CHRISTIANE FALEIRO SIDNEY

PREDIÇÃO DE DESEMPENHO PARA JUNÇÕES POR SIMILARIDADE BASEADAS EM CONJUNTOS

LAVRAS – MG

2014

CHRISTIANE FALEIRO SIDNEY

PREDIÇÃO DE DESEMPENHO PARA JUNÇÕES POR SIMILARIDADE BASEADAS EM CONJUNTOS

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Banco de Dados e Engenharia de *Software*, para a obtenção do título de Mestre.

Orientador

Dr-Ing. Leonardo Andrade Ribeiro

LAVRAS – MG

2014

Ficha Catalográfica Elaborada pela Coordenadoria de Produtos e Serviços da Biblioteca Universitária da UFLA

Sidney, Christiane Faleiro.
Predição de desempenho para junções por similaridade baseadas em conjuntos / Christiane Faleiro Sidney. – Lavras : UFLA, 2014.
91 p. : il.

Dissertação (mestrado) – Universidade Federal de Lavras, 2014.
Orientador: Leonardo Andrade Ribeiro.
Bibliografia.

1. Junção por similaridade. 2. Aprendizagem de máquina. 3. Predição de desempenho para consultas. 4. Integração de dados. 5. Limpeza de dados. I. Universidade Federal de Lavras. II. Título.

CDD – 005.74

CHRISTIANE FALEIRO SIDNEY

PREDIÇÃO DE DESEMPENHO PARA JUNÇÕES POR SIMILARIDADE BASEADAS EM CONJUNTOS

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Banco de Dados e Engenharia de *Software*, para a obtenção do título de Mestre.

APROVADA em 27 de fevereiro de 2014.

Dr. Denilson Alves Pereira	UFLA
Dr. Rafael Andrade	IFC/CTI
Dr. André Luiz Zambalde	UFLA

Dr-Ing. Leonardo Andrade Ribeiro
Orientador

LAVRAS - MG

2014

AGRADECIMENTOS

Ao meu orientador, professor Leonardo Andrade Ribeiro, pelos seus ensinamentos, orientações, palavras de incentivo, "puxões de orelha", paciência e dedicação.

Aos professores do programa, pelos conhecimentos compartilhados.

Aos funcionários do Departamento de Ciência da Computação, pela disponibilidade e gentileza.

Aos colegas de mestrado, pelos momentos bons e ruins divididos.

Aos colegas de laboratório, que sempre estiveram ao meu lado, sempre dando força e apoio.

Ao Laboratório de Estudos e Projetos em Manejo Florestal - LEMAF, pelo apoio durante esta caminhada.

Ao meu namorado, pela presença incansável com que me apoiou ao longo do período.

A minha família, pelo incentivo e apoio nesta etapa.

RESUMO

Previsão do tempo de execução de consultas é essencial para muitas tarefas importantes relacionadas ao gerenciamento de banco de dados baseado em nuvem, incluindo provisionamento de recursos, controle de admissão e precificação de serviços. Recentemente, há grandes esforços na construção de modelos de previsão para estimar o tempo de execução de consultas SQL tradicionais. Embora adequadas para cargas de trabalho OLTP/OLAP, essas abordagens são insuficientes para modelar o desempenho de atividades envolvendo análises complexas de dados, como limpeza e integração de dados. Essas atividades são baseadas tipicamente em operações de similaridade, que, por sua vez, são radicalmente diferentes dos operadores relacionais regulares. Neste trabalho, consideramos modelos de previsão de tempo para junções por similaridade baseadas em conjuntos. Por meio do estudo de técnicas de otimização popularmente utilizadas em algoritmos de junção por similaridade, foram identificadas um conjunto de *features* relevantes, que são usadas na construção de modelos de previsão baseadas em aprendizagem de máquina estatística. Uma extensa avaliação experimental é apresentada para confirmar a precisão da nossa abordagem.

Palavras-chave: Junção por Similaridade. Aprendizagem de Máquina. Predição de Desempenho para Consultas. Integração de Dados. Limpeza de Dados.

ABSTRACT

Query performance prediction is essential for many important tasks related to cloud-based database management including resource provisioning, admission control, and pricing. Recently, there has been great interest in building prediction models to estimate execution time of traditional SQL queries. While suitable for typical OLTP/OLAP workloads, these existing approaches are insufficient to model performance of complex data processing activities for deep analytics such as cleaning and integration of data. These activities are largely based on similarity operations, which are radically different from regular relational operators. In this dissertation, we consider prediction models for set similarity joins. We exploit knowledge of optimization techniques and design details popularly found in set similarity join algorithms to identify relevant features, which are then used to construct prediction models based on statistical machine learning. We present an extensive experimental evaluation to confirm the accuracy of our approach.

Keywords: Similarity Join. Cloud Databases, Machine Learning. Query Performance Prediction. Data Integration. Data Cleaning.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização e Motivação

A computação em nuvem trouxe uma grande evolução no modelo de armazenamento, disponibilidade e processamento de dados. Com isso, mudou-se o hábito de manter dados nos próprios computadores ou servidores para dispô-los em estruturas distribuídas com alto grau de disponibilidade e escalabilidade. Esta nova estrutura de gestão de dados permite de forma onipresente, conveniente e sob demanda um acesso configurável a um *pool* compartilhado dos recursos de computação (LEHNER; SATTLER, 2013).

Porém, esses sistemas têm a difícil tarefa de gerenciar recursos compartilhados seguindo um acordo de nível de serviço (ANS ou SLA, do inglês *Service Level Agreement*). E dentro deste contexto, surge o desafio de prever o tempo de execução de consultas, que podem apoiar a tomada de decisões importantes, incluindo:

a) **Provisionar recursos**: avaliação de desempenho para uma dada configuração de *hardware* é fundamental para minimizar o risco de subestimar ou superestimar recursos necessários. As necessidades podem ser rápida e elasticamente provisionadas, em alguns casos automaticamente, aumentando ou diminuindo a necessidade de recursos. Desta forma, estes parecem ilimitados e podem ser adquiridos em qualquer quantidade e a qualquer hora;
b) **Controle de admissão**: consultas recebidas podem ser bloqueadas, enfileiradas ou executadas imediatamente em função do tempo de execução estimado, pois sistemas de nuvem controlam e otimizam o uso de recursos automaticamente. O uso de recursos pode ser

monitorado, controlado e reportado de forma transparente entre o consumidor e o provedor dos serviços;

c) **Mecanismos de preços**: decidem como as solicitações de serviços são cobradas em detrimento do uso de recursos virtualizados como CPU, I/O e memória. Provedores na nuvem podem prover serviços baseados em garantias de alto nível e métricas de desempenho mensuráveis, tais como tempo de resposta da consulta e taxa de transferência. Preços servem como base para a gestão da oferta e demanda de recursos computacionais, e facilita a priorização de alocação de recursos.

A popularização da computação em nuvem e o crescente entusiasmo em torno do *"Big Data"*, que Kaisler et al. (2013) definem como uma quantidade de dados que está além da capacidade tecnológica de armazenar, gerenciar e processar de forma eficiente, tem aumentado a necessidade de suporte analítico em serviços de banco de dados baseados em nuvem (CHAUDHURI, 2012).

Atualmente tem sido produzida uma grande quantidade de trabalhos sobre previsão do tempo de execução de consultas *SQL* (mais detalhes na Sessão 2.4). No entanto, as abordagens atuais para a previsão de desempenho são insuficientes para modelar atividades complexas envolvendo análise massiva de dados, como por exemplo, atividades de integração e limpeza de dados.

Neste contexto, junções por similaridade são operações essenciais, pois são imprescindíveis para suporte analítico (BAYARDO; MA; SRIKANT, 2007; CHAUDHURI; GANTI; KAUSHIK, 2006; RIBEIRO; HÄRDER, 2011; XIAO et al., 2008). Este tipo especial de junção retorna todos os pares de tuplas em uma relação que são similares acima de um limite estipulado. A similaridade entre pares de tuplas é quantificada numericamente através de uma função de similaridade usada no predicado da junção.

Junções por similaridade são tarefas complexas, pois cálculos de similaridade são muito mais custosos computacionalmente que simples comparações *byte* a *byte*. Além disso, utilizam como modelo de processamento a lista invertida (BAYARDO; MA; SRIKANT, 2007; RIBEIRO; HÄRDER, 2011; SARAWAGI; KIRPAL, 2004; XIAO et al., 2008), que torna a previsão de tempo uma tarefa difícil, pois é, radicalmente, diferente das abordagens baseadas na tecnologia de banco de dados relacionais (CHAUDHURI; GANTI; KAUSHIK, 2006).

Algoritmos para junção por similaridade baseados em listas invertidas possuem desempenho consideravelmente melhor do que soluções baseadas em tecnologia puramente relacional, como visto no trabalho (XIAO et al., 2008), e, portanto, são susceptíveis a ser a melhor solução em sistemas baseados em nuvem com rigorosos requisitos de escalabilidade. Além disso, os tempos de execução de diferentes execuções podem variar drasticamente dependendo dos limiares e distribuição de dados.

Neste trabalho, propõem-se uma abordagem para a previsão de tempo de junção por similaridade baseada em conjuntos. De acordo com o nosso conhecimento, este é o primeiro trabalho que aborda esse problema. Foram exploradas técnicas de otimização populares e algoritmos que selecionam criteriosamente um conjunto de estatísticas que ditam o desempenho da operação de junção por similaridade. Estes dados são usados juntamente com os valores de limiar como entrada para uma técnica de aprendizagem de máquina. Uma observação importante é a de que todos os dados estatísticos podem ser facilmente colhidos na fase de pré-processamento do algoritmo. Adotou-se uma abordagem experimental para amostrar o espaço de distribuição de dados e valores de limiares. Esta estrutura pode ser facilmente estendida para aumentar a precisão da estimativa. Por fim, foi realizada uma extensa avaliação para validar a precisão da abordagem.

1.2 Objetivo Geral

O objetivo geral deste trabalho foi desenvolver modelos de predição que sejam capazes de determinar o tempo de execução de um algoritmo de junção por similaridade. Para alcançar tal objetivo, foram definidos cinco objetivos específicos que são:

a) Estruturação da coleção de dados para treinamento para que cubram uma parte representativa do espaço de entrada;
b) Definição das *features* que serão extraídas para criação do modelo;
c) Construção de modelos utilizando diferentes técnicas de aprendizagem de máquina, verificando quais se adéquam melhor ao problema;
d) Criação de um *framework* extensível para geração e aplicação de modelos de desempenho para execução de junções por similaridade;
e) Verificação da generalização dos modelos por meio de métricas estatísticas sobre bases de dados de teste.

1.3 Estrutura do Trabalho

O restante deste trabalho está organizado da seguinte forma: na Seção 2 são apresentados conceitos sobre funções por similaridade, junções por similaridade e regressão. A definição do problema está na Seção 3. A Seção 4 expõe a metodologia. Os resultados são listados na Seção 5. A conclusão é apresentada na Seção 6.

2 REFERENCIAL TEÓRICO

Neste capítulo serão descritos os principais conceitos acerca de funções por similaridade, junções por similaridade e regressão, além de apresentar trabalhos relacionados ao tema estudado.

2.1 Conceito de Similaridade

Em muitas aplicações do mundo real é necessário realizar buscas que tenham como objetivo encontrar todos os pares de objetos cuja semelhança esteja acima de um limite especificado (BAYARDO; MA; SRIKANT, 2007). Assim, uma maneira quantitativa para definir se dois objetos são duplicados é usando *funções de similaridade* (FS), que fornecem uma medida numérica para o grau em que duas entidades são iguais ou diferentes (RIBEIRO; HÄRDER, 2011).

A escolha do tipo de função de similaridade depende da finalidade da tarefa. Em trabalhos anteriores, foram gastos esforços na identificação de uma função que seja a melhor para todos os domínios e nenhuma foi encontrada (ZOBEL; MOFFAT, 1998). Duas classes de funções de similaridade que têm sido amplamente utilizadas são as funções de distância de edição e as baseadas em conjuntos. *Funções de similaridade baseadas em edição* são definidas como o custo mínimo de operações para transformar uma *string* em outra. As operações de edição incluem copia, inserção, exclusão e substituição de caracteres (LEVENSHTEIN, 1966).

Funções de similaridade baseadas em conjuntos podem ser reduzidas a um problema de interseção de conjuntos, em que diversas maneiras podem ser utilizadas para obter diferentes noções de similaridade. Essas funções retornam valores entre 0 e 1, dependendo da cardinalidade dos conjuntos e da interseção

dos mesmos. Este trabalho irá focar somente na classe de função baseada em conjuntos, pois este método tem boa eficácia, resultados com boa acurácia, possui alto grau de eficiência e é considerado muito versátil (RIBEIRO, 2010). No restante deste documento, usaremos apenas o termo função de similaridade para denotar função de similaridade baseada em conjuntos, exceto quando dito explicitamente o contrário.

2.1.1 Tokenização

Para a criação de conjuntos, as entradas devem ser pré-processadas. Este pré- processamento, denominado de *tokenização*, é a transformação de uma *string* em um conjunto de unidades básicas de informação que chamaremos de *tokens*.

Tokens podem ser definidos de diversas maneiras como, por exemplo, palavras de uma *string* ou *q-grams*. Um *q-gram* é uma *substring* de tamanho *q*, obtida pelo deslizamento de uma janela de tamanho *q* nos caracteres da *string* original (UKKONEN, 1992).

Exemplo 1: Considere s_1 a *string* "Christiane". O conjunto de *q-grams* gerado com $q = 3$ é $x = \{Chr, hri, ris, ist, sti,tia, ian, ane\}$.

2.1.2 Ponderação

Em algumas aplicações, pode-se atribuir relevância a *tokens* mais raros por meio da ponderação. Uma forma de se realizar esta operação é utilizando a técnica *IDF* (*Inverse Document Frequency*), que calcula a pesagem inversamente proporcional à frequência do *token* na base de dados, assim, captando a intuição de que características raras são mais relevantes para a

avaliação de similaridade (ROBERTSON; JONES, 1988). O peso *IDF* de um *token* pode ser calculado utilizando a seguinte fórmula:

$$w(i) = 1 + \log\frac{N}{n_i} \quad (1)$$

Em que N é o número total de conjuntos e n_i é o número de conjuntos em que o *token i* aparece na coleção.

O tamanho do conjunto x considerando os pesos, denotado por $w(x)$, é dado pelo somatório dos pesos dos *f tokens* do conjunto, dado por:

$$w(x) = \sum_{i=1}^{f} w\,(i) \quad (2)$$

Quando a estratégia de ponderação não for utilizada (conjuntos não ponderados), podemos reduzir o caso e simplesmente associar o peso de cada *token* presente no conjunto ao valor 1, assim, o tamanho do conjunto é exatamente igual a sua cardinalidade (número de *tokens* que o conjunto possui).

Exemplo 2: Consideremos as *strings* "Christiane" e "Cristiane" e seus respectivos conjuntos de *tokens*, x = *{ch, hr, ri, is, st,ti, ia, an, ne}* e y = *{cr, ri, is, st, ti, ia, an, ne}*. Conforme a frequência dos *tokens* presentes nestes conjuntos, considere que os *tokens* possuem peso de valor 1, com exceção dos *tokens* {ch, hr, cr} que possuem peso aproximado de 1,3. Sendo assim, $|x| = 9,6$ e $|y| = 8,3$.

Por simplicidade, os exemplos seguintes serão baseados em conjuntos não ponderados, e com isso, $w(x)$ será denotado simplesmente por $|x|$.

2.1.3 Cálculo de Interseção

Após a geração de conjuntos e ponderação destes (nos casos de conjuntos ponderados), são calculadas as funções de similaridade (FS) que verificam algebricamente a similaridade entre dois conjuntos. As funções mais populares são *Jaccard*, *Dice* e *Cosseno*. A Tabela 1 apresenta as definições destas funções. Estas funções possuem a propriedade de serem comutativas, ou seja, $FS(x, y) = FS(y, x)$.

Exemplo: Consideremos as *strings* "Christiane" e "Cristiane" e seus res- pectivos conjuntos de *tokens*, $x = \{ch, hr, ri, is, st,ti, ia, an, ne\}$ e $y = \{cr, ri, is, st, ti, ia, an, ne\}$. O cálculo de similaridade, utilizando a função de *Jaccard (FJ)* é $FJ(x, y) = 7/(9 + 8 - 7) = 0,70$.

Definição 1 (Função de similaridade): Tendo um universo finito U de *tokens* e uma coleção de conjuntos C, em que cada conjunto consiste em um número de *tokens* de U , temos $FS(x, y)$ como uma função de similaridade que mapeia cada par do conjunto x e y para um número em $[0, 1]$.

Tabela 1 Funções de Similaridade

Função	Definição
Jaccard	$\frac{\lvert x \cap y \rvert}{\lvert x \cup y \rvert}$
Dice	$\frac{2\,\lvert x \cap y \rvert}{\lvert x \rvert + \lvert y \rvert}$
Cosseno	$\frac{\lvert x \cap y \rvert}{\sqrt{\lvert x \rvert \lvert y \rvert}}$

2.2 Junção por Similaridade

Junção por similaridade é a tarefa de utilizar funções de similaridade com o objetivo de encontrar pares de objetos que tenham um grau de semelhança acima de um limite estipulado. Uma das principais áreas para se utilizar junção por similaridade é no contexto de integração de dados. O termo junção por similaridade neste trabalho se refere à junção por similaridade baseada em conjuntos.

Definição 2 (Junção por Similaridade): Seja C uma coleção de conjuntos, $FS(x, y)$ uma função de similaridade e um *threshold* γ, o problema da junção por similaridade é encontrar todos os pares de conjuntos x, y, em que sua semelhança satisfaça o predicado $FS(x, y) \geq \gamma$ (RIBEIRO; HÄRDER, 2011).

Os principais algoritmos para junções por similaridade apresentam duas etapas: pares de conjuntos candidatos são primeiramente identificados e depois verificados usando a função de similaridade usada na condição da junção. A principal funcionalidade da primeira etapa é excluir pares de conjuntos que não poderão satisfazer o predicado de similaridade e, com isso, reduzir o espaço de comparações do algoritmo.

2.2.1 Modelos de Processamento

Existem dois modelos de processamento populares para junção de similaridade. O primeiro modelo, apresentado na Figura 1, representa os conjuntos em relações satisfazendo a Primeira Forma Normal, em que tuplas armazenam cada elemento do conjunto juntamente com o identificador único do conjunto. A junção por similaridade é expressa declarativamente usando uma linguagem como *SQL* e o processamento é realizado utilizando recursos nativos

do SGBD (ARASU; GANTI; KAUSHIK, 2006; CHAUDHURI; GANTI; KAUSHIK, 2006).

conjunto	token
x_1	ch
x_1	hr
x_1	ri
x_2	ri
x_1	is
x_2	is
x_1	st
x_2	st
x_1	ti

conjunto	token
x_2	ti
x_1	ia
x_2	ia
x_1	an
x_2	an
x_1	ne
x_2	ne
x_2	cr

Figura 1 Modelo de Processamento utilizando a Primeira Forma Normal

O segundo modelo de processamento, ilustrado na Figura 2, utiliza algoritmos especializados baseados em técnicas de Recuperação de Informação. Um índice invertido é construído para mapear *tokens* para listas de conjuntos que os contêm (BAYARDO; MA; SRIKANT, 2007; SARAWAGI; KIRPAL, 2004; XIAO et al., 2008). Para cada conjunto, o índice é examinado para geração de candidatos que serão posteriormente comparados com este conjunto na etapa de verificação.

Trabalhos anteriores mostraram que as abordagens baseadas em índices invertidos superam as abordagens baseadas em tecnologia relacional em termos de desempenho (BAYARDO; MA; SRIKANT, 2007). A representação de conjuntos através de listas invertidas oferece várias oportunidades de otimização e redução do espaço de comparação (as principais otimizações serão discutidas na Seção 2.2.2).

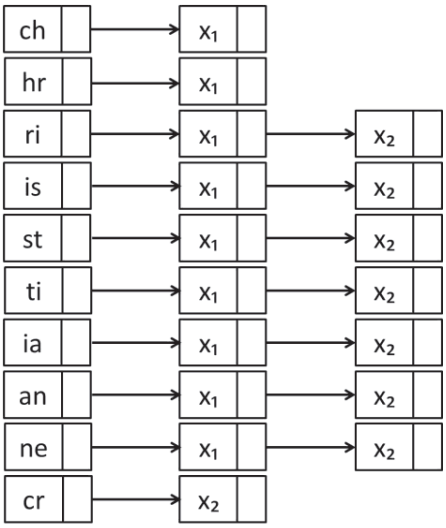


Figura 2 Modelo de Processamento utilizando Índice Invertido

No modelo de processamento baseado em tecnologia relacional, basta que exista um *token* em comum entre duas tuplas para que as mesmas sejam classificadas como um par candidato. Além disso, o processamento realizado durante a geração de pares candidatos não é reaproveitado durante a verificação. Como resultado, dois conjuntos que representam um par candidato devem ser verificados a partir do início para calcular a sua similaridade. Em contraste, abordagens baseadas em índices acumulam o valor de interseção durante a geração de candidatos.

2.2.2 Filtro de Candidatos

Devido ao fato de que cálculos de similaridade são onerosos computacionalmente, existem diversas otimizações que têm o objetivo de diminuir o tempo destas operações e, com isso, construir soluções com desempenho satisfatório. A seguir, as principais otimizações serão apresentadas.

2.2.2.1 Minoverlap

Esta otimização é baseada no fato de que podemos representar o predicado de similaridade como restrições de interseção, assim, o problema da junção por similaridade pode ser reduzido a um problema de interseção de conjuntos. Para isso, é utilizado o conceito de limite de interseção. Consideremos *x* e *y* como conjuntos de *tokens*, o limite de interseção entre *x* e *y* relativo à FS, denotada por *minoverlap* (*x, y*), é uma função que mapeia γ e o tamanho de *x* e *y* para um valor real, $FS(x, y) \geq \gamma$ se $|x \cap y| \geq$ *minoverlap* (*x, y*) (CHAUDHURI; GANTI; KAUSHIK, 2006).

Abaixo temos a função de *Jaccard* reescrita como uma função *minoverlap* (*x, y*).

$$\frac{|x \cap y|}{|x \cup y|} \geq \gamma$$

$$\frac{|x \cap y|}{|x| + |y| - |x \cap y|} \geq \gamma$$

$$|x \cap y| \geq \gamma|x| + \gamma|y| - \gamma\,|x \cap y|$$

$$|x \cap y| + \gamma|x \cap y| \geq \gamma(|x| + |y|)$$

$$(1 + \gamma)|x \cap y| \geq \gamma(|x| + |y|)$$

$$|x \cap y| \geq \frac{\gamma(|x| + |y|)}{1 + \gamma}$$

Desta forma, todos os pares em que a interseção não é menor que *minoverlap* (*x, y*) são retornados porque seguramente satisfazem a condição original da junção por similaridade. Temos na Tabela 2 os limites de interseção vinculados às funções de similaridade *Jaccard*, *Dice* e *Cosseno*. Uma importante observação é que, para todas as funções de similaridade, *minoverlap* aumenta uniformemente com um ou ambos os tamanhos de conjunto.

Tabela 2 *Minoverlap*

Função	*Minoverlap*(*x, y*)
Jaccard	$\frac{\gamma}{1+\gamma}(\|x\| + \|y\|)$
Dice	$\frac{\gamma(\|x\| + \|y\|)}{2}$
Cosseno	$\gamma\sqrt{(\|x\|\|y\|)}$

Exemplo 3: Consideremos as *strings* “Christiane” e “Cristiane” e seus respectivos conjuntos de *tokens*, *x = {ch, hr, ri, is, st,ti, ia, an, ne}* e *y = {cr, ri, is, st, ti, ia, an, ne}*. Temos que |*x*| = 9 e |*y*| = 8. Considerando *γ* = 0.80 e utilizando a função *minoverlap* (*x, y*) de *Jaccard*, temos que *minoverlap* (*x, y*) = ((0, 8/1, 8)(9+ 8)) = 8.

2.2.2.2 Minsize e Maxsize

Outra forma de se otimizar junção por similaridade é filtrando os candidatos pelo seu tamanho, pois conjuntos similares possuem tamanhos não muito discrepantes. Considerando *x* como um conjunto de *tokens*, o limite de tamanho de *x* em relação a FS são funções, denotadas por *minsize*(*x*) e

maxsize(*x*), que mapeiam γ e o tamanho de *x* para um valor real tal que, $\forall y$, se $FS(x, y) \geq \gamma$, então $minsize(x) \leq |y| \leq maxsize(x)$ (SARAWAGI; KIRPAL, 2004). Os limites de tamanho de conjunto das funções de similaridade para as funções *Jaccard*, *Dice* e *Cosseno* são definidos na Tabela 3.

Tabela 3 Minsize e Maxsize

Função	*minsize*(*x*)	*maxsize*(*x*)
Jaccard	$\gamma\|x\|$	$\frac{\|x\|}{\gamma}$
Dice	$\frac{\gamma\|x\|}{2-\gamma}$	$\frac{(2-\gamma)\|x\|}{\gamma}$
Cosseno	$\gamma^2\|x\|$	$\frac{\|x\|}{\gamma^2}$

Portanto, dado um conjunto *x*, pode-se ignorar todos os conjuntos que não estejam dentro do intervalo [*minsize*(*x*), *maxsize*(*x*)].

Exemplo 4: Considere x = {ch, hr, ri, is, st,ti, ia, an, ne}, |x| = 9 e $\gamma = 0.80$, temos que o tamanho dos conjuntos que podem ser candidatos a *x* são os que estão no intervalo de *minsize*(*x*) e *maxsize*(*x*), sendo *minsize*(*x*) = 9 × 0,8 = 7 e *maxsize*(*x*) = 9/0,8 = 12. Desta forma, se *Jaccard*(*x, y*) = γ, então $7 \leq |y| \leq 12$.

2.2.2.3 Prefix Filtering

Outra forma de se diminuir o número de candidatos é observando somente uma fração do conjunto original. Isso pode ser feito utilizando a técnica

de *prefix filtering*, primeiramente explorada por (SARAWAGI; KIRPAL, 2004) e posteriormente definida por Chaudhuri, Ganti e Kaushik (2006).

Fixando uma ordem O sobre os *tokens* em U, se $|x \cap y| \geq \alpha$, então os primeiros $|x| - \alpha + 1$ elementos de x e os primeiros $|y| - \alpha + 1$ elementos de y devem compartilhar pelo menos um elemento. Temos $pre\ f\ (x)$ como o prefixo do conjunto x, isto é $pre\ f\ (x)$ é o subconjunto de x que contém os primeiros $|pre\ f\ (x)|$ elementos de acordo com a ordenação O. Temos que para $\alpha = [minoverlap(x, y)]$, o conjunto de todos os pares (x, y) que compartilham um elemento comum no prefixo é um superconjunto de resultados corretos.

Exemplo 5: Considere os conjuntos x e y ordenados de forma crescente lexicograficamente, obtendo os conjuntos $x = \{an, ch, hr, ia, is, ne, ri, st,ti\}$ e $y = \{an, cr, ia, is, ne, ri, st, ti\}$. Utilizando $\alpha = 8$, temos $|pre\ f\ (x)| = 9 - 8 + 1 = 2$ e $|pre\ f\ (y)| = 8 - 8 + 1 = 1$. Assim, temos que os conjuntos $pre\ f\ (x) = \{an, ch\}$ e $pre\ f\ (y) = \{an\}$ e podemos afirmar que estes têm pelo menos um *token* em comum, então podem ser considerados como candidatos.

A ordenação mais comum é a baseada em pesos *IDF*, em que a ordenação é feita utilizando os pesos atribuídos aos *tokens*, sendo estes inversamente proporcionais a sua frequência. Desta forma *tokens* mais relevantes estarão no prefixo do conjunto, assim, as comparações entre candidatos será entre os elementos mais raros de cada conjunto.

Maxprefix

O tamanho do prefixo é determinado por $minoverlap(x, y)$, que varia de acordo com cada par. Tendo um conjunto x, a questão é como determinar $|pre\ f\ (x)|$ que seja suficiente para identificar todos os seus correspondentes (sem falsos negativos). Desta forma, temos que usar o maior prefixo em relação a todo possível conjunto y. Pelo fato de que o tamanho do prefixo varia

inversamente com *minoverlap*(*x, y*), |*pre f* (*x*)| é maior quando |*y*| é menor. O menor tamanho possível de *y*, de tal forma que a restrição pode ser satisfeita, é *minsize*(*x*).

A partir da noção de *prefix filtering*, podemos expandir para a ideia de *maxprefix*. Seja *x* um conjunto de *tokens*. O *maxprefix* de *x*, denotado por *maxpre f* (*x*), é o maior prefixo necessário para identificar ∀*y*, |*x*∩*y*| ≥ *minoverlap*(*x, y*). O tamanho de *maxprefix* é dado por |*maxpre f* (*x*)| = |*x*| − |*minsize*(*x*)| + 1 (CHAUDHURI; GANTI; KAUSHIK, 2006).

Exemplo 6: Considere x = {ch, hr, ri, is, st,ti, ia, an, ne}, |x| = 9 e γ = 0.80, temos que |maxpre f (x)| = 9 − 7 + 1 = 3, assim, maxpre f (x) = {ch, hr, ri}.

Midprefix

Outra otimização consiste em ordenar *C* em ordem crescente de tamanho de conjunto. Ao explorar essa ordenação, podemos garantir que *x* é somente comparado com *y*, se |*y*| ≥ |*x*|. Como resultado, o tamanho do prefixo de x pode ser reduzido: em vez de *maxpre f* (*x*), é obtido um prefixo mais curto usando *minoverlap*(*x, x*) para calcular o tamanho do prefixo. Tendo *x* como um conjunto de *tokens*, o *midprefix* de *x*, denotado por *midpre f* (*x*), é o menor prefixo necessário para identificar ∀|*y*| ≥ |*x*|, |*x* ∩ *y*| ≥ *minoverlap*(*x, y*). O tamanho do *midprefix* é obtido por: |*mid pre f ix*(*x*)| = |*x*| − |*minoverlap*(*x, x*)| + 1 (XIAO et al., 2008).

Exemplo 7: Considere x = {ch, hr, ri, is, st,ti, ia, an, ne}, |x| = 9 e γ = 0.80, temos que |*mid pre f ix*(*x*)| = 9 − 8 + 1 = 2, assim, *mid pre f ix*(*x*) = *{ch, hr}*.

Minprefix

Minprefix é uma generalização do conceito de *prefix-filtering*, onde o tamanho do prefixo de um conjunto *x* é definido a partir de outro conjunto de referência *y*. Assim, o valor de *minoverlap* aumenta com o tamanho dos conjuntos, e, consequentemente, o tamanho do prefixo diminui. Desta forma, *minprefix* será menor que *midprefix* quando o tamanho de *y* for maior que *x* (RIBEIRO; HÄRDER, 2011).

Seja o conjunto $x = t_1, \ldots, t_{|x|}$, onde os subscritos representam as posições dos *tokens* no conjunto, *rem*(*x, i*) denota o número de *tokens* seguintes ao *token* t_i em *x*, sendo *rem*(*x, i*) = |*x*| − *i*. Seja o conjunto *x*, o prefixo de *x* é denotado por *pre f* (*x*), *y* é um conjunto de referência. Então *pre f* (*x*) é um prefixo mínimo de *x* em relação ao *y*, indicado como *minpre f* (*x, i*), se e somente se 1 + *rem*(*x, j*) ≥ *minoverlap*(*y, x*) vale para todos os *tokens* $f_j \in pre\ f\ (x)$ (RIBEIRO; HÄRDER, 2011).

Exemplo 8: Considere x = {ch, hr, ri, is, st,ti, ia, an, ne}, |x| = 9 e um conjunto candidato y = {ma, ar, ri, ia, ac, ch, hr, ri, is, st,ti, in, na}, |y| = 13. Utilizando a função de Jaccard e com um limiar = 0,6, temos mid pre f (x) = 3 e minpre f (x, y) = 1.

Para entender mais detalhes, considere o segundo *token* de *x* que faz parte de *midprefix*(*x*), assim, temos 1 + *rem*(*x,* 2) = 8, que é menor que o valor de *minoverlap*(*x, y*) = 9, desta forma, não satisfazendo a restrição de *minprefix*.

2.2.3 O algoritmo mpjoin

O algoritmo *mpjoin*, elaborado por Ribeiro e Härder (2011), explora o conceito de *minprefix*. Esta abordagem tem como foco a diminuição do custo

computacional de geração de candidatos. *Min-prefix* garante que o tempo de geração de candidatos seja drasticamente reduzido.

Porém, há um aumento na carga de trabalho na fase de verificação, um efeito colateral desta abordagem. No entanto, esta fase foi otimizada interrompendo o cálculo de pares candidatos que não atendem à restrição de interseção o mais cedo possível, e também são armazenados os valores acumulados que são utilizados nos cálculos da verificação de similaridade.

Antes da execução do Algoritmo 1, temos uma etapa de pré-processamento, na qual os *tokens* são ordenados em ordem crescente de peso dentro de cada conjunto, e estes conjuntos são também ordenados de acordo com o seu tamanho. A primeira etapa deste algoritmo é concentrada na geração dos candidatos, a segunda, realiza a verificação de similaridade e a terceira e última realiza a indexação dos pares similares.

Inicialmente os índices invertidos ($I_1, I_2, \ldots I_{|U|}$) cada *token* e a lista de resultados *S* são inicializados. Inicia-se o *loop* que percorre todos os conjuntos x_1 (Linha 2). É criada a *hash M* que armazena as pontuações para cada conjunto (Linha 3). Início do *loop* que percorre todos os *tokens t* presentes em *maxprefix*(x_1) (Linha 4). Remove-se todos os conjuntos x_2 em que $|x_2|<$ *minsize*(x_1) (Linha 5). Para todos osd conjuntos remanescentes em I_t é iniciado o *loop* (Linha 6). Na chave x_2 da *hash* M, adiciona-se ao escore uma unidade (Linha 7). Verifica-se se o *token* não existe em *minprefix*(x_2, x_1), estes não são suficientes sozinhos para eleger um x_1 qualquer, e podem ser removidos com segurança (Linha 8). Remove-se o conjunto x_2 de I_t (Linha 9). Subtrai-se o tamanho de *t* no escore de x_2 presente na tabela *hash M* (Linha 10). É realizada a verificação, que calcula a similaridade entre x_1 e os demais candidatos gerados na etapa anterior, os pares que satisfazem o predicado de similaridade são adicionados à lista de resultados *S* (Linha 13). Inicia-se o *loop*, em que é realizada a indexação dos *tokens* presentes em *midprefix*(x_1) (Linha 14). Um

ponteiro para o conjunto x_1 é adicionado na lista invertida I_t (Linha 15). Finalmente, todos os pares que satisfazem o predicado de similaridade são retornados (Linha 19).

Algorithm 1 Algoritmo MPJoin

Entrada: Uma coleção de conjuntos C, e um threshold γ

Saída: Um conjunto S contendo todos pares $(x1, x2)$ em que $Sim(x1, x2) \geq \gamma$

$I_1, I_2, \ldots I_{|U|} \leftarrow \varnothing, S \leftarrow \varnothing$

```
1:  function MPJOIN(colecao_de_conjuntos C, threshold τ)
2:      for x1 ∈ C do
3:          M(id, ea) ← ∅
4:          for t ∈ maxpref(x1) do
5:              Remove todos x2 ∈ It em que |x2| < minsize(x1)
6:              for x2 ∈ It do
7:                  M(x2) ← M(x2).ea + 1
8:                  if t ∉ minprefix(x2, x1) then
9:                      Remove x2 de It
10:                     M(x2).ea − |t|
11:                 end if
12:             end for
13:             S ← S ∪ Verificao(x1, M, γ)
14:             for t ∈ midPref(x1) do
15:                 It ← It ∪ +1
16:             end for
17:         end for
18:     end for
19:     Retorne S
20: end function
```

2.3 Predição

Predição consiste em construir um modelo através de uma função com o objetivo de classificar uma variável alvo. Existem dois tipos de métodos para esse modelo: classificação, na qual são utilizadas variáveis alvo discretas e regressão, em que são usadas variáveis alvo contínuas. Pela natureza dos dados empregados, foi utilizada somente a técnica de regressão, explicada adiante.

2.3.1 Regressão

A análise de regressão é uma metodologia estatística mais frequentemente usada para a previsão numérica. Abrange também a identificação de tendências de distribuição com base nos dados disponíveis (HAN; KAMBER; PEI, 2011).

Quando o resultado ou classe é numérico e todos os atributos são numéricos, a regressão linear é uma técnica natural a considerar. Este é um método básico em estatística. A ideia é expressar a classe como uma combinação dos atributos, com pesos pré-determinados: $c = w_0 + w_1 a_1 + w_2 a_2 + ... + w_k a_k$, em que c é a classe; $a_1, a_2, ... a_k$ são os valores dos atributos e $w_0, w_1, ... w_k$ são os pesos (WITTEN; FRANK; HALL, 2011).

2.3.1.1 Árvores de Regressão

A abordagem "dividir para conquistar" para o problema de aprendizagem a partir de um conjunto de instâncias remete ao estilo de representação chamado de **árvore de decisão**. Nós, em uma árvore de decisão testam um atributo específico, normalmente através de constantes, e direcionam para outros nós. Isso é feito até se encontrar nós folha, que geram uma

classificação para a instância (WITTEN; FRANK; HALL, 2011). A Figura 3 mostra um exemplo de árvore de decisão.

Árvores que são usadas para previsão numérica são semelhantes a árvores de decisões comuns, exceto que, em cada folha elas armazenam ou um valor de classe que representa o valor médio de ocorrências que atingem a folha, no caso em que as árvores chamadas de *árvores de regressão*, ou um modelo de regressão linear que prediz o valor da classe das instâncias que atingem a folha, no caso das chamadas *árvores modelo* (WITTEN; FRANK; HALL, 2011).

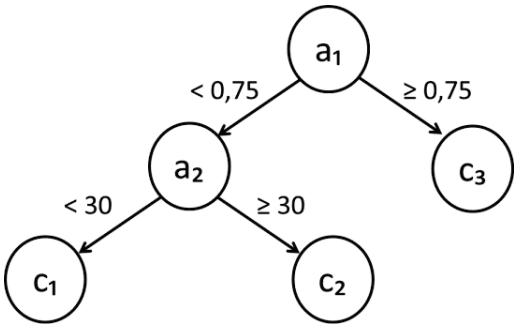


Figura 3 Árvore de Decisão

Quando uma árvore modelo é usada para prever o valor de uma instância de teste, a árvore é aprofundada até uma folha, usando os valores dos atributos da instância para tomar decisões de roteamento em cada nó. A folha irá conter um modelo linear baseado em alguns dos valores de atributos, e este é avaliado para a instância de teste para se obter uma previsão de valor (WITTEN; FRANK; HALL, 2011).

M5

As árvores de decisão M5 foram introduzidas por Quinlan (1992), e tem o objetivo de construir modelos baseados em árvores de decisão. Os modelos

são similares aos gerados por outros sistemas de indução de árvore, mas, ao invés de possuir valores em seus nós folha, os modelos gerados por tal técnica possuem modelos lineares em cada nó folha. Este tipo de modelo pode ser utilizado em diversas tarefas de aprendizagem e tem a capacidade de lidar com dados de alta dimensionalidade. Para a criação de árvores M5, tendo uma coleção *T* de instâncias de treino, cada instância é especificada pelos valores de seus atributos e tem um valor associado à variável alvo. O objetivo é construir um modelo que relacione os valores-alvo das instâncias de treinamento aos valores dos outros atributos. O valor do modelo será medido pela precisão com que ele prevê os valores-alvo de instâncias não conhecidas.

Árvores modelo são construídas pelo método dividir para conquistar. O conjunto T é associado a uma folha, ou são escolhidos testes que dividem *T* em subconjuntos correspondentes aos resultados destes testes. Este processo é aplicado de forma recursiva para os subconjuntos.

Esta divisão, normalmente, gera muitas estruturas que podem ser podadas. Essas podas têm o objetivo de, por exemplo, substituir uma subárvore por uma folha.

O primeiro passo na construção de uma árvore modelo é o cálculo de desvio padrão dos valores-alvo das instâncias de *T*. A menos que *T* contenha poucas instâncias, ou seus valores-alvo variem muito pouco, T é dividido sobre os resultados de um teste. Cada teste potencial é avaliado pela determinação dos subconjuntos de instâncias associados com cada resultado, temos T_i como o subconjunto de instâncias que se enquadram no resultado *i* do teste em potencial. É calculado o desvio padrão sobre os valores-alvo de T_i como uma medida de erro. A redução do erro esperado como resultado deste teste é calculada a partir da Equação 3.

(3)

$$\Delta error = sd(T) - \sum \frac{|T_i|}{T} x\ sd(T_i)$$

Após examinar todos os testes, o algoritmo M5 escolhe o teste que maximize a redução do erro esperado. Após a construção da árvore, temos a etapa de poda, em que é reduzida a quantidade de nós da árvore obtida e em seguida temos a fase de suavização, que reduz a diferença dos valores preditos entre os nós-folhas.

2.3.1.2 Métodos de *Ensemble*

Métodos de *Ensemble* são maneiras de realizar combinação de modelos. Eles combinam uma série de modelos com o objetivo de criar um modelo composto melhorado (HAN; KAMBER, 2006).

Combinar vários modelos ajuda quando estes modelos são significativamente diferentes uns dos outros e cada um trata de uma porcentagem razoável dos dados corretamente. O método de *boosting* para combinar vários modelos explora essa percepção (WITTEN; FRANK; HALL, 2011).

Boosting é um procedimento iterativo usado para alterar adaptativamente a distribuição de exemplos de treinamento de modo que os classificadores enfoquem em exemplos difíceis de classificar (TAN et al., 2009). Utiliza votação (para a classificação) ou média (para previsão numérica) para combinar a saída de modelos individuais. Nesta técnica, cada novo modelo é influenciado pelo desempenho daqueles construídos anteriormente. E incentiva novos modelos para se tornarem especialistas em instâncias anteriormente manipuladas incorretamente, atribuindo maior peso para essas instâncias (WITTEN; FRANK; HALL, 2011).

Para dados numéricos, a técnica de *boosting* é denominada regressão aditiva (*additive regression*). Esta é reformulada como um algoritmo guloso para a montagem de um modelo aditivo, que gera previsões somando contribuições obtidas a partir de outros modelos. A maioria dos algoritmos de aprendizagem para modelos aditivos não constroem modelos base de forma independente, mas garantem que eles se complementem um ao outro e formem um conjunto de modelos de base que otimizem o desempenho de previsão de acordo com algum critério especificado (WITTEN; FRANK; HALL, 2011).

2.3.2 Medidas de Erro

Resultados de regressão retornam valores contínuos, assim, é difícil dizer exatamente se o valor previsto está correto. Em vez de se concentrar em verificar se o valor exato foi encontrado, deve-se analisar o quão longe o valor previsto é do valor real conhecido (HAN; KAMBER, 2006).

É de conhecimento que a seleção de medidas de erro tem efeitos importantes sobre as conclusões sobre quais métodos são mais acurados. Desta forma, neste trabalho, foram calculados o Erro Absoluto Médio (EAM), Erro Relativo Médio (ERM) e Erro Absoluto Relativo (EAR), que são apresentados nas Equações 4, 5 e 6 respectivamente.

$$EAM = \frac{\sum_{i=1}^{N} |p_i - e_i|}{N} \quad (4)$$

$$EAR = \frac{\sum_{i=1}^{N} |p_i - e_i|}{\sum_{i=1}^{N} |e_i - e_m|} \quad (5)$$

$$ERM = \frac{1}{N} \text{ x } \sum_{i=1}^{N} \frac{|p_i - e_i|}{e_i} \quad (6)$$

Em que, p_i é o valor previsto, e_i é o valor estimado para a instância i, e_m é a média dos valores estimados e N é o total de instâncias.

Erro absoluto médio é a média dos erros individuais, sem levar em consideração os sinais. Não tende a exagerar o efeito de *outliers*, instâncias que o erro de predição é significativamente maior que outras. Os tamanhos de erro são tratados de maneira uniforme de acordo com sua magnitude (WITTEN; FRANK; HALL, 2011). Neste trabalho, este resultado apresenta o erro médio em segundos da predição.

O Erro Relativo Médio mede o erro como a diferença proporcional entre a saída desejada e a saída obtida, em que valores mais baixos significam maior precisão do modelo.

Erro Absoluto Relativo é definido como a soma das diferenças entre o valor predito e o valor estimado, dividido pela soma das diferenças dos valores estimados e valores estimados médios. É uma medida útil, especialmente ao fazer comparações entre conjuntos de dados relativamente pequenos, em que os valores- alvo se diferem substancialmente (ARMSTRONG; COLLOPY, 1992).

2.3.3 Avaliação

Para prever o desempenho de um classificador ou modelo de regressão em novos dados, é preciso avaliar a sua taxa de erro em um conjunto de dados que não desempenharam nenhum papel na formação do classificador. Este conjunto de dados independente é chamado conjunto de teste. Assumimos que tanto os dados de treinamento e os dados de teste são amostras representativas do problema subjacente (WITTEN; FRANK; HALL, 2011).

O uso da técnica de dividir os dados entre conjuntos de treino e conjuntos de teste, mesmo que aumente o tempo total de computação, é muito útil na seleção de um bom modelo (HAN; KAMBER, 2006). A taxa de erro calculada a partir do conjunto de teste também pode ser usada para comparar o desempenho relativo de diferentes classificadores no mesmo domínio (TAN et al., 2009). Como exemplo de técnicas comuns que realizam essa tarefa temos: *holdout*, subamostragem aleatória, validação cruzada (*cross-validation*) e *bootstrap*.

Holdout

No método de *holdout*, os dados são divididos aleatoriamente em dois grupos independentes, um conjunto de treinamento e um conjunto de teste. Normalmente, dois terços dos dados são alocados para o conjunto de treinamento, e o restante é atribuído ao conjunto de teste. O conjunto de treinamento é usado para derivar o modelo, cuja precisão é estimada através do conjunto de teste. A estimativa é pessimista porque apenas uma porção dos dados iniciais é utilizada para derivar o modelo (HAN; KAMBER, 2006).

De acordo com Tan et al. (2009), este método possui diversas limitações. Uma das limitações é que o modelo é construído com menos exemplos, pois grande parte é colocada no teste. Outro problema é que o modelo

pode ser altamente dependente da composição dos dados de treinamento e teste, sendo que, quanto menor o tamanho do conjunto de treinamento, maior a variância do modelo. No entanto, com o conjunto de treinamento muito grande, a precisão é calculada a partir de um subconjunto de teste muito pequeno, tendo assim pouca confiança. E também há o problema de termos classes representadas em excesso em um subconjunto, podendo não ser utilizados na criação do modelo.

Subamostragem aleatória

Subamostragem aleatória é uma variação do método *holdout* em que este é repetido k vezes. A estimativa da acurácia é obtida pela média de acurácias obtidas em cada iteração, no caso de classificação e média de taxas de erros em regressão (HAN; KAMBER, 2006). Este método possui alguns associados ao método *holdout*, como não estender ao máximo o uso de dados para treinamento. Também não possui um meio de controlar o número de vezes que uma instância é utilizada para treinamento e teste (TAN et al., 2009).

Validação Cruzada

No método de validação-cruzada, os dados iniciais são divididos aleatoriamente em k subconjuntos mutuamente exclusivos (*folds*), $D_1, D_2, \ldots, D_k$ cada um com tamanho aproximadamente igual. Treinamento e testes são realizados k vezes. Na iteração i, a partição D_i é reservada como o conjunto de teste, e as demais partições são usadas coletivamente para treinar o modelo. Isto é, na primeira iteração, os subconjuntos $D_2, \ldots, D_k$ servem coletivamente como o conjunto de treino, a fim de obter um primeiro modelo, que é testado com o conjunto D_1, a segunda iteração são treinados os subconjuntos $D_1, D_3, \ldots, D_k$ e testado em D_2, e assim por diante. Desta forma, cada amostra é utilizada o

mesmo número de vezes para o treinamento e exatamente uma vez para o teste (HAN; KAMBER, 2006).

Essa abordagem tem a vantagem de permitir utilizar grande quantidade de dados para treinamento. Além disso, os conjuntos de dados são mutuamente exclusivos e cobrem todo o conjunto de dados. Porém, tem a desvantagem de executar o procedimento *k* vezes, gerando uma grande carga computacional (TAN et al., 2009).

2.4 Trabalhos Relacionados

Grande quantidade de trabalhos está sendo realizada na tentativa de se prever o tempo de execuções de consultas *SQL*. Estes trabalhos utilizam técnicas de aprendizagem de máquina para descobrir as relações entre os tempos de execuções das consultas e suas *features*. Estes trabalhos podem ser divididos entre os que realizam modelos de regressão e os de modelos analíticos (a grande maioria se concentra nos modelos de regressão). A seguir são listados os trabalhos que usam modelos de regressão e em seguida o trabalho que utiliza modelos analíticos.

Os trabalhos de Akdere et al. (2012), Duggan et al. (2011), Ganapathi et al. (2009) e Li et al. (2012) utilizam modelos de regressão, que usam estimativas de cardinalidade e custo de operadores baseados nos valores obtidos pelo otimizador dos sistemas de banco de dados. Após o treinamento, o modelo extrai um conjunto de *features* que descreve a consulta e gera uma estimativa de um recurso específico (como tempo de execução, consumo de *CPU* e consumo de *I/O*). Como exemplo de *features* para estes modelos temos número de tuplas de entrada, número de tuplas de saída do operador, média de largura das tuplas de entrada e saída do operador.

Ganapathi et al. (2009) desenvolveram um sistema que usa a aprendizagem de máquina para prever com precisão métricas de performance de consultas de banco de dados cujos tempos de execução variam de milissegundos a horas. Foram utilizadas somente informações colhidas antes do início da execução das consultas como comandos *SQL* e planos de consultas produzidas pelo otimizador de consultas. Os dados de treino e teste foram categorizados para distinguir as consultas de acordo com seus tempos de execução. Como método de predição, foi utilizado KCCA (*Kernel Canonical Correlation Analysis*). Foi possível prever em consultas individuais o tempo restante de execução com 20% do tempo real de 85% das consultas de teste.

Duggan et al. (2011) apresentaram uma abordagem de modelagem para estimar o impacto da concorrência sobre o desempenho de consultas para cargas de trabalho analíticas. A solução baseou-se na análise do comportamento de consultas de forma isolada, interações das consultas em pares e técnicas de amostragem para prever a contenção de recursos em várias combinações de consultas e os níveis de concorrência.

O trabalho de Akdere et al. (2012), assim como o de Ganapathi et al. (2009), utilizaram estimativas de tempos de execuções e recursos no uso de consultas *SQL*. Foi proposta uma abordagem híbrida que compõem modelos no nível de planos de consultas e de nível de operadores. Foram utilizadas, como técnicas de aprendizagem de máquina, o SVM (*Support Vector Machine*) e o KCCA.

Li et al. (2012) combinaram conhecimentos de processamento de consultas de banco de dados com modelos estatísticos. Foi modelado o uso de recursos a nível de operadores básicos dos SGBDs, para cada um deles foram definidas funções de escalonamento específicas, o que foi possível a partir do pré-conhecimento do comportamento assintótico destes. Foi utilizado *Multiple Additive Regression-Trees* (MART) como modelo básico de aprendizagem de

máquina. Desta forma, este trabalho conseguiu aumentar significativamente a precisão das estimativas e a capacidade de estimar o uso de recursos de planos arbitrários, mesmo quando muito diferentes das instâncias de treinamento.

Modelos analíticos, que realizam ajustes no modelo de custos dos sistemas gerenciadores de banco de dados, foram utilizados na abordagem de Wu et al. (2013). Para isso, calibrou-se as constantes dos modelos de custo do otimizador mantendo as demais isoladas. Calibrando corretamente estas constantes e tendo boas estimativas de cardinalidade, foi possível uma boa previsão de tempo de execução de consultas. Para se melhorar as estimativas, utilizou-se técnicas, como amostragem, que melhoram as estimativas de cardinalidade para o plano escolhido. Exemplos de constantes utilizadas nessa abordagem são: custo de *I/O* para acesso sequencial à página, custo de *I/O* para acesso randômico a uma página, custo de *CPU* para processar uma tupla e custo de *CPU* para processar uma tupla utilizando índices.

A predição de desempenho de consultas usando métodos de aprendizagem estatística também estão sendo realizadas em outros contextos, como: estimação de custo XML (ZHANG et al., 2005) e fusão de banco de dados (AHMAD; BOWMAN, 2011). O trabalho de Mogrovejo et al. (2013) é o que mais se aproxima do cenário deste trabalho, pois são exploradas técnicas estatísticas para a predição de processamento de junções, diferenciando principalmente na utilização do paradigma *MapReduce*. O trabalho de Vernica, Carey e Li (2010) propõe um algoritmo de três fases, incluindo o pré-processamento, geração de candidatos e a junção de similaridade baseada em conjunto que é executada durante a última função de redução. Neste contexto, é possível combinar uma coleção de modelos para diferentes fases do *MapReduce* com o objetivo de construir uma solução de predição para a execução completa do algoritmo em *MapReduce*, de maneira similar à abordagem de Akdere et al. (2012): a técnica descrita neste trabalho é usada para modelar o algoritmo de

similaridade, ao passo que técnicas específicas para *MapReduce* (MOGROVEJO et al., 2013) são usadas para modelar as tarefas restantes.

Existe uma grande quantidade de trabalhos sobre junção por similaridade (BAYARDO; MA; SRIKANT, 2007; CHAUDHURI; GANTI; KAUSHIK, 2006; RIBEIRO; HÄRDER, 2011; SARAWAGI; KIRPAL, 2004; XIAO et al., 2008, 2009). Aspectos mais relevantes para o nosso trabalho já foram discutidos em profundidade na Sessão 2.2.

3 ARCABOUÇO PARA PREDIÇÃO DE TEMPO DE EXECUÇÃO PARA JUNÇÃO POR SIMILARIDADE

Nesta seção, é apresentada uma visão geral sobre a abordagem do presente trabalho. A Figura 4 ilustra as principais operações que abrangem o arcabouço de predição de tempo para junção de similaridade. Foram adotadas técnicas de aprendizagem estatísticas para construir um modelo de previsão de desempenho. Como nos trabalhos de Akdere et al. (2012), Ganapathi et al. (2009) e Li et al. (2012), o modelo é construído *offline* utilizando os dados de treinamento e implantado *online* para obter estimativas de tempo de execução.

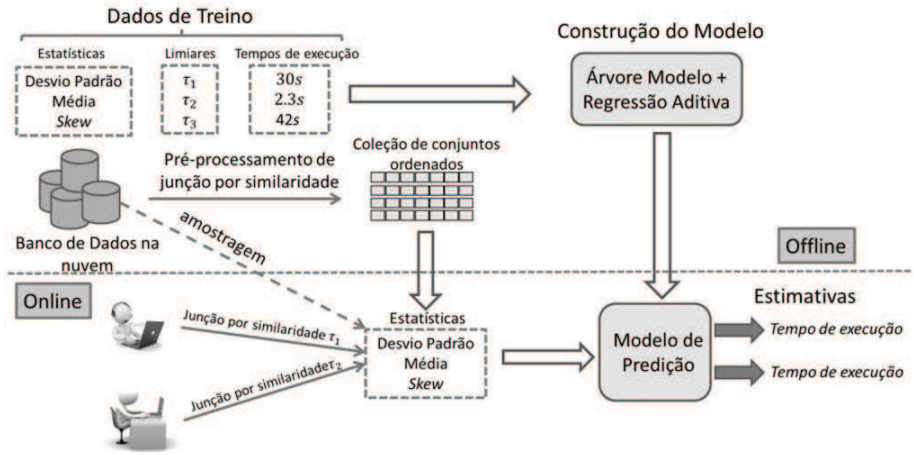


Figura 4 Arcabouço para predição de tempo para junções de similaridade

Na fase *offline*, utiliza-se principalmente as estatísticas coletadas a partir de dados semissintéticos como recurso de treinamento. Foram gerados novos dados a partir de dados do mundo real para aumentar substancialmente a cobertura da distribuição de dados e limiares (ver Seção 4 para mais detalhes).

O limiar de similaridade é a única *feature* extraída da junção por similaridade. Este aspecto contrasta com abordagens em consultas *SQL*, onde

um espaço rico de *features* podem ser extraídas a partir de planos de consulta. Juntamente com a variável-alvo, ou seja, valores de tempo de execução de junções por similaridade, estatísticas e limiares são usados como entrada para uma técnica de aprendizagem estatística baseada em árvore modelo e utilizando regressão aditiva (QUINLAN, 1992; WITTEN; FRANK; HALL, 2011).

Na fase *online*, o modelo de previsão construído *offline* é implantado para estimar o tempo de execução de consultas recebidas, antes de começar a execução. Para isso, é necessário extrair estatísticas do mesmo tipo das usadas nos dados de treinamento. Podemos obter essas estatísticas usando amostragem *offline*, portanto evitando a alta sobrecarga de execução relativa à amostragem *online*. Uma alternativa é coletar estatísticas durante a fase de pré-processamento do algoritmo de junção por similaridade, que geralmente é realizado *offline* de qualquer forma.

Serviços de integração de dados baseados em nuvem como o *Google Fusion Tables* (GONZALEZ et al., 2010) são exemplos proeminentes de sistemas que podem se beneficiar desta estrutura para predição de execução de junção por similaridade. Mas, um exemplo ainda mais atraente é, talvez, o suporte baseado em nuvem para exploração de análise de dados (CHAUDHURI, 2012). Neste cenário, em vez de ser usado em atividades de integração de dados *"single-shot"*, operações de junção por similaridade podem ser repetidamente invocadas por diferentes usuários com diferentes limiares durante o processo de exploração de dados.

4 METODOLOGIA

Neste capítulo, são apresentadas informações referentes à metodologia deste trabalho. Primeiramente, este trabalho é classificado em relação ao tipo de pesquisa. Em seguida são apresentados os *hardwares* e *softwares* utilizados.

São apresentadas as metodologias de geração de conjuntos de dados para execução de junções por similaridade e geração de dados para criação e validação de modelos de predição. Também são demonstradas as informações acerca da escolha das *features* utilizadas, bem como a criação e validação dos modelos de predição.

4.1 Tipo de pesquisa

Esta pesquisa classifica-se como tecnológica, exploratória e experimental, fundamentada em *design science*. Conforme Vaishnavi e Kuechler (2004), *design science* tem o objetivo de gerar novo conhecimento ao design (de soluções de problemas do mundo real) e prover ferramentas necessárias para ações adequadas de domínio dos profissionais em seus campos de atuação para resolução de problemas.

4.2 *Hardware* e *software* utilizados

Foram utilizadas duas arquiteturas de *hardware* diferentes em nossos experimentos. Foi utilizado um servidor com processador Intel Xeon E3-1270 3,4 GHz, 8 MB de cache, e 8 GB de memória RAM, nomeamos esta arquitetura de *servidor*. A outra máquina utilizada possui como configuração um processador Intel i5, 2.9 GHz a 3.6 GHz, 6 MB de cache e 4 GB de memória RAM, que foi nomeada de *cliente*.

Para as implementações do algoritmo *mpjoin* e geradores de dados sintéticos e semissintéticos, foi utilizada a linguagem de programação Java(JDK 7 (Oracle) 1.7.0). O *software* utilizado para prover algoritmos de aprendizagem de máquina foi o Weka (*Waikato Environment for Knowledge Analysis*), versão 3.7.9.

4.3 Geração de conjuntos de dados para execução de junções por similaridade

Para construir modelos de previsão precisos, uma coleção de dados de treinamento que cubra uma parte representativa do espaço de entrada é fundamental. Esta tarefa é um desafio no contexto deste problema, porque o espaço de possíveis distribuições de dados e valores de limiar é muito grande. Desta forma, temos que lidar com este problema recorrendo a bases de dados sintéticas e semissintéticas (são gerados novos dados a partir de banco de dados reais).

Inicialmente, neste trabalho foram criados dados sintéticos. O gerador de dados sintéticos produz diretamente uma coleção de conjuntos e permite a especificação de diversos parâmetros como distribuição do tamanho dos conjuntos e frequências dos elementos. Com estes dados foi possível simular e visualizar o comportamento dos dados de entrada para junção de similaridade, ajudando na tomada de decisões iniciais deste projeto.

Porém, com o objetivo de se utilizar dados mais próximos da realidade, toda a etapa de geração de conjuntos de dados foi realizada sobre um ambiente de geração de dados semissintéticos, do qual é possível extrair informações de uma base de dados estruturada em XML, criando amostras com um número determinado de conjuntos de *tokens*.

Neste trabalho, utilizamos como base de dados reais o DBLP contendo informações sobre publicações de ciência da computação (dblp.uni-trier.de/xml) e o IMDB contendo informações sobre filmes (www.imdb.com). Na base de dados do DBLP, foram selecionados aleatoriamente e concatenados título da publicação e nomes de autores para formar *strings* de entrada para o algoritmo *mpjoin*. E no caso do IMDB, foram selecionados randomicamente títulos de filmes e nomes de atrizes ou atores. A quantidade de títulos de publicação, nomes de autores, títulos de filmes e nomes de atrizes ou atores nas *strings* foi determinada de acordo com o tipo de experimento realizado (ao final desta seção é detalhado como isso foi feito). Essas *strings* resultantes foram convertidas para letras maiúsculas e eliminou-se os espaços em branco. A Figura 5 apresenta exemplo deste processamento.

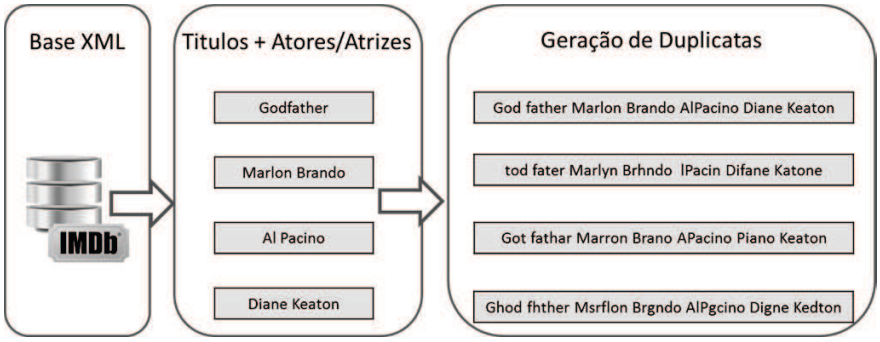


Figura 5 Criação dos conjuntos de dados gerados a partir do IMDB

Além disso, foram geradas duplicatas difusas ao longo dos conjuntos de dados. Duplicatas difusas são geradas através da criação de cópias exatas da *string* original e, em seguida, executa-se de 1 a 5 transformações, tais como inserção, exclusão ou substituição de caracteres. Os valores da coluna "N^{o} de Duplicatas" da Tabela 4 referem-se ao número de cópias, incluindo a *string* original, portanto, o valor 1 resulta em um conjunto de dados contendo apenas

strings originais, enquanto 10 resulta em 10% de *strings* originais e 90% de duplicatas. A presença de duplicatas conduz a conjuntos semelhantes que, por sua vez, aumentam o tamanho da saída, bem como o tempo de execução do algoritmo *mpjoin*. A quantidade total de *strings* por conjunto de dados foi determinada de acordo com o tipo de experimento realizado (mais detalhes no final desta seção).

Após a criação das *strings*, foram criados os conjuntos a partir destas, utilizando como métodos de geração de *tokens q-grams* com *q* igual a 2, 3, 4 ou 5. O objetivo de se definir diferentes métodos é pelo fato da distribuição de frequência ser proporcional ao tamanho do *token*.

Em seguida, foi mapeado cada *token* para um valor de *hash*. Em seguida, ordenou-se os *tokens* dentro do conjunto de acordo com a sua frequência e os conjuntos foram armazenados em ordem ascendente de tamanho. Finalmente, uma versão com conjuntos ponderados foi gerada a partir de cada conjunto de dados. A Figura 6 apresenta a segunda etapa do processamento para criação dos conjuntos de dados.

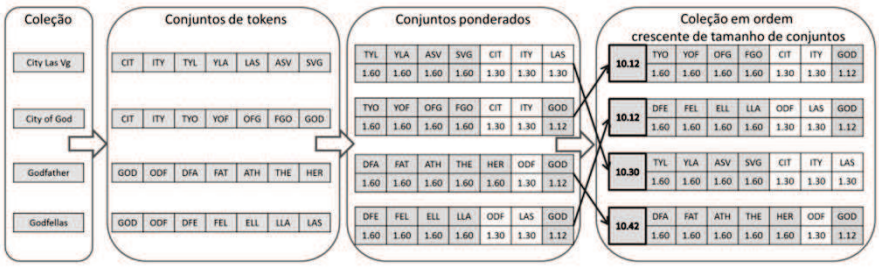


Figura 6 Criação dos conjuntos de dados gerados a partir do IMDB

Foram executadas 25 gerações de dados, com o objetivo de se obter um total de 400 conjuntos de dados, os parâmetros estão apresentados na Tabela 4.

Tabela 4 Parâmetros de geração de conjuntos de dados

Tokenização	Nº de Duplicatas	Execução
2-grams	1	1
3-grams	2	.
4-grams	5	.
5-grams	10	25

Para possibilitar diversas formas de validação dos modelos gerados, foram criados novos conjuntos com variações durante a criação destes. Segue nomenclatura e definição desses conjuntos de dados:

a) *Base*: Conjuntos de dados nesta configuração são compostos por 50.000 a *strings* (no qual o valor é escolhido randomicamente nesta faixa). Suas *strings* contêm nenhum ou um título concatenado com 0 a 10 nomes de autores/atores ou atrizes para as bases de dados DBLP/IMDB. A presença de título e quantidade de autores/atores ou atrizes também são determinados randomicamente. Para essa configuração, temos os seguintes conjuntos de dados:
   - DBLP não ponderado;
   - DBLP ponderado;
   - IMDB não ponderado;
   - IMDB ponderado.

b) *Hardware*: Estes conjuntos de dados possuem a mesma configuração do anterior, diferenciando pelo local de execução, que neste caso foi feito na plataforma denominada *cliente* (os demais conjuntos de dados foram feitos na plataforma *servidor*). Para essa configuração, temos os seguintes conjuntos de dados:

- DBLP não ponderado;
- DBLP ponderado;
- IMDB não ponderado;
- IMDB ponderado.

c) *Conjuntos Maiores*: Este conjunto de dados se diferencia do *base* pela composição dos seus conjuntos, que neste caso possui entre 150.000 a 200.000 *strings*. Para essa configuração, temos os seguintes conjuntos de dados:
   - DBLP ponderado;
   - IMDB não ponderado;
   - IMDB ponderado.

d) *Strings Maiores*: Nesta configuração, a variação para os conjuntos do *base* foi no tamanho das *strings* contidas nestes. As *strings* contêm nenhum ou um título concatenado com 10 a 20 nomes de autores/atores ou atrizes para as bases de dados DBLP/IMDB. A presença de título e quantidade de autores/atores ou atrizes também é determinado randomicamente. Temos os seguintes conjuntos de dados para essa configuração:
   - IMDB não ponderado;
   - IMDB ponderado.

4.4 Dados para criação e validação de modelos de predição

Para a geração de dados para treinamento de modelos, foram executados o algoritmo *mpjoin* com diferentes valores de limiar (0.5, 0.6, 0.7, 0.8 e 0.9)

sobre os conjuntos de dados criados conforme explicado na Seção 4.3 e foi recolhido a média de tempo de 3 execuções.

Os dados de treinamento foram contemplados com 2.000 registros de execuções de junção por similaridade para cada variante de dados não ponderado e ponderado, sendo assim, temos os seguintes dados de treinamento/teste:

a) Base:
   - DBLP não ponderado;
   - DBLP ponderado;
   - IMDB não ponderado;
   - IMDB ponderado.

b) *Hardware*:
   - DBLP não ponderado;
   - DBLP ponderado;
   - IMDB não ponderado;
   - IMDB ponderado.

c) Conjuntos Maiores:
   - DBLP ponderado;
   - IMDB não ponderado;
   - IMDB ponderado.

d) Strings Maiores:
   - IMDB não ponderado;
   - IMDB ponderado.

4.5 *Features* utilizadas

O colhimento das *features*, necessárias para criação de modelos de predição, foi realizado antes da execução do algoritmo *mpjoin*, pois na etapa de criação de conjuntos ordenados é realizada uma passagem completa sobre o conjunto. Desta forma, não houve intervenção no tempo final de execução do algoritmo.

A facilidade de coleta e extração de informações necessárias também foi um critério considerado. Por esta razão, não foram incluídas *features* de estimativas de tamanho do resultado e eficácia de filtragem, que exigem técnicas sofisticadas recentemente publicadas na literatura (LEE; NG; SHIM, 2009, 2011; LU et al., 2013).

A seguir são descritas as *features* que foram utilizadas como entrada para o modelo e fornecer a lógica por trás de sua escolha. A definição destas foi guiada pelo conhecimento de técnicas de otimização e detalhes do algoritmo descrito na Seção 2.2.3. Os principais conceitos acerca desta escolha são:

a) Valores de limiar: Há uma clara correlação entre valores de limiar e desempenho de junções por similaridade. Invariavelmente, o tempo diminui ou aumenta seguindo altos ou baixos valores de limiar, respectivamente. A explicação é de que os valores de limiar mais elevados implicam limites maiores de sobreposição e prefixos mais curtos. Como resultado, um número maior de pares candidatos são descartados em fases anteriores de processamento ou mesmo não são considerados;

b) Distribuição da Freqüência dos *Tokens*: A frequência dos *tokens* no prefixo determina o número de candidatos gerados para um determinado conjunto. A principal motivação para a adoção de uma ordenação global baseada na frequência do conjunto é colocar *tokens* raros nos prefixos. Com *tokens* de baixa frequência no prefixo, listas invertidas são mais curtas e há muito menos sobreposição de prefixos entre conjuntos diferentes, diminuindo assim o número de candidatos gerados.
c) Tamanho do conjunto e distribuição de peso: Para conjuntos não ponderados, conjuntos maiores traduzem prefixos maiores e, portanto, mais *tokens* são usados para examinar o índice na fase de geração de candidatos. Para conjuntos ponderados, o tamanho do prefixo é ditado pelo conjunto de pesos. A carga de trabalho na fase de verificação também depende do tamanho e/ou peso do conjunto.

Com base nas considerações acima, foram projetadas *features* visando a caracterização dos valores de limiar, distribuições de frequência de *tokens* e tamanho/peso de *tokens*. O limiar pode diretamente representar uma *feature*.

Para as distribuições de dados, utilizou-se medidas gerais com base nos momentos de distribuição. Como medida de tendência central, na qual gera-se um valor central da distribuição dos dados, foram utilizadas média, média geométrica e tri-média. Como métrica de dispersão dos dados, foram utilizados desvio padrão e percentil. E como medida de assimetria, foi utilizado *skew*, medida F_0 e medida F_2.

O percentil foi calculado através da divisão da amostra ordenada em porcentagens de dados aproximadamente iguais:

a) *Min*: é o menor elemento da amostra;

b) Q_1 (primeiro quartil): é o número que deixa 25% das observações abaixo e 75% acima;
c) Mediana (segundo quartil): é o número que deixa 50% das observações abaixo e 50% acima;
d) Q_3 (terceiro quartil): é o número que deixa 75% das observações abaixo e 25% acima;
e) *Max*: é o maior elemento da amostra.

A tri-média é calculada através da ponderação de 25% para o primeiro quartil (Q_1), 50% para a mediana e 25% para o terceiro quartil (Q_3).

Mediu-se assimetria de acordo com a medida F_2, que é dada por $F_k = \sum_{i=1}^{f} n_i^k$, em que n_i é o número de vezes que o *token i* aparece na coleção e k = 2. Para a distribuição de frequência de *tokens*, também usamos o número de *tokens* distintos como uma *feature* (definindo k = 0 na fórmula F_k).

Desta forma, foram geradas *features* calculadas a partir dessas estatísticas. Ambos os conjuntos de dados não ponderados e ponderados são representados por *features* para a distribuição de tamanho dos conjuntos e frequência de *tokens*; conjuntos ponderados também têm características para a distribuição de peso. Ao total foram geradas 26 e 38 *features* que sumarizam esses dados para conjuntos não ponderados e ponderados, respectivamente.

Porém, muitas destas *features* possuem significado intrinsecamente igual a outras, tornando necessário enxugá-las. Uma primeira tentativa foi utilizar algoritmos de seleção de *features*, porém não houve bons resultados. Desta forma, foi realizada uma seleção utilizando métodos empíricos.

Foram realizados testes utilizando algumas *features* base como comparação. A partir disso, para cada conjunto de estatísticas, foi selecionada a que obteve menor erro de predição em relação à base.

Desta forma, destas *features* foram selecionadas 13 e 16, para conjuntos não ponderados e ponderados, respectivamente. Além destas, foram somadas às *features*: *threshold*, *noSets* e *actualTime* para os dois conjuntos. Detalhes destas *features* são apresentados na Tabela 5.

4.6 Criação de modelos de predição

Para construir os modelos de predição, foram utilizadas árvores modelo, que são apresentadas na Seção 2.3.1.1. Dentro desde conceito, foi utilizado o algoritmo M5, que é um modelo de aprendizagem eficiente, que pode trabalhar com grandes dimensões de atributos, e foi introduzido por QUINLAN (1992). Foi também utilizado *boosting*, que é uma das técnicas de *ensemble*, que iterativamente cria novos modelos baseados no desempenho de modelos criados anteriormente.

Em nossos experimentos, usamos M5 e regressão aditiva com M5 para construção de modelos, ambas as implementações foram fornecidas pelo pacote de *software WEKA* (HALL et al., 2009) (o algoritmo M5 está nomeado com M5P neste pacote).

Também foram testadas outras técnicas de aprendizado de máquina, como *multilayer perceptron* e *support vector machines*. Desde os primeiros experimentos, a técnica *support vector machines* obteve resultados muito inferiores em relação ao M5P, que também superou a técnica *multilayer perceptron*. O trabalho de Mendes (2013) mostra que, para o problema proposto, M5P com regressão aditiva é a melhor opção, dentre as 3 técnicas estudadas.

Tabela 5 Features

Feature	Descrição
threshold	Limiar de similaridade passado como entrada do algoritmo *mpjoin* (valor entre 0 e 1)
noSets	Número de conjuntos existentes no conjunto de dados
setSizeTriMean	Tri-média dos tamanhos dos conjuntos
setSizeMin	Valor mínimo da amostra ordenada de tamanho do conjunto
setSizeQ1	Valor que se encontra 25% dos valores da amostra ordenada de tamanho de conjunto
setSizeMedian	Valor que se encontra 50% dos valores da amostra ordenada de tamanho de conjunto
setSizeQ3	Valor que se encontra 75% dos valores da amostra ordenada de tamanho de conjunto
setSizeMax	Valor máximo da amostra ordenada de tamanho de conjunto
setSizeF0	Quantidade de tamanhos de conjuntos distintos
setSizeF2	Frequência dos tamanhos de conjunto ao quadrado
setNormTriMean	Tri-média das normas dos conjuntos
setNormF0	Quantidade de normas distintas (somente para conjuntos
setNormF2	Frequência das normas ao quadrado (somente para conjuntos
tokFreqMean	Média da frequência dos *tokens* que compõem o conjunto de dados
tokFreqStdDeviatio	Desvio padrão da frequência dos *tokens* que compõem o conjunto
tokFreqSkewness	Assimetria dos *tokens* que compõem o conjunto de dados
tokF0	Quantidade de *tokens* distintos
tokF2	Frequência dos *tokens* ao quadrado
actualTime	Tempo que o algoritmo levou para executar a junção por similaridade sobre o

A configuração utilizada no algoritmo M5P utiliza os parâmetros:

a) número mínimo de instâncias nos nós folha (*minNumInstances*) = 4
b) construir árvore de regressão (*buildRegressionTree*) = *false*
c) não utilizar suavização (*useUnsmoothed*) = *false*

Para a utilização de regressão aditiva, a configuração foi:

- número de iterações (*numIterations*) = 10
- contração (*shrinkage*) = 1.0
- M5P com os mesmos parâmetros descritos acima

4.7 Configuração dos experimentos realizados

Esta seção tem o objetivo de detalhar a arquitetura dos experimentos executados. Foram executados diferentes experimentos que criaram e validaram os modelos de predição. Basicamente, os experimentos podem ser divididos entre os que realizam validação cruzada e os que treinaram com uma base de dados e testaram sobre outra base completamente diferente.

Abaixo são mostrados, para cada experimento, os dados para treinamento e os dados para validação. Os dados de treino ponderado são testados com dados para validação ponderados, da mesma forma que os dados de treino não pondera- dos são testados com dados não ponderados.

a) Experimento usando validação cruzada na plataforma servidor:
   - Treino: *Base;*
   - Validação: validação cruzada com dez subamostras aleatórias (chamadas *10-fold-cross-validation*).

b) Experimento usando validação cruzada na plataforma cliente:
   - Treino: *Hardware;*
   - Validação: validação cruzada com dez subamostras aleatórias (chamadas *10-fold-cross-validation*).

c) Experimento usando dados de treinamento DBLP e teste IMDB:
   - Treino: *Base* - conjuntos de dados extraídos a partir da base de dados DBLP (não ponderados e ponderados);
   - Validação: *Base* - conjuntos de dados extraídos a partir da base de dados IMDB (não ponderados e ponderados).

d) Experimento usando dados de treinamento IMDB e teste DBLP:
   - Treino: *Base* - conjuntos de dados extraídos a partir da base de dados IMDB (não ponderados e ponderados);
   - Validação: *Base* - conjuntos de dados extraídos a partir da base de dados DBLP (não ponderados e ponderados).

e) Experimento usando dados de treinamento obtidos de conjuntos de 50 k a 100 k *strings* e teste com 150 k a 200 k *strings*:
   - Treino: *Base*
   - Validação: *Conjuntos Maiores*

f) Experimento usando dados de treinamento obtidos de conjuntos de 150 k a 200 k *strings* e teste com 50 k a 100 k *strings*:
   - Treino: Conjuntos Maiores;
   - Validação: Base.

g) Experimento usando dados de treinamento obtidos de conjuntos de *strings* com 0 a 10 autores/atores ou atrizes e teste com 10 a 20 autores/atores ou atrizes:
   - Treino: *Base*
   - Validação: *Strings Maiores*

h) Experimento usando dados de treinamento obtidos de conjuntos de *strings* com 10 a 20 autores/atores ou atrizes e teste com 0 a 10 autores/atores ou atrizes:
   - Treino: Strings Maiores
   - Validação: Base

Mais detalhes sobre essas configurações são apresentadas na Seção 5, durante a apresentação dos resultados.

4.8 Validação de modelos de predição

Baseado no que foi discutido na Seção 2.3.2, foram calculados como medidas de erros: erro absoluto médio (EAM), erro absoluto relativo (EAR) e erro relativo médio (ERM) para medir a precisão da previsão. Como medida principal, estamos utilizando erro absoluto relativo (EAR), baseado nas informações presentes na Seção 2.3.2.

5 RESULTADOS E DISCUSSÃO

Após detalhamento das configurações dos experimentos, esta seção apresenta e analisa os resultados dos modelos de previsão criados. São apresentados os resultados obtidos utilizando o método de *cross-validation* e experimentação sobre diferentes bases de dados com o objetivo de verificar se os modelos gerados conseguem generalizar bem o problema. Foi utilizado o erro absoluto relativo como métrica de erro para comparação entre os experimentos, pois esta métrica é muito utilizada em problemas em que os valores alvo se diferem substancialmente, enquadrando neste trabalho, pois foram observadas execuções variando de poucos segundos a minutos.

5.1 Validação cruzada na plataforma servidor

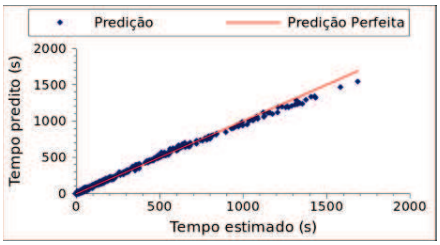
A Tabela 6 apresenta os resultados obtidos do experimento usando validação cruzada na plataforma servidor, para criação de modelos utilizando o algoritmo M5P e M5P com regressão aditiva. Como método de validação, foi utilizada validação cruzada.

Tabela 6 Erros de predição usando validação cruzada na plataforma servidor

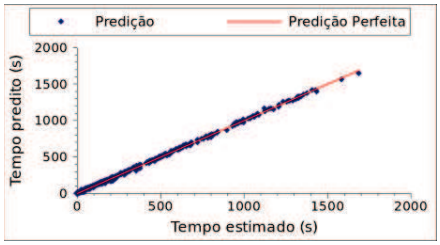
Dados de treinamento	Esquema de pesos	Modelo	EAM (s)	EAR (%)	ERM (%)
DBLP	não ponderado	M5P	7.05	5.76	68.07
		M5P + Regressão Aditiva	2.49	2.03	19.33
	ponderado	M5P	1.96	7.07	32.96
		M5P + Regressão Aditiva	0.82	2.96	12.15
IMDB	não ponderado	M5P	3.24	6.14	39.89
		M5P + Regressão Aditiva	1.12	2.12	11.37
	ponderado	M5P	1.09	7.99	26.15
		M5P + Regressão Aditiva	0.64	4.73	10.75

A princípio, nota-se que os erros de previsão (EAR) em todos os resultados estão com valor máximo de 7,99% (IMDB com ponderação, utilizando M5P). Além disso, pode-se observar que o uso de regressão aditiva conseguiu melhorar o desempenho para todos os modelos utilizados, possuindo o menor erro de predição como 2,03% (DBLP não ponderado), e mesmo sem o uso deste artifício, os resultados são considerados bons, o menor erro para M5P foi de 5,76% (DBLP não ponderado). Outra observação é que, há apenas uma ligeira variação nos resultados do DBLP e IMDB. Não foi possível inferir se bases de dados com conjuntos ponderados possuem potencial de serem mais bem preditas que bases de dados não ponderadas.

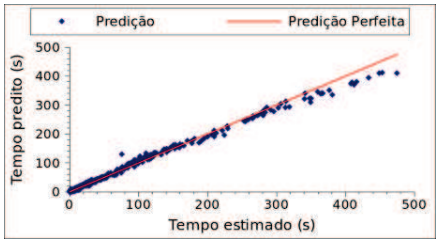
As Figuras 7 e 8 apresentam os gráficos de dispersão em que são confrontados os valores de tempo estimado e valores de tempos preditos para os dados de treinamento DBLP e IMDB, com e sem esquema de pesos. A partir da observação destes gráficos, verifica-se que houve um baixo número de previsões mal estimadas. Desta forma, a maioria das predições foram próximas à linha de predição perfeita. Grande parte dos pontos que se distanciaram da predição ideal quando utilizado M5P, foram mais bem preditos utilizando regressão aditiva.



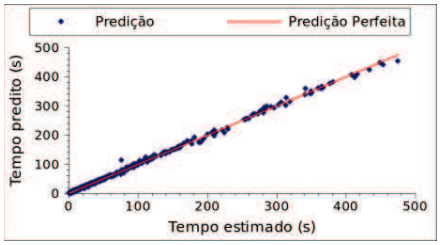
(a) M5P (sem peso)



(b) M5P + Regressão Aditiva (sem peso)

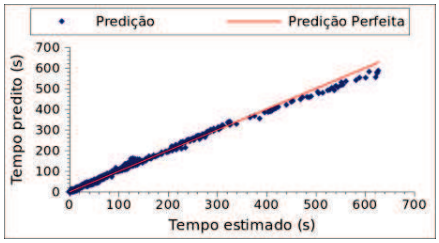


(c) M5P (com peso)

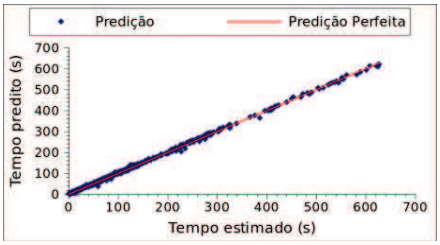


(d) M5P + Regressão Aditiva (com peso)

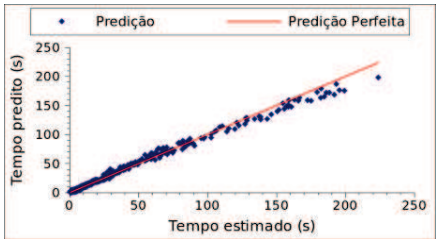
Figura 7 Treinamento sobre base DBLP, usando validação cruzada na plataforma servidor



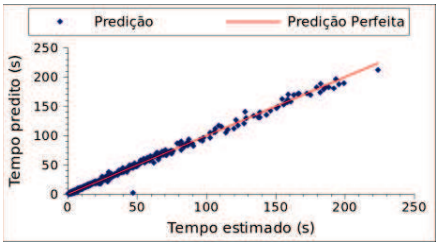
(a) M5P (sem peso)



(b) M5P + Regressão Aditiva (sem peso)



(c) M5P (com peso)



(d) M5P + Regressão Aditiva (com peso)

Figura 8 Treinamento sobre base IMDB, usando validação cruzada na plataforma servidor

5.2 Validação cruzada na plataforma cliente

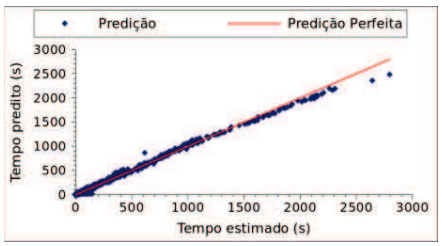
A Tabela 7 apresenta os resultados obtidos no experimento usando validação cruzada na plataforma cliente. Da mesma maneira que foram realizados os experimentos na plataforma servidor, para cada uma dessas bases, foram criados modelos utilizando o algoritmo M5P e M5P com regressão aditiva.

Neste experimento, foi possível notar que os erros de previsão permanecem quase os mesmos da plataforma servidor, sendo o maior erro 7,34% (IMDB com ponderação, utilizando M5P). Neste caso, o uso de regressão aditiva também possibilitou uma melhora na performance para todos os modelos utilizados, possuindo o menor erro de predição como 1,91% (DBLP não ponderado) e utilizando M5P temos 5,79% (DBLP não ponderado).

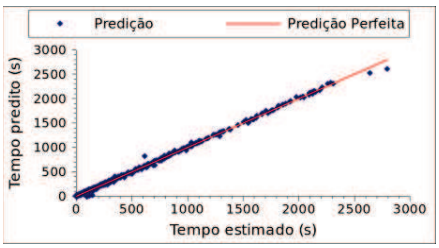
Tabela 7 Erros de predição usando validação cruzada na plataforma cliente

Dados de treinamento	Esquema de pesos	Modelo	EAM (s)	EAR (%)	ERM (%)
DBLP	não ponderado	M5P	12.70	6.07	71.32
		M5P + Regressão Aditiva	4.00	1.91	12.79
	ponderado	M5P	3.68	7.15	32.22
		M5P + Regressão Aditiva	1.30	2.53	7.76
IMDB	não ponderado	M5P	5.76	5.79	33.70
		M5P + Regressão Aditiva	2.25	2.26	9.45
	ponderado	M5P	1.93	7.34	24.39
		M5P + Regressão Aditiva	0.82	3.13	7.23

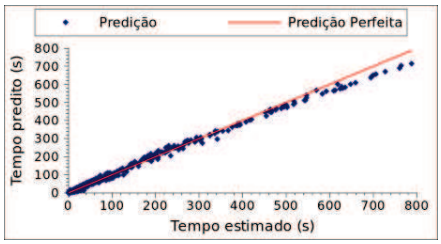
As Figuras 9 e 10 mostram os gráficos de dispersão para as predições de dados de treinamento DBLP e IMDB, respectivamente. Pode-se notar que as previsões mantiveram o mesmo comportamento da plataforma cliente, possuindo um resultado ligeiramente melhor. Assim, os nossos resultados demonstram que, em geral, pode-se prever efetivamente o tempo de junção por similaridade independente das configurações do sistema computacional.



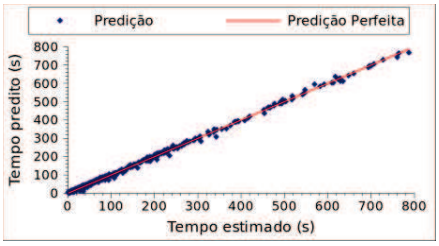
(a) M5P (sem peso)



(b) M5P + Regressão Aditiva (sem peso)

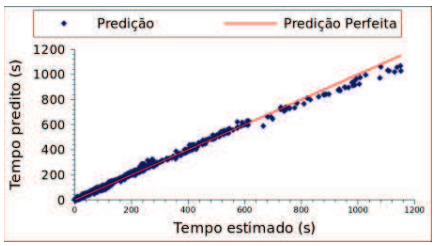


(c) M5P (com peso)

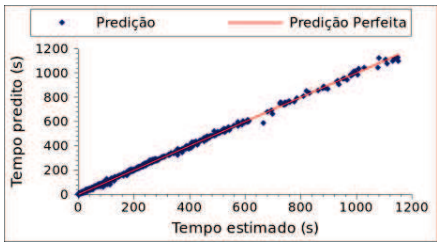


(d) M5P + Regressão Aditiva (com peso)

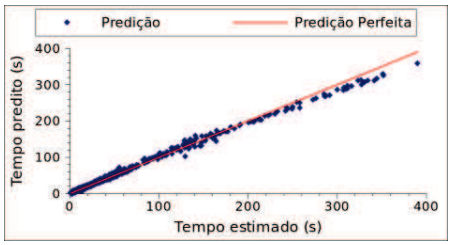
Figura 9 Treinamento sobre base DBLP, usando validação cruzada na plataforma cliente



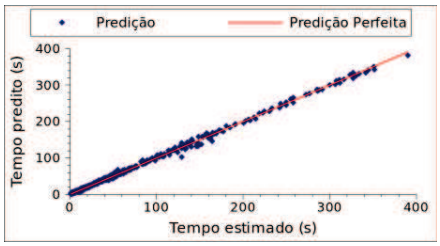
(a) M5P (sem peso)



(b) M5P + Regressão Aditiva (sem peso)



(c) M5P (sem peso)



(d) M5P + Regressão Aditiva (sem peso)

Figura 10 Treinamento sobre base IMDB, usando validação cruzada na plataforma cliente

5.3 Dados de treinamento DBLP

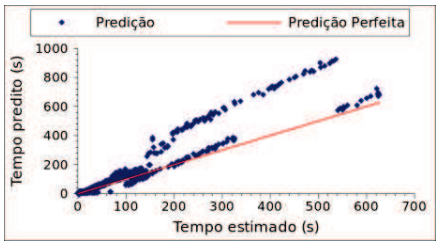
Serão apresentados na sequência, os resultados do experimento usando dados de treinamento DBLP e teste IMDB. A criação dos modelos foi feita a partir dos dados de treinamento relativos ao DBLP da configuração *base* e validados sobre os dados de treinamento relativos ao IMDB da configuração *base*.

Na Tabela 8, verificamos que os melhores resultados foram utilizando regressão aditiva, em que o erro de predição foi 16,86% e 11,06%, para conjuntos não ponderados e ponderados, respectivamente. Estes resultados surpreendem, pois os erros produzidos são relativamente baixos se considerarmos que as duas bases (DBLP e IMDB) possuem distribuição de valores de *features* diferentes.

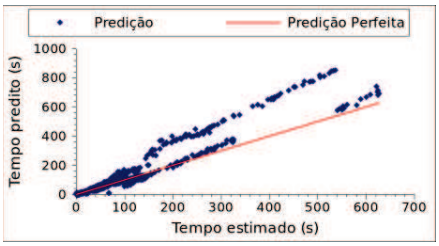
Tabela 8 Erros de predição usando dados de treinamento DBLP e teste IMDB

Dados de treinamento	Dados de teste	Esquema de pesos	Modelo	EAM (s)	EAR (%)	ERM (%)
DBLP	IMDB	não ponderado	M5P	18.25	21.86	81.11
			M5P + Regressão Aditiva	14.07	16.86	38.58
		ponderado	M5P	3.11	15.11	57.08
			M5P + Regressão Aditiva	2.27	11.06	36.81

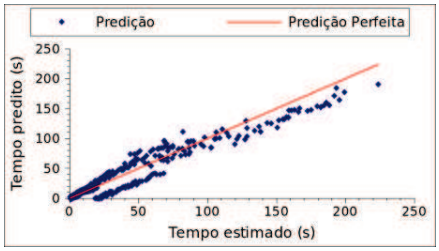
A Figura 11 mostra os gráficos de dispersão para as predições de treino usando base DBLP e teste sobre base IMDB. Vemos que grande parte das predições, para dados de treinamento sem peso, estão acima do valor estimado, o que indica que os dados DBLP (sem ponderação) possuem valores de distribuição das *features* maiores que os apresentados no IMDB.



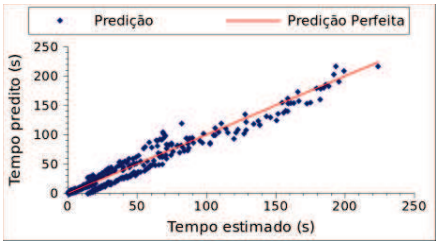
(a) M5P (sem peso)



(b) M5P + Regressão Aditiva (sem peso)



(c) M5P (com peso)



(d) M5P + Regressão Aditiva (com peso)

Figura 11 Treinamento sobre base DBLP, validação usando base IMDB

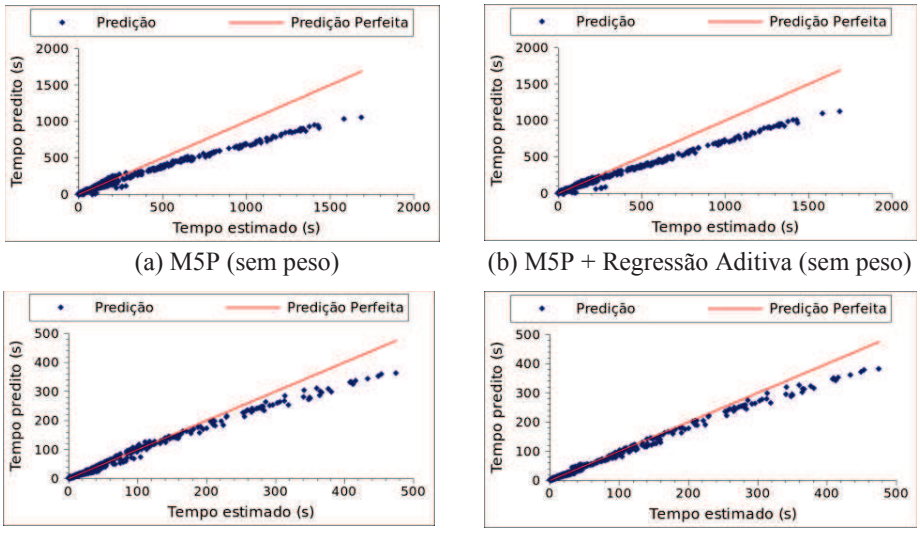
5.4 Dados de treinamento IMDB

O experimento usando dados de treinamento IMDB e teste DBLP adotou a mesma metodologia do experimento anterior, trocando a base de treino e teste. Assim, os modelos foram criados a partir dos dados relativos ao IMDB e testados sobre os dados relativos ao DBLP. Para esse experimento, verificamos na Tabela 9 o erro de predição de 24,73% e 11,27%, para conjuntos não ponderados e ponderados, respectivamente.

Tabela 9 Erros de predição utilizando dados de treinamento IMDB e teste DBLP

Dados de treinamento	Dados de teste	Esquema de pesos	Modelo	EAM (s)	EAR (%)	ERM (%)
IMDB	DBLP	não ponderado	M5P	24.28	25.45	51.68
			M5P + Regressão Aditiva	23.60	24.73	46.37
		ponderado	M5P	3.09	14.09	29.00
			M5P + Regressão Aditiva	2.47	11.27	25.22

A Figura 12 mostra os gráficos de dispersão para as predições de treino usando base IMDB e teste sobre base DBLP. As predições possuem erros de predição baixos. O uso de regressão aditiva gera uma ligeira melhora na predição. Conforme afirmado no experimento anterior, os gráficos de dispersão para as predições de treino usando base IMDB e teste sobre base DBLP mostram que houve uma subestimação do tempo, causada pelas diferenças de distribuição das *features* nas bases de dados utilizadas.



(a) M5P (sem peso)

(b) M5P + Regressão Aditiva (sem peso)

(c) M5P (com peso)

(d) M5P + Regressão Aditiva (com peso)

Figura 12 Treinamento sobre base IMDB, validação usando base DBLP

5.5 Dados de treinamento de tamanho 50 k a 100 k

Nesta seção, são apresentados os resultados do experimento usando dados de treinamento obtidos de conjuntos de 50 k a 100 k *strings* e teste com 150 k a 200 k *strings*. Assim, o modelo foi aprendido por meio de dados da configuração *base* e testados sobre conjuntos de dados da configuração *conjuntos Maiores*.

Lidar com diferenças de tamanho entre dados de treino e teste é um problema notoriamente difícil, independente da técnica de aprendizagem de máquina. Por isso, de antemão já não foi esperado obter os mesmos resultados dos experimentos anteriores.

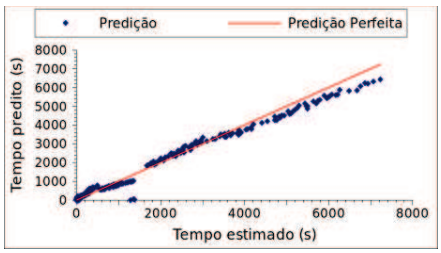
A Tabela 10 mostra os resultados dos modelos criados. Como esperado, os erros na predição aumentaram dramaticamente, atingindo quase 84,08% no caso do treinamento sobre a base de dados IMDB de configuração *base* e testado sobre a configuração *conjuntos Maiores*.

Tabela 10 Erros de predição usando dados de treinamento obtidos de conjuntos de 50 k a 100 k *strings* e teste com 150 k a 200 k *strings*

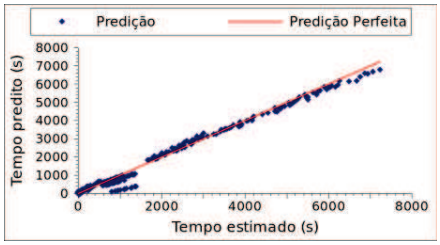
Dados	Tamanho dados treino	Tamanho dados teste	Esquema de pesos	Modelo	EAM (s)	EAR (%)	ERM (%)
DBLP	50k a 100k	150k a 200k	não ponderado	M5P	134.19	24.73	52.74
				M5P + Regressão Aditiva	64.03	11.80	70.10
			ponderado	M5P	23.12	19.99	69.60
				M5P + Regressão Aditiva	27.56	23.83	59.67
IMDB	50k a 100k	150k a 200k	não ponderado	M5P	53.41	20.70	100.63
				M5P + Regressão Aditiva	48.54	18.81	135.91
			ponderado	M5P	51.86	84.08	108.77
				M5P + Regressão Aditiva	50.01	81.08	100.73

Entretanto, para os dados de treinamento IMDB não ponderado, houve uma diminuição dos erros, se comparado ao experimento anterior (treinamento com IMDB e teste sobre DBLP): atingiu-se 20,70% e 18,81% de erro para os modelos M5P e M5P com regressão aditiva, respectivamente.

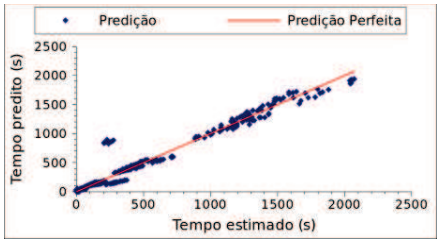
Observando a Figura 13, notamos que os dois modelos comportaram de forma semelhante, tendo como erro 19,99% e 23,83%, para os modelos M5P e M5P com regressão aditiva, respectivamente. A Figura 14 apresenta os gráficos de dispersão para os experimentos de dados de treinamento IMDB. Constata-se que apesar de haver predições fora do esperado, a grande maioria dos pontos seguiu a tendência da linha de predição perfeita para a base de dados sem peso. A base de dados com peso, apesar de possuir o maior erro dentre as predições, possui parte dos pontos de predição alinhados à reta de predição perfeita, tendo como motivo para a grande porcentagem de erro, os pontos que estão visivelmente fora da reta.



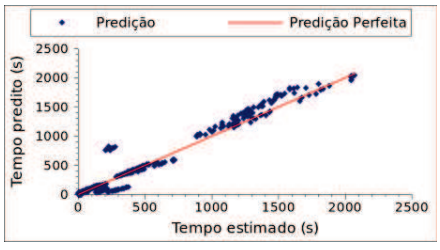
(a) M5P (sem peso)



(b) M5P + Regressão Aditiva (sem peso)

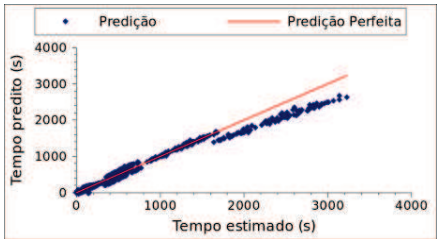


(c) M5P (com peso)

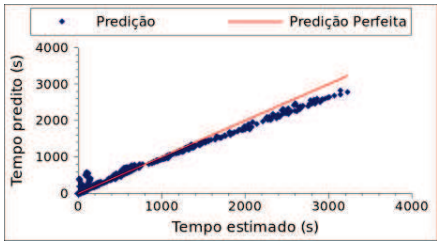


(d) M5P + Regressão Aditiva (com peso)

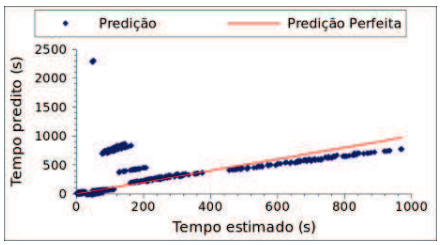
Figura 13 Treinamento sobre base DBLP, usando dados de treinamento obtidos de conjuntos de 50 k a 100 k *strings* e teste com 150 k a 200 k *strings*



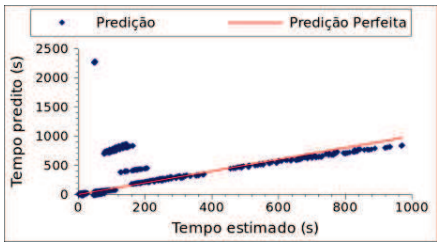
(a) M5P (sem peso)



(b) M5P + Regressão Aditiva (sem peso)



(c) M5P (com peso)



(d) M5P + Regressão Aditiva (com peso)

Figura 14 Treinamento sobre base IMDB, usando dados de treinamento obtidos de conjuntos de 50 k a 100 k *strings* e teste com 150 k a 200 k *strings*

5.6 Dados de treinamento de tamanho 150 k a 200 k

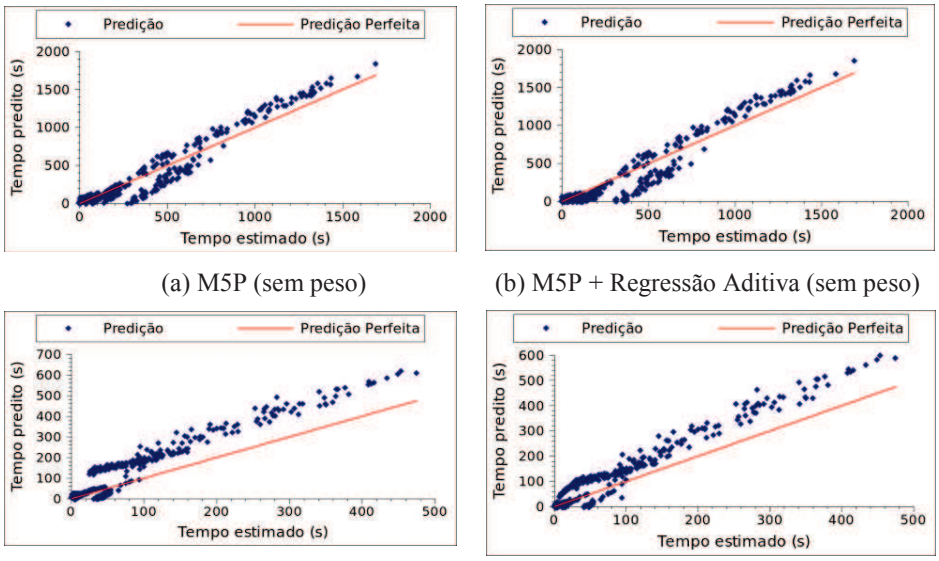
O experimento usando dados de treinamento obtidos de conjuntos de 150 k a 200 k *strings* e teste com 50 k a 100 k *strings*, repetiu a mesma configuração utilizada no experimento anterior. Todavia, foram invertidos os dados de treino e teste, desta forma, a criação do modelo foi feita a partir de dados da configuração *conjuntos Maiores* e a validação com conjuntos de dados da *base*. A Tabela 11 apresenta os erros de predição para essa abordagem.

Tabela 11 Erros de predição usando dados de treinamento obtidos de conjuntos de 150 k a 200 k *strings* e teste com 50 k a 100 k *strings*

Dados	Tamanho dados treino	Tamanho dados teste	Esquema de pesos	Modelo	EAM (s)	EAR (%)	ERM (%)
DBLP	150 k a 200 k	50 k a 100 k	não ponderado	M5P	50.00	10.03	674.21
				M5P + Regressão Aditiva	42.35	8.50	271.25
			ponderado	M5P	19.43	17.60	330.41
				M5P + Regressão Aditiva	15.29	13.85	136.56
IMDB	150 k a 200 k	50 k a 100 k	não ponderado	M5P	34.55	14.58	546.59
				M5P + Regressão Aditiva	46.04	19.43	125.51
			ponderado	M5P	8.71	14.62	267.26
				M5P + Regressão Aditiva	7.51	12.60	139.68

Percebe-se uma melhora no erro de predição em quase todos os experimentos (exceto IMDB não ponderado utilizando M5P + regressão aditiva), em relação aos experimentos anteriores, que possuem as bases de treino e testes opostas. Destacaremos os dados de treinamento IMDB ponderado, que obteve erro de somente 12,60% utilizando o modelo de regressão aditiva com M5P, que no experimento anterior foi de 81,08%.

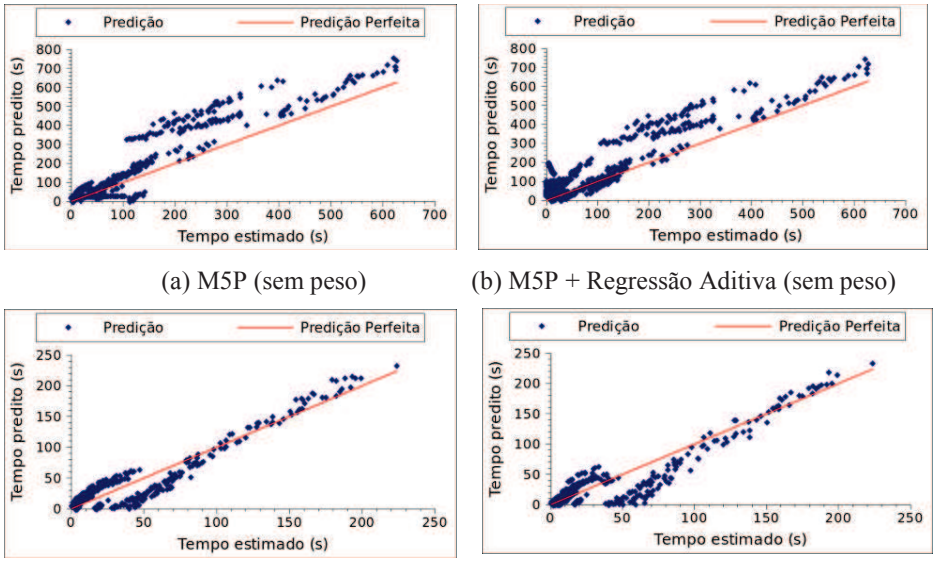
Apesar do baixo erro, vemos na Figura 15 que valores preditos na grande maioria das vezes estão acima do valor estimado. O mesmo ocorre para as previsões apresentadas na Figura 16 para a base de dados IMDB sem peso.



(a) M5P (sem peso) (b) M5P + Regressão Aditiva (sem peso)

(c) M5P (com peso) (d) M5P + Regressão Aditiva (com peso)

Figura 15 Treinamento sobre base DBLP, usando dados de treinamento obtidos de conjuntos de 150 k a 200 k *strings* e teste com 50 k a 100 k *strings*



(a) M5P (sem peso) (b) M5P + Regressão Aditiva (sem peso)

(c) M5P (com peso) (d) M5P + Regressão Aditiva (com peso)

Figura 16 Treinamento sobre base IMDB, usando dados de treinamento obtidos de conjuntos de 150 k a 200 k *strings* e teste com 50 k a 100 k *strings*

5.7 Dados de treinamento de *strings* tamanho 0 a 10

O experimento, usando dados de treinamento obtidos de conjuntos de *strings* com 0 a 10 autores/atores ou atrizes e teste com 10 a 20 atores ou atrizes, criou modelos com os dados de treinamento IMDB da configuração *base*, seguida da validação sobre dados da configuração *strings Maiores*.

A Tabela 12 apresenta o resultado desse experimento. Podemos notar que este experimento é o que possui maior erro de predição. O erro para o modelo M5P foi de 240,86% e para o modelo M5P com regressão aditiva foi de 253,18% em bases de dados ponderados.

Acreditamos que tal dificuldade em generalizar modelos seja devido às características intrínsecas de distribuições de frequência de palavras, que são caracterizadas por grande quantidade de palavras raras, e que influenciam no

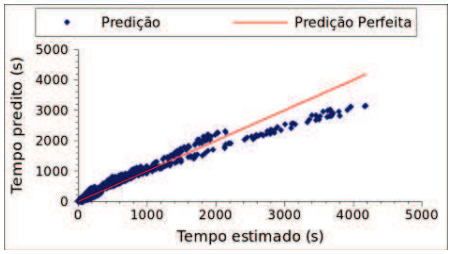
desempenho de junções por similaridade. Essas distribuições são caracterizadas por mudanças sistemáticas nos momentos de frequência como o aumento ou diminuição do tamanho da amostra (BAAYEN, 2001).

Ao considerar mais ou menos os campos quando formamos *strings* para o conjunto de dados, estamos mudando a distribuição da frequência das palavras subjacentes, que por sua vez, afetam estatísticas utilizadas no nosso modelo.

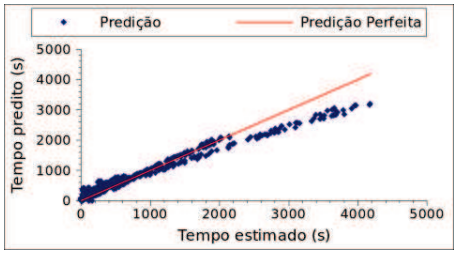
A Figura 17 mostra o gráfico de dispersão para os modelos, observa-se que nos dois modelos, em dados ponderados, uma grande quantidade das predições está bem acima do valor estimado. Em contrapartida, houve uma boa regressão para conjuntos sem peso.

Tabela 12 Erros de predição usando dados de treinamento obtidos de conjuntos de *strings* com 0 a 10 autores/atores ou atrizes e teste com 10 a 20 autores/atores ou atrizes

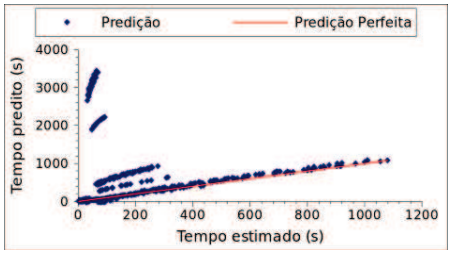
Dados	Tamanho *strings* treino	Tamanho *strings* teste	Esquema de pesos	Modelo	EAM (s)	EAR (%)	ERM (%)
IMDB	0 a 10	10 a 20	não ponderado	M5P	61.71	22.21	73.25
				M5P + Regressão Aditiva	76.18	27.42	151.06
			ponderado	M5P	133.16	240.86	293.67
				M5P + Regressão Aditiva	139.97	253.18	321.61



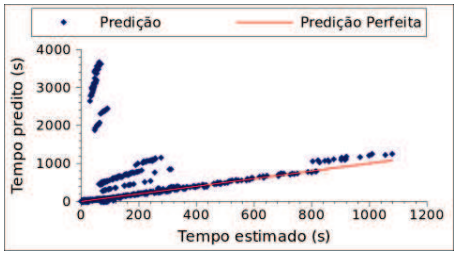
(a) M5P (sem peso)



(b) M5P + Regressão Aditiva (sem peso)



(c) M5P (com peso)



(d) M5P + Regressão Aditiva (com peso)

Figura 17 Predição de tempo sobre base IMDB, usando dados de treinamento obtidos de conjuntos de *strings* com 0 a 10 atores ou atrizes e teste com 10 a 20 atores ou atrizes

5.8 Dados de treinamento de *strings* tamanho 10 a 20

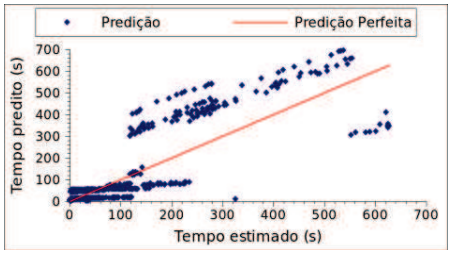
Finalmente foi executado o experimento usando dados de treinamento obtidos de conjuntos de *strings* com 10 a 20 atores ou atrizes e teste com 0 a 10 atores ou atrizes. A Tabela 13 traz os resultados das predições de tempo de execução para os dados de treinamento da configuração *strings Maiores* e validação realizada sobre dados *base*. Esta abordagem possui resultados melhores que a anterior, tendo erro para o modelo M5P de apenas 8,22%, que é um erro próximo do encontrado utilizando validação cruzada. Porém, ocorreram erros altos durante a utilização de regressão aditiva, sendo estes consideravelmente maiores que os apresentados com M5P.

A Figura 18 apresenta o gráfico de dispersão para os modelos, observa-se que o modelo utilizando M5P com dados ponderados foi o que obteve

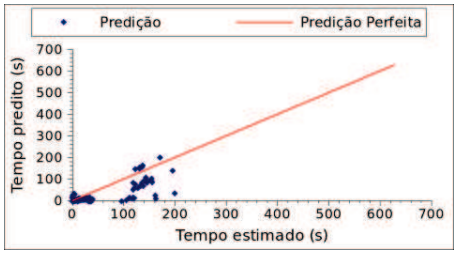
melhor resultado. Enquanto o pior foi o que continha dados ponderados e utilizando M5P + regressão aditiva.

Tabela 13 Erros de predição usando dados de treinamento obtidos de conjuntos de *strings* com 10 a 20 autores/atores ou atrizes e teste com 0 a 10 autores/atores ou atrizes

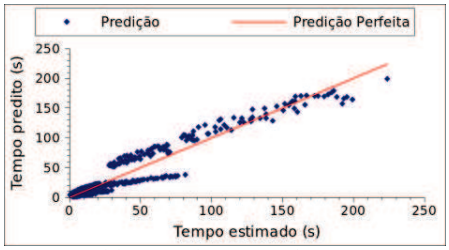
Dados	Tamanho *strings* treino	Tamanho *strings* teste	Esquema de pesos	Modelo	EAM (s)	EAR (%)	ERM (%)
IMDB	10 a 20	0 a 10	não ponderado	M5P	31.51	12.07	283.87
				M5P + Regressão Aditiva	160.75	61.57	1081.1
			ponderado	M5P	4.46	8.22	141.51
				M5P + Regressão Aditiva	37.98	69.91	931.62



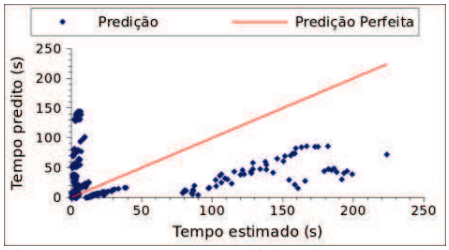
(a) M5P (sem peso)



(b) M5P + Regressão Aditiva (sem peso)



(c) M5P (com peso)



(d) M5P + Regressão Aditiva (com peso)

Figura 18 Treinamento sobre base IMDB, usando dados de treinamento obtidos de conjuntos de *strings* com 10 a 20 atores ou atrizes e teste com 0 a 10 atores ou atrizes

6 CONCLUSÃO E TRABALHOS FUTUROS

6.1 Conclusão

Previsão de desempenho de consultas é uma tecnologia fundamental em serviços de gestão de dados hospedados na nuvem. Por esse motivo, uma grande quantidade de trabalhos recentes tem abordado o problema de previsão de tempo de execução de consultas tradicionais SQL.

Neste trabalho, considerou-se o problema da predição de tempo para junções por similaridade baseadas em conjuntos. Adotou-se uma abordagem orientada a experimentos com o objetivo de construir modelos para este fim. Desta forma, foram identificadas *features* de fácil coleta para serem utilizadas como entrada de técnicas de aprendizagem de máquina. Assim, apresentou-se um arcabouço que implanta o modelo de previsão de tempo de consultas por similaridade. A avaliação experimental, não só demonstrou a viabilidade de prever o desempenho de algoritmos complexos de similaridade, mas também mostrou resultados com bons valores de precisão. Dentre todos os experimentos, os que obtiveram melhores resultados foram os apresentados nas Seções 5.1 e 5.2, pelo fato dos dados de treino e teste serem extraídos da mesma base de dados, e a partir destes concluímos que é possível prever o tempo de junções de similaridade independente da arquitetura de *hardware* utilizada.

Os resultados dos experimentos apresentados nas Seções 5.3 e 5.4 mostram que é possível gerar um modelo a partir de uma base de dados e utilizá-lo na predição de tempo de outras bases de dados. Isto valida a utilidade do arcabouço desenvolvido.

Os demais experimentos mostram que pode ser necessário outros artifícios para encontrar modelos que consigam generalizar o problema estudado. Acreditamos que estes problemas são causados pelas características

intrínsecas da distribuição de frequência de palavras, que influenciam substancialmente o desempenho de junções por similaridade. Ao considerar mais ou menos campos quando formamos *strings* para o conjunto de dados, estamos realmente mudando o tamanho da distribuição de frequência das palavras, que por sua vez, afetam as estatísticas em que o nosso modelo se baseia (BAAYEN, 2001).

Essas variações complexas não puderam ser capturadas até agora por qualquer técnica de aprendizado de máquina investigada. Por isso, deixamos este problema desafiador aberto para trabalhos futuros. Na Seção 6, são listadas algumas ideias que possuem potencial para isso.

6.2 Trabalhos Futuros

Como trabalhos futuros, visamos novas metodologias para a criação de modelos de predição. Primeiramente, podem ser utilizadas estatísticas adicionais, como, por exemplo, estimativas de tamanho de candidatos e resultado de junção por similaridade recentemente publicados na literatura por Lee, Ng e Shim (2009, 2011) e Lu et al. (2013), com o objetivo de melhorar a generalização quando existem consideráveis divergências entre os dados de treino e teste.

Além da criação dos modelos de aprendizagem de máquina, podem ser criadas funções de escalonamento, que são utilizadas nas *features* com o objetivo de melhorar os valores de predição, como feito no trabalho de LI et al. (2012).

Outra abordagem, que pode melhorar os resultados, é a criação de mais de um modelo, como explorado no trabalho de Ganapathi et al. (2009). Neste caso, poderia ser explorada a criação de dois modelos: um para a predição de tempo da geração de candidatos e o outro para a fase de verificação. Desta

forma, podem ser selecionadas *features* mais específicas para cada fase, havendo um potencial de melhoria dos resultados.

Consideramos neste trabalho apenas a previsão de uma única consulta de junção por similaridade por vez. Prever o tempo de execução de várias consultas executadas simultaneamente é um problema importante e desafiador para trabalhos futuros.

REFERÊNCIAS

AHMAD, M.; BOWMAN, I. T. Predicting system performance for multi-tenant database workloads. In: INTERNATIONAL WORKSHOP ON TESTING DATABASE SYSTEMS, 4., 2011, New York. **Proceedings…** New York: ACM, 2011. p. 6:1-6:6.

AKDERE, M. et al. Learning-based query performance modeling and prediction. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 28., 2012, Washington. **Proceedings…** Washington: IEEE Computer Society, 2012. p. 390-401.

ARASU, A.; GANTI, V.; KAUSHIK, R. Efficient exact set-similarity joins. **VLDB Endowment**, Seoul, v. 6, p. 918-929, 2006. Disponível em: <http://dl.acm.org/citation-.cfm?id=1182635.1164206>. Acesso em: 10 fev. 2014.

ARMSTRONG, J.; COLLOPY, F. Error measures for generalizing about forecas- ting methods: empirical comparisons. **International Journal of Forecasting**, Amsterdam, v. 8, n. 1, p. 69-80, 1992.

BAAYEN, H. **Word frequency distributions**. Dordrecht: Kluwer Academic, 2001. (Text, Speech and Language Technology). Disponível em: <http://opac.inria.fr/record=b1097264>.

BAYARDO, R. J.; MA, Y.; SRIKANT, R. **Scaling up all pairs similarity search**. New York: ACM, 2007. 140 p.

CHAUDHURI, S. What next?: a half-dozen data management research goals for big data and the cloud. In: SYMPOSIUM ON PRINCIPLES OF DATABASE SYSTEMS, 31., 2012, New York. **Proceedings…** New York: ACM, 2012. p. 1-4.

CHAUDHURI, S.; GANTI, V.; KAUSHIK, R. **A primitive operator for similarity joins in data cleaning**. Washington: IEEE Computer Society, 2006. 5 p.

DUGGAN, J. et al. Performance prediction for concurrent database workloads. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 11., 2011, New York. **Proceedings…** New York: ACM, 2011. p. 337-348.

GANAPATHI, A. et al. Predicting multiple metrics for queries: better decisions enabled by machine learning. In: IOANNIDIS, Y. E.; LEE, D. L.; NG, R. T. (Ed.). **ICDE**. New York: IEEE, 2009. p. 592-603.

GONZALEZ, H. et al. Google fusion tables: web-centered data management and collaboration. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 10., 2010, New York. **Proceedings…** New York: ACM, 2010. p. 1061-1066.

HALL, M. et al. The weka data mining software: an update. **SIGKDD Explorations News**, New York, v. 11, n. 1, p. 10-18, Nov. 2009.

HAN, J.; KAMBER, M. **Data mining:** concepts and techniques. San Francisco: M. Kaufmann, 2006. 770 p.

HAN, J.; KAMBER, M.; PEI, J. **Data mining:** concepts and techniques. 3rd ed. San Francisco: M. Kaufmann, 2011. 744 p.

KAISLER, S. et al. Big data: issues and challenges moving forward. In: HAWAII INTERNATIONAL CONFERENCE SYSTEM SCIENCES, 46., 2013, Waikoloa. **Proceedings…** Waikoloa: HICSS, 2013. p. 995-1004.

LEE, H.; NG, R. T.; SHIM, K. Power-law based estimation of set similarity join size. **VLDB Endowment**, Singapore, v. 2, n. 1, p. 658-669, Aug. 2009. Disponível em: <http://dl.acm.org/citation.cfm?id=1687627-.1687702>. Acesso em: 12 fev. 2014.

LEE, H.; NG, R. T.; SHIM, K. Similarity join size estimation using locality sensitive hashing. **VLDB Endowment**, Singapore, v. 4, n. 6, p. 338-349, Mar. 2011. Disponível em: <http://dl.acm.org/citation-.cfm?id=1978665.1978666>. Acesso em: 10 fev. 2014.

LEHNER, W.; SATTLER, K. U. **Web-scale data management for the cloud**. New York: Springer, 2013. 193 p.

LEVENSHTEIN, V. I. Binary codes capable of correcting deletions, insertions and reversals. **Soviet Physics Doklady**, Moscow, v. 10, n. 8, p. 707-710, Feb. 1966.

LI, J. et al. Robust estimation of resource consumption for sql queries using statistical techniques. **PVLDB**, Singapore, v. 5, n. 11, p. 1555-1566, 2012.
LU, J. et al. String similarity measures and joins with synonyms. In: ACM

SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 13., 2013, New York. **Proceedings…** New York: ACM, 2013. p. 373-384.

MENDES, D. S. **Uma abordagem baseada em aprendizagem de máquina para predição de desempenho de junções por similaridade**. 2013. 69 p. Monografia (Graduação em Ciência da Computação) - Universidade Federal de Lavras, Lavras, 2013.

MOGROVEJO, R. J. M. et al. Towards a statistical evaluation of piglatin joins. **Journal of Information and Data Management**, Recife, v. 4, n. 3, p. 483-498, 2013.

QUINLAN, R. J. Learning with continuous classes. In: AUSTRALIAN JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 5., 1992, Singapore. **Proceedings…** Singapore: World Scientific, 1992. p. 343-348.

RIBEIRO, L. A. **A framework for XML similarity joins**. 2010. 220 p. Thesis (Ph.D. in Computer's Science) - University of Kaiserslautern, Kaiserslautern, 2010.

RIBEIRO, L. A.; HÄRDER, T. Generalizing prefix filtering to improve set similarity joins. **Information Systems**, Oxford, v. 36, n. 1, p. 62-78, Mar. 2011.

ROBERTSON, S. E.; JONES, K. S. Document retrieval systems. In: WILLETT, P. (Ed.). **Relevance weighting of search terms**. London: T. Graham, 1988. p. 143-160. Disponível em: <http://dl.acm.org/citation.cfm?id=106765.106783>. Acesso em: 10 dez. 2013.

SARAWAGI, S.; KIRPAL, A. **Efficient set joins on similarity predicates**. New York: ACM, 2004. 754 p.

TAN, P. et al. **Introdução ao datamining:** mineração de dados. São Paulo: Ciência Moderna, 2009. Disponível em: <http://books.google.com.br/books?id=69d6PgAACAAJ>. Acesso em: 10 dez. 2013.

UKKONEN, E. Approximate string-matching with q-grams and maximal matches. **Theoretical Computer Science**, Essex, v. 92, n. 1, p. 191-211, Jan. 1992.

VAISHNAVI, V.; KUECHLER, W. **Design research in information systems**. Disponível em: <http://www.isworld.org/Researchdesign/drisISworld.htm>. Acesso em: 9 set. 2004.

VERNICA, R.; CAREY, M. J.; LI, C. Efficient parallel set-similarity joins using mapreduce. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 10., 2010, New York. **Proceedings…** New York: ACM, 2010. p. 495-506.

WITTEN, I. H.; FRANK, E.; HALL, M. A. **Data mining:** practical machine learning tools and techniques. 3rd ed. Amsterdam: M. Kaufmann, 2011. 664 p.

WU, W. et al. Predicting query execution time: are optimizer cost models really unusable? In: JENSEN, C. S.; JERMAINE, C. M.; ZHOU, X. (Ed.). **ICDE**. New York: IEEE Computer Society, 2013. p. 1081-1092.

XIAO, C. et al. **Efficient similarity joins for near duplicate detection**. New York: ACM, 2008. 140 p.

XIAO, C. et al. Top-k set similarity joins. In: IOANNIDIS, Y. E.; LEE, D. L.; NG, R. T. (Ed.). **ICDE**. New York: IEEE, 2009. p. 916-927.

ZHANG, N. et al. Statistical learning techniques for costing xml queries. In: _____.**VLDB**. New York: IEEE, 2005. p. 289-300.

ZOBEL, J.; MOFFAT, A. Exploring the similarity space. **SIGIR Forum**, New York, v. 32, n. 1, p. 18-34, Apr. 1998.