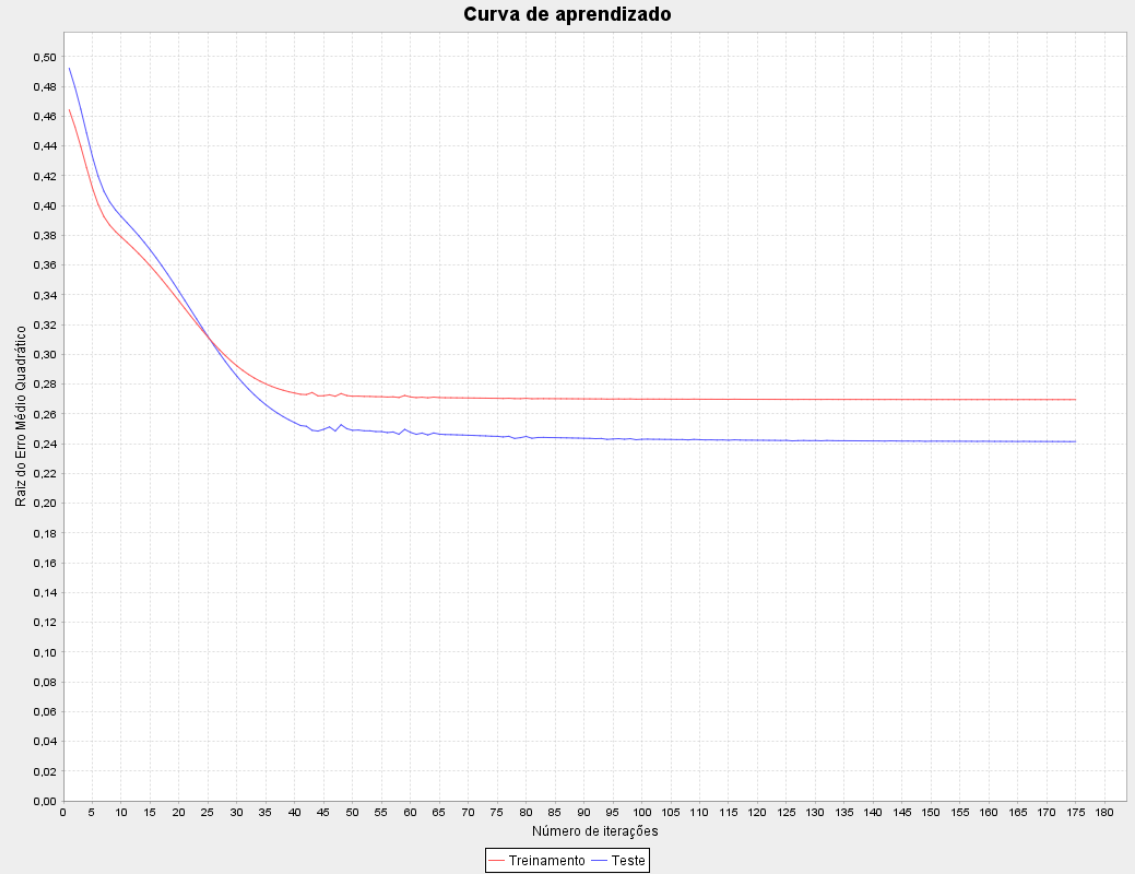


Figura 3.3: Curva de aprendizado do modelo da tabela 3.3

No gráfico da Figura 3.3, observa-se que nas primeiras 40 iterações existe uma

queda abrupta do erro, tanto no conjunto de treinamento, quanto no conjunto de teste. Em seguida, ocorrem pequenas oscilações no erro, que se estabilizam poucas iterações depois e por fim convergem para o valor final. O comportamento das curvas de teste e treino são bem similares: ambas estão sempre decrescendo de forma equivalente, sendo um indicativo de que o algoritmo não está perdendo sua capacidade de generalização ao longo das iterações. Por outro lado, nota-se que o algoritmo poderia ter parado antes até da iteração de número 80, já que o erro passa a variar muito pouco depois deste ponto, porém isto acaba não prejudicando sua performance: o erro do conjunto de teste não aumenta conforme o conjunto de treino melhora lentamente.

A análise apresentada anteriormente estuda o funcionamento do algoritmo de aprendizado de forma singular, porém como o processo de separação dos conjuntos de teste e treino é feito aleatoriamente, também é interessante observar como o algoritmo se comporta de forma geral. Para tal, o processo de separação dos dados e o treinamento foram realizado 100 vezes, onde todas as rodadas são independentes entre si e as métricas de avaliação são coletadas de cada um dos modelos gerados, sendo avaliados em onze diferentes limiares: começando do limiar 0 e indo até o limiar 1, dando passos de tamanho $0,1$. Para analisar tais métricas, os valores são resumidos nos gráficos das Figuras 3.4 e 3.5.

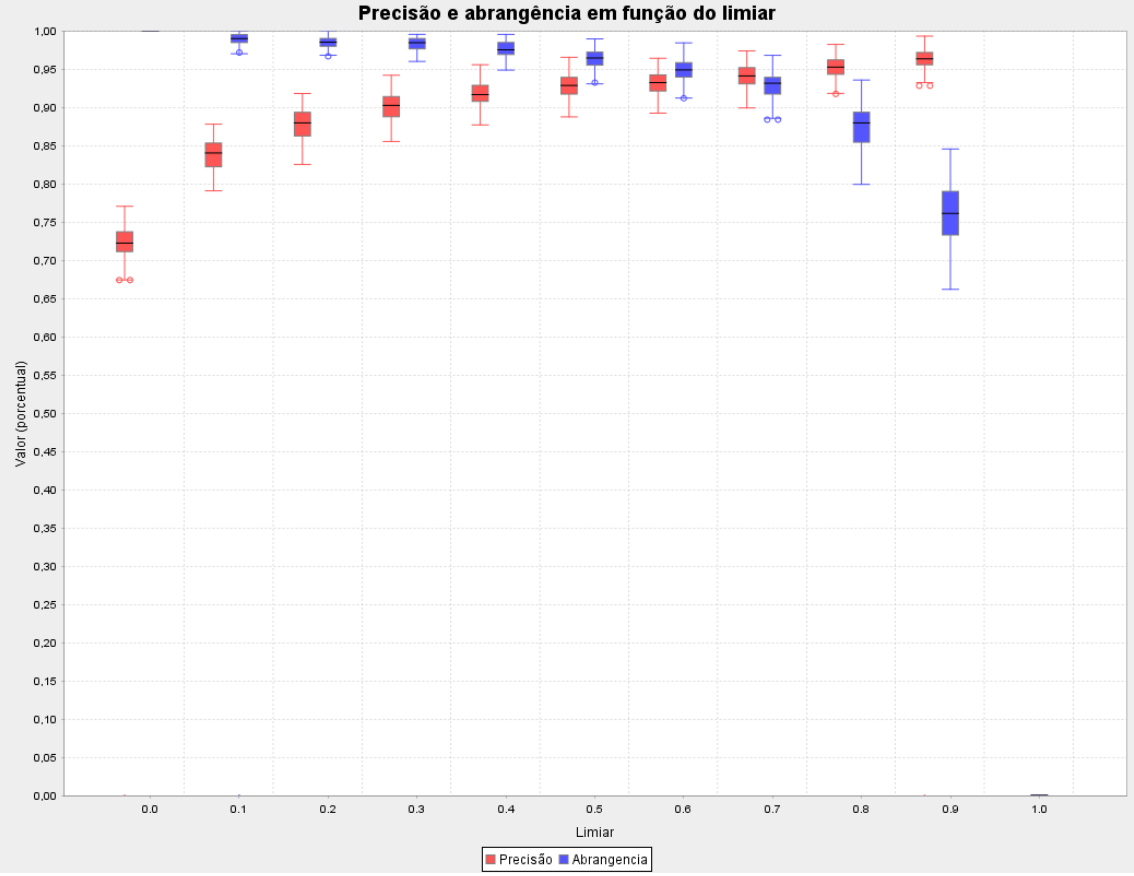


Figura 3.4: Gráfico de caixa da precisão e abrangência em função do limiar para 100 treinamentos (sem uso de regularização)

Com o intuito de apresentar a variação das taxas de precisão e abrangência dos 100 modelos treinados, o diagrama de caixa da Figura 3.4, ou *box-and-whisker plot* [27], foi usado. O gráfico demonstra cada limiar como uma faixa do eixo horizontal, a borda superior de cada retângulo preenchido representa o valor do terceiro quartil para a série daquela configuração, a borda inferior dos retângulos significam o valor do primeiro quartil e o traço horizontal dentro dos retângulos é a mediana. As retas verticais logo acima (ou abaixo) dos retângulos indicam qual maior (ou menor) observação, que mesmo fora do intervalo interquartil, definido como o intervalo entre o primeiro quartil e o terceiro quartil, não é considerada um *outlier* (valor atípico). O critério usual [27], e o utilizado no gráfico, é definido como: uma observação não é considerada *outlier* se ela está em uma distância de até $1,5$ vezes a distância

interquartil, partindo do primeiro ou terceiro quartil. Os círculos mostram de fato alguns pontos que são considerados *outliers* por este critério.

Analisando a Figura 3.4, observa-se de forma geral, que as métricas obtidas no experimento estão bem distribuídas em torno da mediana, já que as caixas são razoavelmente simétricas, também com relação a mediana, e que os pontos indicados como *outliers* pelo diagrama ocorrem bem próximos do limite de tolerância. Além disto, nota-se que o comprimento de quase todos os retângulos centrais de cada série não chegam a variar em 5%.

As taxas de abrangência da Figura 3.4, em geral, permanecem pouco dispersas e acima da faixa dos 95% até o limiar de $0,5$, na qual começa a cair e a dispersar bastante, até ficar por volta de 70% no limiar de $0,9$. Mesmo desta forma, obtêm-se valores em torno de 90% para as faixas de $0,6$ e $0,7$ do limiar.

Para as taxas de precisão, na Figura 3.4, pode-se observar que a dispersão não sofre grandes variações como no caso da abrangência, porém a curva tende a ter uma menor dispersão conforme o limiar aumenta. A precisão começa a ficar em torno de 90% já no limiar de $0,4$ e vai aumentando lentamente até chegar em seu máximo na casa dos 95% para o limiar de $0,9$. Em conjunto, os valores de precisão e abrangência são muito similares no limiar de $0,7$, ambos medianamente por volta de 93% e 94%, podendo ser um bom ponto de equilíbrio entre ambos.

Nota-se que para o limiar 0, a abrangência é máxima em todos os casos, enquanto para o limiar 1, a abrangência é mínima, como já discutido anterior. Outra singularidade observada é a precisão inexiste para o limiar 1, já que o classificador não aceita nenhuma observação como sendo da classe de interesse, então não existem nem verdadeiros positivos e nem faltos positivos, logo a formula da precisão é indeterminada por uma divisão por zero.

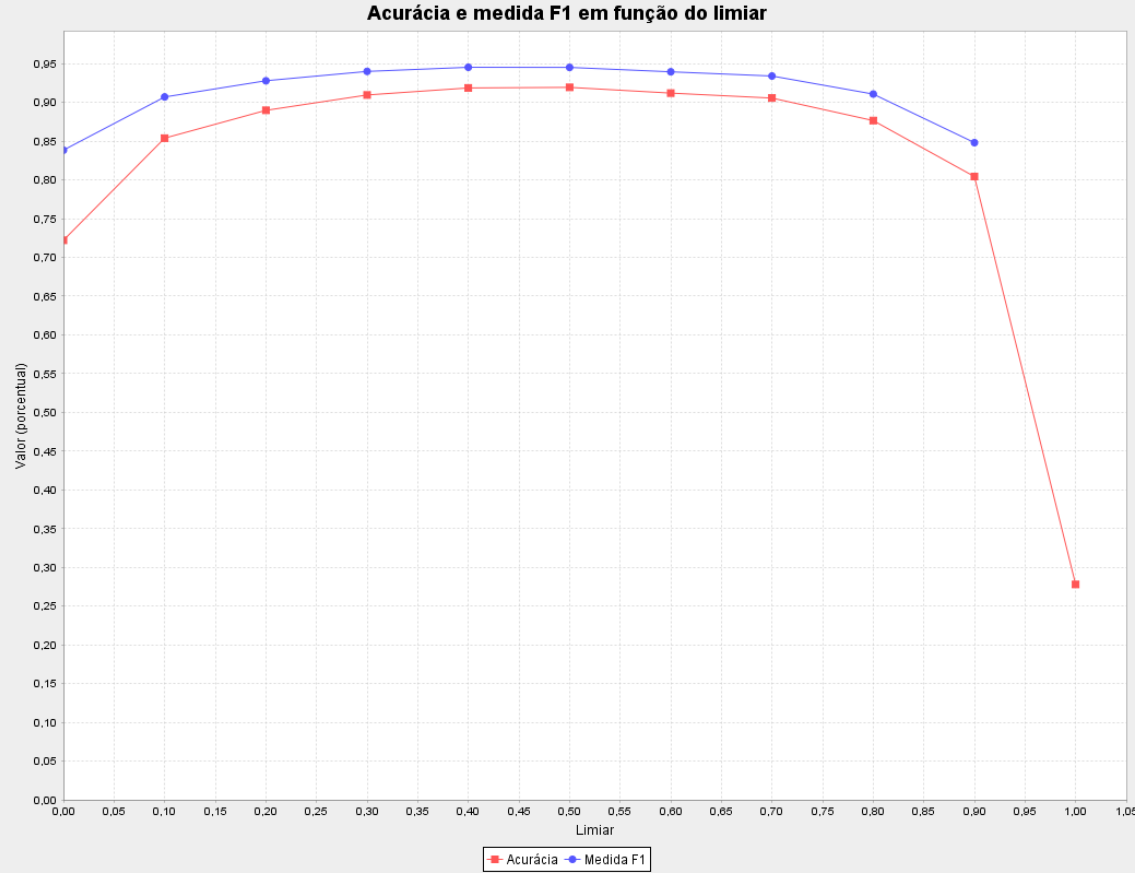


Figura 3.5: Média da acurácia e da medida F1 em função do limiar para 100 treinamentos (sem uso de regularização)

Por fim, pode-se a analisar a qualidade geral do algoritmo por dois aspectos diferentes pela Figura 3.5, que exibe a média aritmética das medidas F1 e acurácia para os 100 experimentos. As curvas apresentam comportamentos bem similares entre si, especialmente nos valores mais centrais do limiar. Novamente existe uma singularidade para o limiar de 1, no caso da medida F1, devido a um denominador zero vindo da medida de precisão neste ponto. Como esperado da regressão logística, ambas as métricas atingem seus valores máximos no limiar $0,5$, com cerca de 92% de acurácia e quase $0,95$ de medida F1, compatíveis com os valores obtidos nos trabalhos similares [5, 8] para outras bases de dados. Interessante notar também que tais valores permanecem pouco alterados para a faixa de limiar de $0,4$ até $0,7$, sendo uma boa faixa para se obter o limiar final, uma escolha na qual deve se levar

em conta a problemática causada por falsos positivos ou falsos negativos, como discutido anteriormente.

Para se analisar o efeito do uso da técnica de regularização, a iteração do método dos gradientes foi incrementada com o termo $-\lambda\theta$, apresentado no capítulo anterior e alguns resultados para os coeficientes de regularização $0,001$, $0,01$ e $0,1$ foram coletados. Primeiramente, para uma análise qualitativa, um gráfico similar ao da Figura 3.5 será exibido para os três coeficientes nas Figuras 3.6, 3.7 e 3.8.

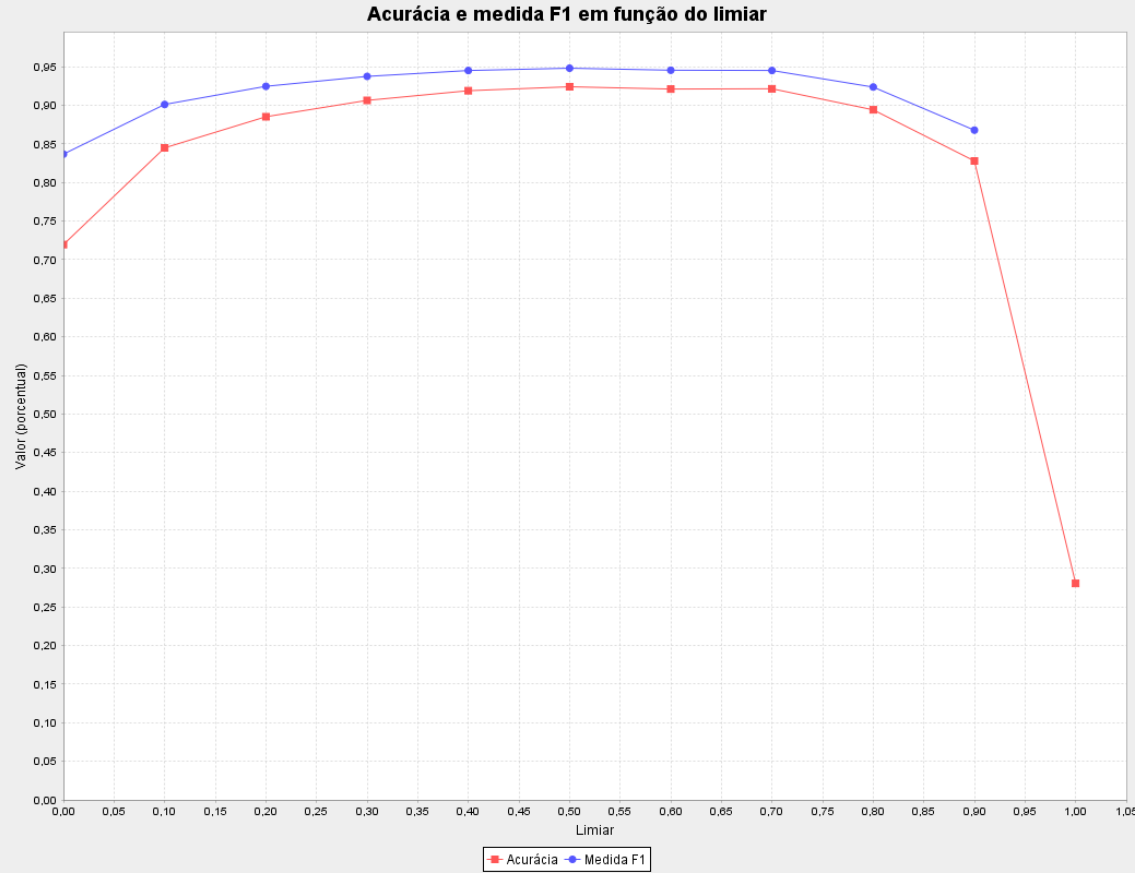


Figura 3.6: Média da acurácia e da medida F1 em função do limiar para 100 treinamentos (Coeficiente de regularização $\lambda = 0,001$)

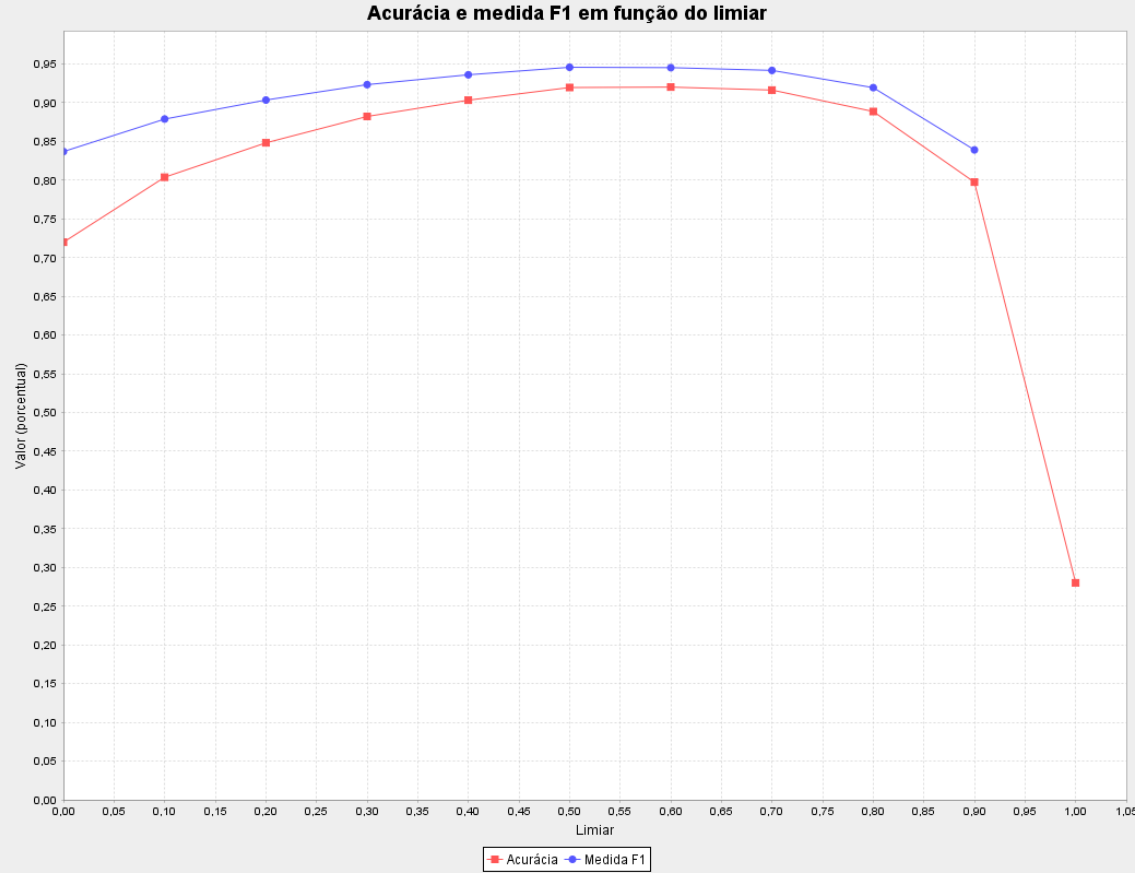


Figura 3.7: Média da acurácia e da medida F1 em função do limiar para 100 treinamentos (Coeficiente de regularização $\lambda = 0,01$)

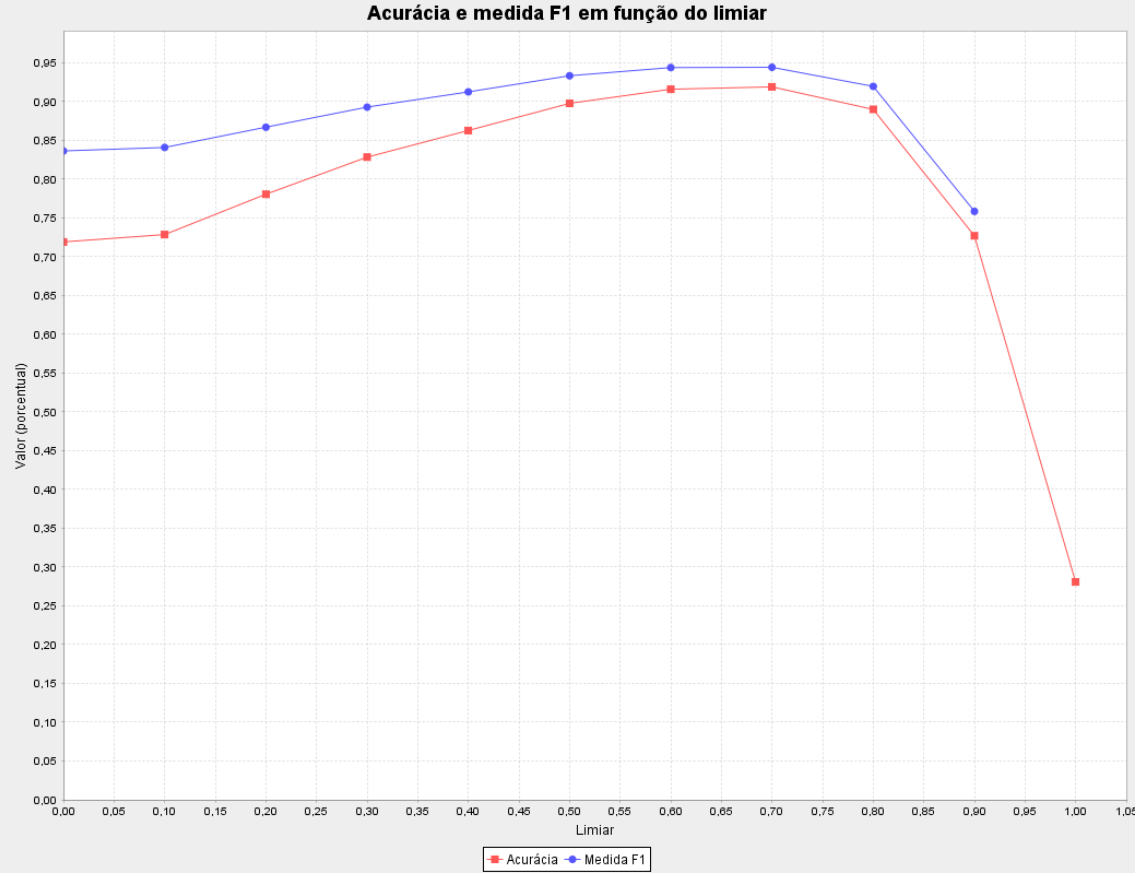


Figura 3.8: Média da acurácia e da medida F1 em função do limiar para 100 treinamentos (Coeficiente de regularização $\lambda = 0,1$)

No caso do coeficiente de 0,001, observa-se que existe muito pouca diferença desta curva com relação a curva dos modelos gerados sem regularização: em geral, este coeficiente de regularização gerou uma diferença muito sutil no resultado final. A Tabela 3.6 exibe um exemplo de modelo final, na qual é possível observar a similaridade com o modelo da Tabela 3.3.

| **Atributo/Medida** | **Jaro-Winkler** | **Monge-Elkan** | **Levenstein** |
|---|---|---|---|
| Nome | 2,553789 | 4,537180 | 3,844999 |
| Nome da mãe | 1,264520 | 1,245289 | 1,243098 |
| Data de nascimento | - | - | 0,977818 |
| *Viés* | -6,997848 | | |

Tabela 3.6: Pesos de um modelo de regressão logística com coeficiente de regularização 0.001

Para o caso do coeficiente $0,01$, pode se notar que a curva começa a perder seu platô, que ocorria pela faixa do limiar de $0,3$ a $0,7$, em comparação com as duas curvas anteriores, porém sem haver um incremento no seu valor máximo de acurácia ou medida F1. Para o coeficiente de regularização $0,1$, observa-se novamente este comportamento bem mais acentuado, na qual os limiares $0,6$ e $0,7$ passam a ser os máximos da função, sugerindo até que o coeficiente possa estar muito alto. Ao se observar as curvas de aprendizado destes casos nas Figuras 3.9 e 3.10, é possível observar que o algoritmo não convergiu naturalmente, parando na milésima iteração devido ao critério de parada.

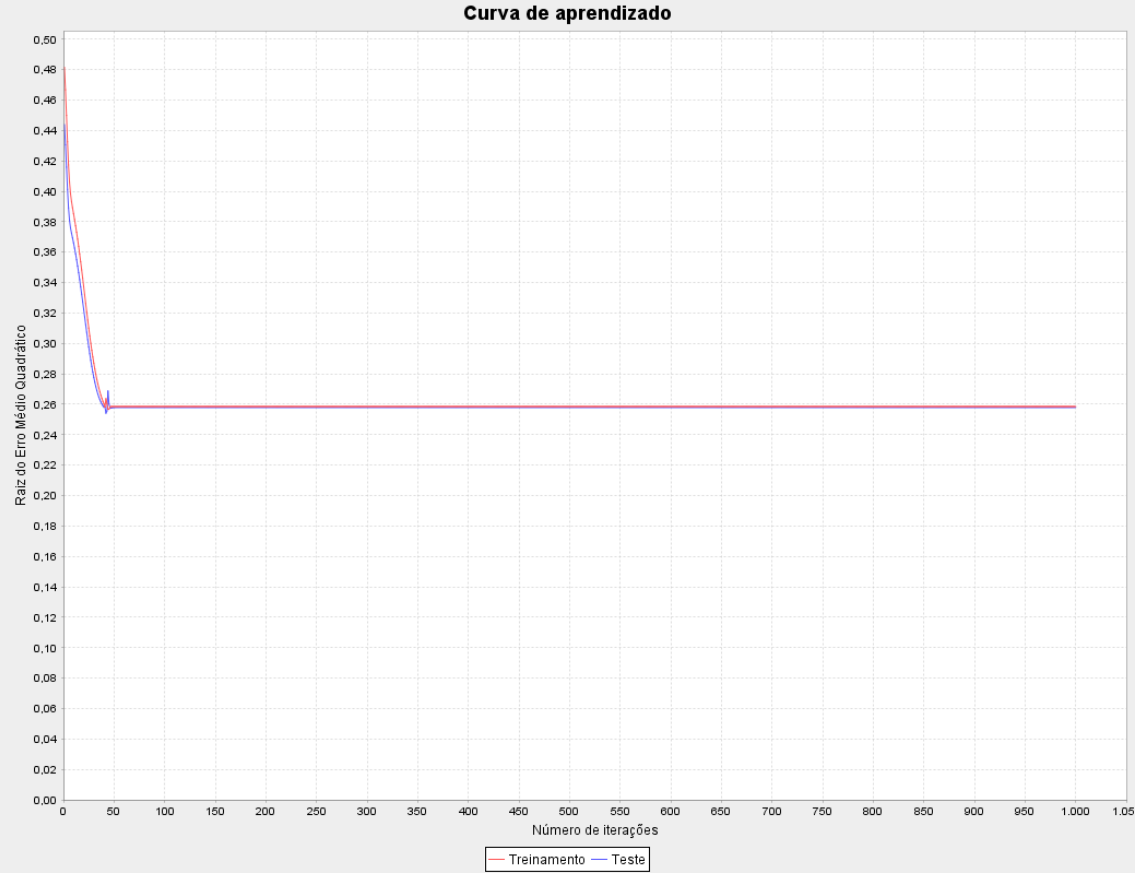


Figura 3.9: Exemplo de curva de aprendizado (Coeficiente de regularização $\lambda = 0,01$)

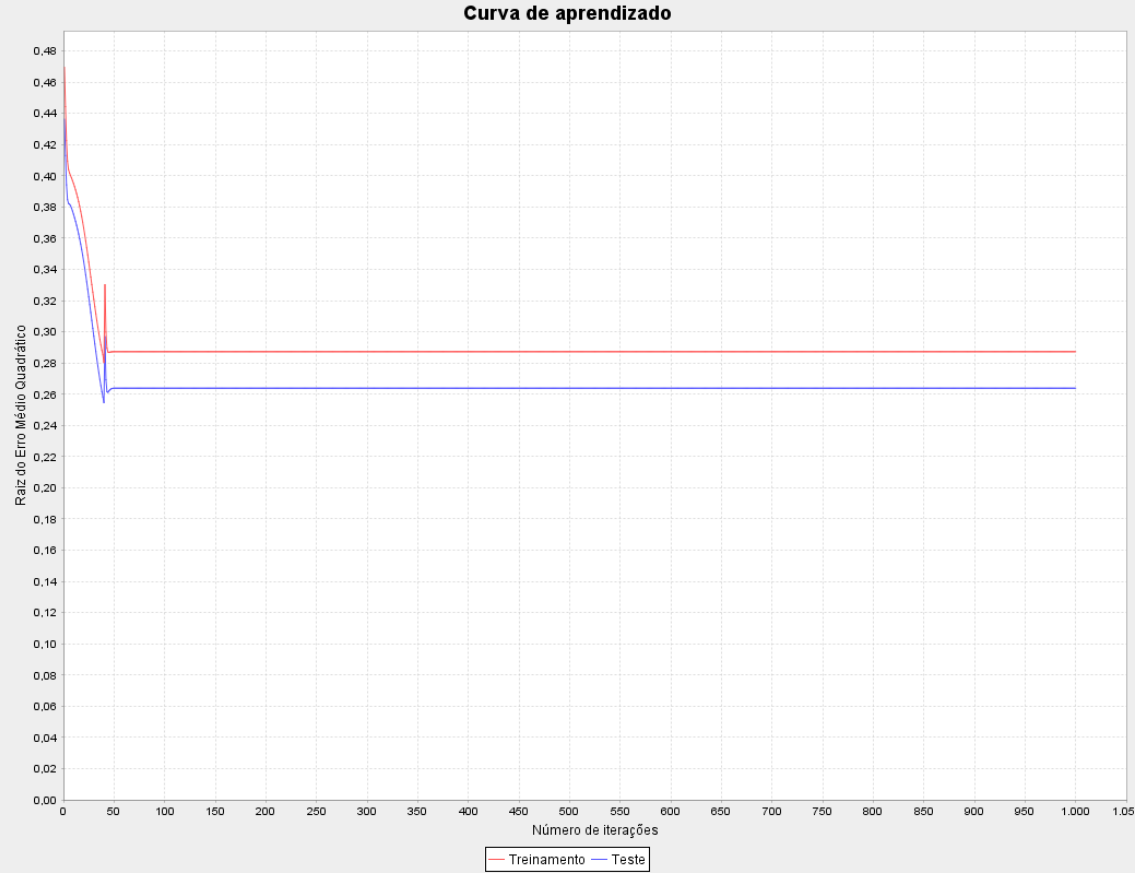


Figura 3.10: Exemplo de curva de aprendizado (Coeficiente de regularização $\lambda = 0,1$)

Por fim, observando os pesos de um modelo que usou o coeficiente $\lambda = 0,01$ para ser treinado e sua matriz de confusão, respectivamente nas Tabelas 3.7 e 3.8, nota-se que houve uma redução muito pequena no módulo destes pesos e que as matrizes de confusão são bem similares, ambos com relação ao modelo sem regularização. O uso da técnica de regularização, no caso desta base de dados, acaba não se mostrando necessário: os modelos gerados usando a técnica são bem similares aos modelos que não usam, tanto qualitativamente, quanto quantitativamente analisados, mesmo no caso do coeficiente $\lambda = 0,01$, que já modifica de forma perceptível a curva da Figura 3.7 com relação as Figuras 3.6 e 3.5, porém não contribui com uma melhoria das métricas de qualidade em geral.

| **Atributo/Medida** | **Jaro-Winkler** | **Monge-Elkan** | **Levenstein** |
|---|---|---|---|
| Nome | 1,583113 | 3,293635 | 3,745271 |
| Nome da mãe | 0,394293 | 0,394228 | 0,385819 |
| Data de nascimento | - | - | 0,417803 |
| *Viés* | -5,284217 | | |

Tabela 3.7: Pesos de um modelo de regressão logística com coeficiente de regularização 0,01

| **Classe real** | **Classe predita** | |
|---|---|---|
| - | Verdadeiro | Falso |
| Verdadeiro | 194 (71,85%) | 9 (3,33%) |
| Falso | 11 (4,07%) | 56 (20,74%) |

Tabela 3.8: Matriz de confusão do modelo apresentado na Tabela 3.7 (regularização $\lambda = 0,01$)

# Capítulo 4

# Conclusão

Neste trabalho foi proposto um estudo de técnicas de aprendizado supervisionado, usando a regressão logística como caso de estudo, aplicado durante a fase de comparação de registros em um processo de *data matching*. Isto foi feito implementado-se uma *framework* de *data matching* na linguagem Java, e usando como descritores algumas das medidas de similaridades de *string*, são elas: distância de Leveishtein, similaridade de Jaro e similaridade de Monge-Elkan.

Quando comparado com trabalhos similares, como o de Bilenko[5] e Camacho[8], resultados compatíveis são obtidos pelas métricas de performance, como a precisão e a medida F1, todas acima do valor de 90%, mesmo quando o algoritmo é testado com inúmeras configurações do conjunto de treino. É importante lembrar que diferentes bases de dados foram usadas para se calcular as métricas, o que impede uma comparação mais justa. Esforços adicionais também não exibiram melhoras significativas no modelo de classificação gerado pela regressão logística.

Tendo em vista estes fatos, pode-se dizer que o objetivo final desta monografia de desenvolver um modelo de *data matching* usando aprendizado de máquina foi alcançado.

## 4.1 Trabalhos futuros

Como trabalhos posteriores, se sugere uma comparação aprofundada entre as diversas ferramentas de *data matching* ou desduplicação existentes e o método apresentado nesta monografia, de forma que a comparação ocorra em situações igualitárias, com relação a base de dados, conjunto de treino e algoritmo de *blocking*.

Algoritmos de codificação fonética, como o Metaphone, são uma vertente de pesquisa interessante, visto que poucos algoritmos desse tipo foram desenvolvidos para português, até onde se sabe, e eles são muito úteis para as mais diversas formas de busca, além de serem usados como uma forma de *blocking*.

Outro ponto interessante seria o desenvolvimento de um sistema ou módulo semi-automático de pareamento de esquemas de bancos de dados, o que facilitaria muito a configuração das ferramentas de *data matching*, qualidade de dados, migração de dados ou quaisquer tarefas que envolvam o uso de diversas bases heterogêneas. Tal sistema idealmente usaria informações de metadados de ambas as bases, como nomes de atributos, nomes de entidades, relacionamentos entre entidades e tipos de atributos, além dos próprios dados para sugerir possíveis mapeamentos entre as bases de dados para serem usados em algum processamento posterior. [25]

# Referências


[1] Cross validation. https://www.cs.cmu.edu/~schneide/tut5/node42.html. Acessado em 6/12/2015.

[2] Java se development kit 8. http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html. Acessado em 6/12/2015.

[3] F1-measure. In *Encyclopedia of Machine Learning*, C. Sammut e G. Webb, Eds. Springer US, 2010, pp. 397–397.

[4] Holdout evaluation. In *Encyclopedia of Machine Learning*, C. Sammut e G. Webb, Eds. Springer US, 2010, pp. 506–507.

[5] Bilenko, M., Mooney, R., Cohen, W., Ravikumar, P., e Fienberg, S. Adaptive name matching in information integration. *IEEE Intelligent Systems 18*, 5 (2003), 16–23.

[6] Bishop, C. M., e others. *Pattern recognition and machine learning*, vol. 4. springer New York, 2006.

[7] Braun, M. L. jblas - linear algebra for java. http://jblas.org/. Acessado em 11/12/2015.

[8] Camacho, D., Huerta, R., e Elkan, C. An evolutionary hybrid distance for duplicate string matching. Relatório técnico, Universidad Autonoma de Madrid, 2008.

[9] Carterette, B. Precision and recall. In *Encyclopedia of Database Systems*, L. LIU e M. OZSU, Eds. Springer US, 2009, pp. 2126–2127.

[10] Cohen, W., Ravikumar, P., e Fienberg, S. Second string. `http://secondstring.sourceforge.net/`. Acessado em 11/12/2015.

[11] Cohen, W., Ravikumar, P., e Fienberg, S. A comparison of string metrics for matching names and records. In *Kdd workshop on data cleaning and object consolidation* (2003), vol. 3, pp. 73–78.

[12] Cormen, T. H. *Introduction to algorithms*. MIT press, 2009.

[13] DeGroot, M. H., Schervish, M. J., Fang, X., Lu, L., e Li, D. *Probability and statistics*, vol. 2. Addison-Wesley Reading, MA, 1986.

[14] Du, K., e Swamy, M. *Neural Networks and Statistical Learning*. SpringerLink : Bücher. Springer, 2013.

[15] Elmagarmid, A. K., Ipeirotis, P. G., e Verykios, V. S. Duplicate record detection: A survey. *Knowledge and Data Engineering, IEEE Transactions on 19*, 1 (2007), 1–16.

[16] Fellegi, I. P., e Sunter, A. B. A theory for record linkage. *Journal of the American Statistical Association 64*, 328 (1969), 1183–1210.

[17] Garshol, L. M. Duke. `https://github.com/larsga/Duke`. Acessado em 6/12/2015.

[18] Gregg, F., e Eder, D. Dedupe python library. `https://github.com/datamade/dedupe`. Acessado em 6/12/2015.

[19] Hilbert, M., e López, P. The world's technological capacity to store, communicate, and compute information. *science 332*, 6025 (2011), 60–65.

[20] Jaro, M. A. *Unimatch: A record linkage system: Users manual*. Bureau of the Census, 1978.

[21] Levenshtein, V. I. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady* (1966), vol. 10, pp. 707–710.

[22] Monge, A. E., Elkan, C., e others. The field matching problem: Algorithms and applications. In *KDD* (1996), pp. 267–270.

[23] Murphy, K. P. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[24] Ng, A. Cs229 lecture notes. *CS229 Lecture notes 1*, 1 (2000), 1–3.

[25] Rahm, E., e Bernstein, P. A. A survey of approaches to automatic schema matching. *the VLDB Journal 10*, 4 (2001), 334–350.

[26] Ting, K. Precision and recall. In *Encyclopedia of Machine Learning*, C. Sammut e G. Webb, Eds. Springer US, 2010, pp. 781–781.

[27] Weisstein, E. W. Box-and-whisker plot. From MathWorld—A Wolfram Web Resource. `http://mathworld.wolfram.com/Box-and-WhiskerPlot.html`. Acessado em 6/12/2015.

[28] Weisstein, E. W. Method of steepest descent. From MathWorld—A Wolfram Web Resource. `http://mathworld.wolfram.com/MethodofSteepestDescent.html`. Acessado em 6/12/2015.

[29] Weisstein, E. W. Metric space. From MathWorld—A Wolfram Web Resource. `http://mathworld.wolfram.com/MetricSpace.html`. Acessado em 6/12/2015.

[30] Winkler, W. E., e Thibaudeau, Y. An application of the fellegi-sunter model of record linkage to the 1990 us decennial census. *US Bureau of the Census* (1991), 1–22.