DIEGO SARMENTO MENDES

UMA ABORDAGEM BASEADA EM APRENDIZAGEM DE MÁQUINA PARA PREDIÇÃO DE DESEMPENHO DE JUNÇÕES POR SIMILARIDADE

LAVRAS – MG

2013

DIEGO SARMENTO MENDES

UMA ABORDAGEM BASEADA EM APRENDIZAGEM DE MÁQUINA PARA PREDIÇÃO DE DESEMPENHO DE JUNÇÕES POR SIMILARIDADE

Monografia apresentada ao Colegiado do Curso de Ciência da Computação, para obtenção do título de bacharel em Ciência da Computação.

Orientador
Prof. Dr. Leonardo Andrade Ribeiro

LAVRAS – MG

2013

DIEGO SARMENTO MENDES

UMA ABORDAGEM BASEADA EM APRENDIZAGEM DE MÁQUINA PARA PREDIÇÃO DE DESEMPENHO DE JUNÇÕES POR SIMILARIDADE

Monografia apresentada ao Colegiado do Curso de Ciência da Computação, para obtenção do título de bacharel em Ciência da Computação.

APROVADA em 29 de Agosto de 2013.

Prof. Cristiano Leite de Castro	UFLA
Prof. Denilson Alves Pereira	UFLA

Leonardo A Ribeiro

Prof. Dr. Leonardo Andrade Ribeiro

(Orientador)

LAVRAS – MG

2013

Dedico esta monografia primeiramente a Deus, à minha família e a minha namorada Karina Rodrigues, que sempre me apoiaram.

AGRADECIMENTOS

Agradeço principalmente ao meu orientador Leonardo Ribeiro, que considero um exemplo a ser seguido.

À Comp Júnior, empresa da qual só guardo boas recordações e experiências.

Aos meus colegas de laboratório e amigos, que me acompanharam durante esta etapa de minha vida.

Obrigado.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Com a necessidade cada vez maior de lidar com grandes volumes de dados e a expansão das tecnologias de computação em nuvem, prever de forma precisa a performance de consultas é uma tarefa imprescindível em sistemas de otimização e gerenciamento de recursos de *hardware* para bancos de dados. Trabalhos recentes abordaram este problema no contexto de operadores tradicionais de Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDRs). Entretanto, até o presente momento não se conhece nenhum trabalho abordando a predição de operações avançadas baseadas no conceito de similaridade. Diante deste cenário, nesta pesquisa foram desenvolvidos modelos para prever, de forma precisa, o tempo de execução de junções por similaridade, que são essenciais para atividades de limpeza e integração de dados. Resultados utilizando técnicas de aprendizagem de máquina se mostraram eficientes, indicando que tais modelos poderiam ser utilizados para compor ferramentas de controle de admissão de recursos de *hardware* para a execução de tais atividades.

Palavras-Chave: similaridade, duplicatas, predição de performance, aprendizagem de máquina

1 INTRODUÇÃO

Um problema muito comum em sistemas de banco de dados é a presença de múltiplas representações de uma mesma entidade do mundo real, também chamadas de duplicatas, por brevidade. Este tipo de informação redundante pode causar sérios transtornos em diversos cenários. Por exemplo, em sistemas de controle de estoque, operações como envio de cobranças e produtos podem ser repetidas desnecessariamente; em sistemas de suporte a decisão, modelos de mineração de dados podem ser corrompidos devido a estatísticas infladas erroneamente.

A identificação de duplicatas é essencial em diversos contextos, sendo central em aplicações de limpeza e integração de dados, tarefas amplamente utilizadas atualmente. Por exemplo, mecanismos de busca na Web necessitam encontrar e filtrar páginas na Internet de forma extremamente rápida, onde páginas duplicadas devem ser identificadas para que o serviço seja provido com a melhor qualidade. Outras aplicações nas quais identificar duplicatas é essencial são tarefas que envolvem mineração de dados, onde em mais de 89% dos projetos que envolvem tal atividade gastam mais de 40% do tempo em atividades de limpeza e preparação dos dados (BRAZIER, 2003).

Entretanto, a identificação de duplicatas é difícil porque, muitas vezes, as mesmas não são idênticas entre si. Este é o caso de duplicatas textuais geradas por erros tipográficos de usuários ou ausência de padronização na forma de representar informações, como nome e endereço, por exemplo. Nestes casos, operações baseadas no conceito de similaridade são imprescindíveis: por exemplo, uma junção por similaridade pode ser usada para encontrar objetos similares em um banco de dados; estes objetos são então classificados como possíveis duplicatas e selecionados para análise posterior.

Diante deste cenário, trabalhos anteriores mostraram que técnicas de junção por similaridade baseadas em conjuntos oferecem uma alternativa versátil e eficiente para junções por similaridade, utilizando uma série de filtros em etapas de geração de candidatos para evitar comparações entre todos os registros (SARAWAGI; KIRPAL, 2004; XIAO *et al.*, 2011; RIBEIRO; HÄRDER, 2011).

Por outro lado, com a rápida expansão das tecnologias de computação em nuvem, diversas tarefas relacionadas ao controle de admissão de *hardware* para clientes tornaram-se fundamentais. Saber ao certo qual a quantidade de recursos de *hardware* a ser reservada para clientes de forma dinâmica exige estimativas precisas do custo computacional de consultas às bases de dados, o que não é uma tarefa simples, uma vez que os modelos de custo disponíveis nos Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDRs) visam encontrar o melhor plano de consultas, e não realizar predições exatas de desempenho, indicando custos que não são facilmente mapeados para o tempo e a quantidade de processamento que uma consulta irá demandar. A predição de desempenho de consultas pode ser utilizada em diferentes aspectos: dar suporte à decisão de executar ou não tal consulta, caso a mesma demore horas ou apenas segundos; determinar qual a quantidade de recursos de *hardware* a ser utilizado para tal tarefa; planejar a quantidade ideal de recursos de *hardware* alocados para a execução de uma tarefa para continuar provendo um serviço de qualidade com o menor custo.

1.1 Justificativa

Técnicas atuais do estado da arte tem focado em desenvolver modelos para predição de tempo e recursos de *hardware* demandados por consultas baseando-se em operadores básicos do SGBDR, como *JOIN*, *SCAN*, *SORT*, entre outros, ou analisando as declarações das consultas em SQL, contando o número de palavras chave que podem impactar no desempenho das consultas, como *SELECT* e *GROUP BY*.

Contudo, o operador de junção por similaridade é fundamentalmente diferente dos operadores encontrados em SGBDRs, o que torna inviável utilizar tais abordagens para prever o tempo de execução de operações de junção por similaridade.

1.2 Contribuições deste Trabalho

1. Inédito: até o presente momento, não sabe-se de trabalhos anteriores investigando o tempo de execução de consultas de junções baseadas no conceito de similaridade, o que, devido a rápida expansão das tecnologias de computação em nuvem, é cada vez mais crucial em sistemas de informação que lidam com grande volume de dados.

2. *Features* utilizadas: neste trabalho foram desenvolvidos modelos capazes de capturar o comportamento do operador de junção por similaridade apenas com informações estatísticas dos dados, que estão disponíveis antes da execução da operação de junção por similaridade.

1.3 Objetivos

O objetivo deste trabalho é deselvolver um modelo de predição capaz de determinar o tempo de execução de operações de junção por similaridade em banco de dados utilizando técnicas de aprendizagem de máquina.

Objetivos Específicos:

As atividades envolvidas no escopo deste projeto de pesquisa para atingir o objetivo geral são listadas abaixo:

- Revisão de literatura em similaridade e, em particular, em algoritmos especializados para junção por similaridade baseada em conjuntos (PPJOIN e MPJOIN);
- Revisão de literatura em aprendizagem de máquina e, principalmente, técnicas de regressão;
- Criação de um ambiente de testes para geração de dados sintéticos e semi-sintéticos;
- Determinação das características (*features*) dos dados gerados que serão extraídas para a construção do modelo;
- Construção de modelos de regressão utilizando Árvores de Decisão M5P, Redes Neurais Artificiais e técnicas de *Boosting*;
- Validação dos resultados com dados reais utilizando métricas baseadas em erros de predição;
- Elaboração de um *benchmark* entre os modelos obtidos.

1.4 Estrutura do Documento

Na Seção 2, *Referencial Teórico*, são abordados conceitos importantes para a compreenção deste trabalho. Na Seção 3, *Metodologia*, são explicadas como foram realizadas as atividades envolvidas no escopo deste projeto. A Seção 4, *Resultados e Discussão*, ilustra, por meio de gráficos e discussões, os resultados obtidos nesta pesquisa. Finalmente, a Seção 5, *Conclusões e Trabalhos Futuros*, traz quais foram as conclusões obtidas neste trabalho.

2 REFERENCIAL TEÓRICO

Nesta seção são abordados conceitos fundamentais para a compreensão desta pesquisa, assim como tendências atuais da literatura em relação ao tema estudado.

2.1 Similaridade em Banco de Dados

O conceito de similaridade pode ser utilizado em diferentes áreas de pesquisa, sendo que a intuição mais comum é considerar que uma entidade é similar a outra caso ambas apresentem características (atributos) em comum (TVERSKY, 1977). Contudo, existem outras noções de similaridade, como o caso da similaridade baseada na distância Euclidiana, na qual entidades com atributos diferentes podem ser similares em casos que, em uma representação gráfica dos atributos de tais entidades, a distância entre os mesmos seja pequena. Em algumas aplicações, o conceito de similaridade pode ser utilizado para identificação de imagens, busca de perfis de usuário similares, entre outras. Porém, neste trabalho o conceito de similaridade está relacionado a similaridade textual, onde o foco é similaridade entre palavras.

Diante disso, uma *função de similaridade* tem a finalidade de quantificar o grau de semelhança entre objetos de um banco de dados. Em grande parte das implementações de funções de similaridade, dois atributos textuais são comparados e um valor entre 0 e 1 é retornado, indicando a similaridade entre os mesmos. Valores próximos de 1 indicam maior similaridade enquanto valores próximos de 0 indicam baixa similaridade.

De acordo com a aplicação, uma função de similariade pode ser mais adequada do que outra. Diante disso, duas classes de funções de similaridade amplamente utilizadas, devido a qualidade de seus resultados sobre uma grande va-

riedade de aplicações, são as funções de similaridade baseadas em distância de edição e as baseadas em conjuntos de *tokens*.

As funções baseadas em distância de edição identificam a similaridade entre palavras indicando o número mínimo de operações como substituição, deleção, inserção ou troca de caracteres, necessárias para transformar uma sequência de caracteres *A* em outra sequência de caracteres *B* (GRAVANO *et al.*, 2001).

Já as funções de similaridade baseadas em conjuntos de *tokens* utilizam de operações de conjuntos, tais como união, interseção e módulo, para medir a similaridade entre as entidades. As funções de similaridade baseadas em conjuntos, além de possuir uma qualidade de resultados comparável aos de funções por distância de edição, são mais eficientes do que funções que utilizam tal abordagem (RIBEIRO; HÄRDER, 2011).

No restante deste documento iremos utilizar o termo função de similaridade para denotar funções de similariade textuais baseadas em conjuntos de *tokens*.

2.2 Similaridade Baseada em Conjuntos

Nesta seção serão abordadas algumas das funções de similaridade baseadas em conjuntos de *tokens* mais utilizadas. A seguir, será explicado como é feito o pré-processamento dos textos para transformá-los em conjuntos de *tokens*.

2.2.1 Tokenização

O processo de *tokenização* consiste em converter a representação textual de uma entidade para um conjunto de tokens. Dessa forma, neste trabalho o termo tokens denota a unidade básica e indivisível de informação textual.

Por exemplo, um simples processo de tokenização consiste em mapear palavras separadas por caracteres especiais (como espaços em branco, hífens, pontos ou vírgulas) em tokens. Por exemplo, a palavra "Diego Sarmento Mendes" poderia ser transformada no conjunto de tokens { Diego, Sarmento, Mendes } utilizando tal estratégia.

Outro processo amplamente utilizado consiste no uso de *q-grams* como tokens (UKKONEN, 1992). *Q-grams* são subpalavras que podem ser geradas pelo deslocamento de uma janela de tamanho q nos caracteres da palavra original. Por exemplo, a string "DiegoMendes" pode ser transformada nos conjunto de tokens:

- {Di, ie, eg, go, oM, Me, en, nd, de, es}, considerando $q = 2$.
- {Die, ieg, ego, goM, oMe, Men, end, nde, des}, considerando $q = 3$;
- {Dieg, iego, egoM, goMe, oMen, Mend, ende, ndes}, considerando $q = 4$.

2.3 *Inverse Document Frequency* (IDF)

A intuição da medida *Inverse Document Frequency* (IDF) é que *tokens* que ocorrem em muitos documentos são menos determinantes para discriminar palavras (ROBERTSON; JONES, 1976). Assim sendo, *tokens* frequentes devem receber pesos menores do que aqueles que ocorrem poucas vezes na coleção. Desta forma, se duas palavras compartilham um *token* muito raro, a chance de tais palavras serem similares é bem maior do que se as mesmas tivessem apenas *tokens* muito frequentes em comum.

A medida IDF é uma estratégia amplamente utilizada na área de recuperação de informação, na qual pesos são atribuídos a documentos (*tokens*) baseando-se no número de vezes que os mesmos aparecem em uma coleção. O peso *IDF* de um *token* pode ser calculado utilizando a seguinte fórmula:

$$w(t_i) = 1 + \log \frac{N}{n_i} \tag{2.1}$$

Onde N é o número total de documentos presentes na coleção e n_i é o número de vezes que o token t_i aparece na coleção.

No restante deste documento, quando for utilizado o termo *tamanho* de um conjunto de *tokens*, entende-se por tamanho o somatório dos pesos individuais dos *tokens* contidos neste conjunto e, quando for utilizado o termo *cardinalidade* de um conjunto, estamos nos referindo ao número de *tokens* presentes no conjunto. Quando nenhuma estratégia de ponderação de *tokens* é utilizada, todos os *tokens* recebem um peso unitário e, neste caso, o tamanho de um conjunto é igual a sua respectiva cardinalidade (número de tokens que o conjunto possui). De maneira geral, o tamanho de um conjunto de *tokens x* é definido como:

$$tam(x) = \sum_{t \in x} w(t) \tag{2.2}$$

Onde $w(t_i)$ é o peso do token i.

2.4 Funções de Similaridade Baseadas em Conjuntos

Quando as entidades comparadas pela função de similaridade são mapeadas para um conjunto de subpalavras, denominadas *tokens*, as funções de similaridade determinam a similaridade entre as entidades comparadas por meio de operações de conjuntos (tais como união, interseção e módulo) sobre tais conjuntos de tokens. Tal classe de funções de similaridade compõe as *Funções de Similaridade Baseadas em Conjuntos*.

Na maior parte destas funções, são consideradas similares entidades em que o resultado da interseção de seus conjuntos de tokens retorna grande parte de seus

elementos. Na Tabela 2.1 são ilustradas as fórmulas de algumas das principais funções de similaridade baseadas em conjuntos, onde x_1 e x_2 são conjuntos de tokens e γ é o *threshold* fornecido pelo usuário (RIBEIRO; HÄRDER, 2011).

Tabela 2.1: Definição das Funções de Similaridade Baseadas em Conjunto (RIBEIRO; HÄRDER, 2011)

Função	Definição	$minoverlap(x_1, x_2)$	$[minsize(x), maxsize(x)]$
Jaccard	$\frac{tam(x_1 \cap x_2)}{tam(x_1 \cup x_2)}$	$\frac{\gamma}{1+\gamma}(tam(x_1)+tam(x_2))$	$[\gamma * tam(x), \frac{tam(x)}{\gamma}]$
Dice	$\frac{2*tam(x_1 \cap x_2)}{tam(x_1)+tam(x_2}$	$\frac{\gamma*(tam(x_1)+tam(x_2))}{2}$	$[\frac{\gamma*tam(x)}{2-\gamma}, \frac{(2-\gamma)*tam(x)}{\gamma}]$
Cosine	$\frac{tam(x_1 \cap x_2)}{\sqrt{tam(x_1)*tam(x_2)}}$	$\gamma\sqrt{tam(x_1)*tam(x_2)}$	$[\gamma^2 * tam(x), \frac{tam(x)}{\gamma^2}]$

Por simplicidade, no restante deste documento será utilizado o termo função de similaridade para referir-se a função de similaridade Jaccard. Entretanto, todas as técnicas que serão descritas são diretamente aplicáveis a qualquer uma das funções de similaridade descritas na Tabela 2.1.

2.5 Junção por Similaridade

Dados uma coleção de registros de alguma base de dados, uma função de similaridade *sim*, e um limiar (*threshold*) γ definido pelo usuário, uma Junção por Similaridade (JS) consiste em encontrar todos os pares de tuplas, $< x, y >$, que possuam similariade maior ou igual ao threshold γ, ou seja, $sim(x,y) \geq \gamma$ (XIAO *et al.*, 2011).

Uma abordagem direta para encontrar duplicatas textuais é, inicialmente, fazer um mapeamento de todas palavras em uma base de dados para um conjunto de tokens, conforme ilustrado na Figura 2.1 (considerando *tokens* com peso unitário).

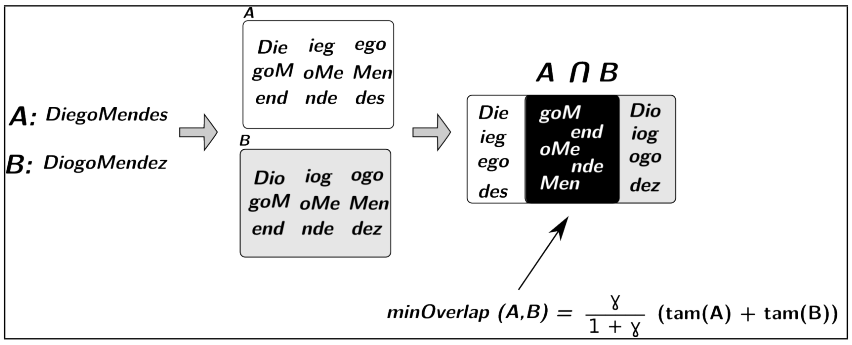


Figura 2.1: Passo a passo da junção por similaridade baseada em conjuntos.

Depois de gerados os conjuntos de tokens de cada palavra, o problema de mesurar a similaridade de dois textos é reduzido para um problema de interseção de conjuntos (RIBEIRO; HÄRDER, 2011). Ou seja, dado um threshold t, o problema de indicar a similaridade entre dois textos é convertido para o problema de contar o número mínimo de elementos em comum que os dois conjuntos de tokens possuem, informação esta que é calculada utilizando a fórmula do *minOverlap*, na Tabela 2.1. Este processo é ilustrado na Figura 2.1.

Desta forma, são comparados todos os pares de tuplas de duas base de dados, retornando pares que possuam uma similaridade *Jaccard* maior que o *threshold* definido pelo usuário. Contudo, esta abordagem possui um custo computacional proibitivamente alto quando as bases de dados possuem muitos registros, uma vez que o número total de comparações feitas é $O(n^2)$ (XIAO *et al.*, 2011).

2.6 Algoritmo *Min-Prefix Similarity Join* (MPJOIN)

O algoritmo de junção por similaridade *Min-Prefix Similarity Join* (MPJOIN) (RIBEIRO; HÄRDER, 2011) é uma generalização do *Prefix Filtering* do algoritmo PPJOIN e possui resultados eficientes mesmo para grandes volumes de dados. Tal abordagem utiliza a divisão da junção por similaridade em três etapas, onde uma

destas é uma fase de Indexação, a segunda etapa corresponde a geração de candidatos e, por último, há uma etapa de verificação dos candidatos remanescentes, conforme detalhado a seguir:

- Indexação: etapa onde todos os *tokens* são indexados em uma lista invertida, que é utilizada para mapear cada token t para uma lista de identificadores de registros que possuem t. Assim, pode-se facilmente analisar um token t e todas as tuplas que possuem tal token e, combinando-se estas tuplas, gera-se os primeiros pares de candidatos;
- Geração de candidatos: etapa onde os primeiros pares candidatos a duplicatas gerados na fase de indexação são obtidos e uma série de filtros são aplicados sobre estes candidatos. Mais detalhes sobre estes filtros serão abordados em breve;
- Verificação: Os pares de candidatos remanescentes da etapa de geração de candidatos são passados para esta fase, na qual finalmente a função de similaridade é aplicada. Apenas os pares que atendam uma similaridade maior que o *threshold* passado pelo usuário são enviadas para a saída do algoritmo.

Desta forma, o algoritmo evita a comparação entre todos os pares de tuplas da base de dados alvo da junção, uma vez que são utilizados filtros na etapa de geração de candidatos. Esta estratégia é adotada por diversas técnicas presentes na literatura, tais como o PPJOIN e o PPJOIN+ (SARAWAGI; KIRPAL, 2004; BAYARDO; MA; SRIKANT, 2007; XIAO *et al.*, 2011). A seguir, são detalhados os filtros por tamanho e por prefixo, que são utilizados pelo algoritmo MPJOIN.

2.6.1 *Size Filtering*

O filtro por tamanho (*size filtering*) parte do conceito que palavras que possuam uma alta similaridade não diferem muito em seus respectivos tamanhos. Assim, é

possível remover pares de palavras dos candidatos caso seus respectivos tamanhos se diferam mais do que um limite, o qual é determinado de acordo com o valor do *threshold* da função de similaridade. A Tabela 2.1 mostra como é possível determinar o tamanho mínimo e máximo que duas palavras devem possuir para possuir uma similaridade γ de acordo com a função de similaridade aplicada (RIBEIRO; HÄRDER, 2011).

Considere um exemplo onde a palavra "DiegoMendes" foi mapeada para um conjunto de *3-grams*, gerando um conjunto

$$x = \{Die, ieg, ego, goM, oMe, Men, end, nde, des\},$$

onde cada *token* possui um peso unitário. Para que uma outra palavra qualquer p tenha uma similaridade *Jaccard* igual a $0,7$ com a palavra "DiegoMendes", o conjunto de *3-grams* gerado após uma tokenização de p não pode possuir um tamanho menor que 7 e seu tamanho máximo não pode ser maior que 12. Estes cálculos foram feitos utilizando as fórmulas *minsize* e *maxsize* presentes na Tabela 2.1, conforme detalhado a seguir:

$$minsize(x) = \lceil 0,7 * tam(x) \rceil = 7 \tag{2.3}$$

$$maxsize(x) = \left\lfloor \frac{tam(x)}{0,7} \right\rfloor = 12 \tag{2.4}$$

Desta forma, evita-se que palavras cujos respectivos tamanhos se difiram substancialmente sejam comparadas, reduzindo o número de candidatos a serem verificados.

Caso seja feita uma ordenação dos conjuntos pelos seus respectivos tamanhos, é possível identificar palavras de tamanhos similares de forma mais eficiente.

2.6.2 *Prefix Filtering*

O filtro por prefixo (*prefix filtering*) consiste em calcular de uma janela na qual o prefixo, ou seja, os tokens iniciais de dois conjuntos de tokens, são comparados, sendo que, caso não exista nenhum token em comum dentro deste prefixo mínimo, não há nenhuma possibilidade de tais conjuntos possuírem uma similaridade maior que o limiar de similaridade (*threshold*) definido pelo usuário. O tamanho desta janela é calculado em função do *threshold* de similaridade e do tamanho dos conjuntos. O cálculo desta janela é feito da seguinte forma:

$$prefSize(x_i) = tam(x_i) - minOverlap(x_i, x_j) + 1 \tag{2.5}$$

Desta forma, considere dois conjuntos x_1 e x_2 de tamanho 10 que, por simplicidade, não possuem ponderação de *tokens* e estão ordenados por um mesmo critério de ordenação O. Para que x_1 possua similaridade de pelo menos 0,7 com x_2, sabe-se que os mesmos deverão possuir pelo menos $minoverlap(x_1, x_2) = 7$ *tokens* em comum (cálculo feito utilizando a Tabela 2.1). Assim, podemos construir uma janela com tamanho $10 - 7 + 1 = 4$, sendo que se não houver nenhum token em comum entre tais palavras de tamanho 10 dentro de seus prefixos de tamanho 4, não há nenhuma possibilidade de atender o limiar de similaridade passado pelo usuário, descartando então este par de conjuntos de tokens da saída do algoritmo.

Caso seja adotada uma ordenação dos *tokens* de acordo com seus respectivos pesos *IDF*, colocando *tokens* mais raros no prefixo dos conjuntos de *tokens* do *dataset*, a eficiência deste tipo de filtro aumenta significativamente, uma vez que menos candidatos irão possuir *tokens* raros em comum em seus respectivos prefixos. Desta forma, a performance do MPJOIN é superior quando utiliza-se dados com *tokens* ponderados.

2.7 Pseudo-código do MPJOIN

O Algoritmo 2.1 traz, em alto nível, as ideias centrais da implementação da junção por similaridade utilizando o Min-Prefix Similarity Join (MPJOIN).

Algorithm 2.1: Algoritmo Min-Prefix Similarity Join

Data: Coleção de Conjuntos de *tokens* ordenados C, $threshold$

Result: Lista de Pares de Conjuntos $< x, y >$, tal que $sim(x, y) \geq threshold$

```
1 foreach x_i ∈ C do
2     Candidates ← PrefixFiltering(x_i);
3     Candidates ← SizeFiltering(Candidates, threshold);
4     Candidates ← Verification(Candidates, threshold);
5     Index(Candidates, L);
6 end
7 return L
```

Como pode-se notar, o algoritmo recebe dois parâmetros: uma coleção de C conjuntos de *tokens* ordenados por tamanho e por frequência, e um limiar de similaridade (*threshold*).

Na Linha 1, é feita uma iteração sobre todos os conjuntos de *tokens* x_i pertencentes à coleção C, e, em seguida, na Linha 2, para cada conjunto x_i é aplicado o filtro por prefixo (*prefix filtering*), fazendo com que a lista de candidatos receba todos os conjuntos de *tokens* que possuem pelo menos um *token* em comum com x_i em seus respectivos prefixos.

Após esta etapa, na Linha 3 o filtro por tamanho (*size filtering*) é aplicado sobre os primeiros candidatos gerados, eliminando aqueles que possuem tamanhos muito distintos de x_i.

Em seguida, na Linha 4, uma verificação é feita sobre os candidatos remanescentes, utilizando uma função de similaridade baseada em conjuntos. Os pares de

candidatos que passaram pelos filtros e pela verificação são, na linha 5, indexados em uma lista invertida L, e o processo é repetido até que todos os conjuntos de *tokens* de C sejam analisados.

Ao final, na linha 7, a lista invertida L, que possui todos pares de candidatos que atenderam o limiar de similaridade definido pelo usuário, é enviada para a saída do algoritmo.

2.8 Aprendizagem de Máquina

A aprendizagem de máquina, do inglês *machine learning*, é uma área de estudos da ciência da computação que busca fazer com que computadores simulem um comportamento inteligente, tornando-os capazes de aprender a partir de experiências anteriores e identificar padrões usando características nunca vistas antes, assim como o cérebro humano faz.

As técnicas de aprendizagem de máquina são divididas em três categorias:

- Aprendizagem supervisionada;
- Aprendizagem não-supervisionada;
- Aprendizagem por reforço.

Nas técnicas de aprendizagem supervisionada, dados de um determinado problema que já foram rotulados previamente por um especialista são fornecidos para os algoritmos de aprendizagem, os quais extraem informações importantes destes dados para criar modelos capazes de atribuir rótulos de classe a dados desconhecidos.

Já nas técnicas não-supervisionadas, não existem dados previamente rotulados para que os algoritmos criem modelos capazes de rotular os dados. Tais técnicas

buscam extrair informações baseando-se na similaridade que os dados desconhecidos possuem entre si, agrupando dados similares em uma mesma classe. Tais técnicas são amplamente utilizadas em tarefas de agrupamento (*clustering*).

Outra categoria de técnicas de aprendizagem é a por reforço, a qual busca criar modelos a partir de dados que, assim como nas técnicas não-supervisionadas, não possuem uma rotulação prévia. A diferença desta abordagem para a anterior é que um especialista no problema faz, iterativamente, avaliações do modelo criado, penalizando o mesmo caso não apresente bons resultados e beneficiando caso os resultados sejam bons.

Neste trabalho serão apenas utilizadas técnicas de aprendizagem de máquina supervisionadas.

2.8.1 Regressão vs Classificação

A principal diferença entre regressão e classificação está na Imagem da função da qual se deseja aproximar, sendo que em tarefas de classificação a imagem da função que se deseja aproximar é um conjunto finito de rótulos de classe, enquanto que em tarefas de regressão a imagem da função que se deseja aproximar é numérica, podendo assumir infinitos valores.

Exemplos de aplicações que utilizam regressão são: previsão de temperatura; determinar o tamanho que uma árvore irá alcançar ao atingir a fase adulta; prever qual a quantidade de energia que uma usina hidrelétrica deverá produzir para atender uma cidade em um determinado dia; prever qual será a quantidade de produtos vendidos em um determinado mês por uma loja, entre outras.

Como os modelos que serão utilizados neste trabalho irão prever o tempo de execução, o foco deste trabalho serão técnicas de regressão, uma vez que a variável envolvida (tempo) é numérica.

2.8.2 Árvores de Decisão

As árvores de decisão são métodos que utlizam a abordagem de divisão e conquista para resolver problemas de aprendizagem de máquina (WITTEN; FRANK, 2005), criando ramificações que separam diferentes entidades baseando nos valores de atributos (atributos teste), conforme é ilustrado na Figura 2.2. Para classificar uma instância cujo rótulo é desconhecido, basta atribuir o rótulo deste de acordo com o presente no nó folha alcançado ao serem avaliados seus respectivos atributos utilizando a árvore construída.

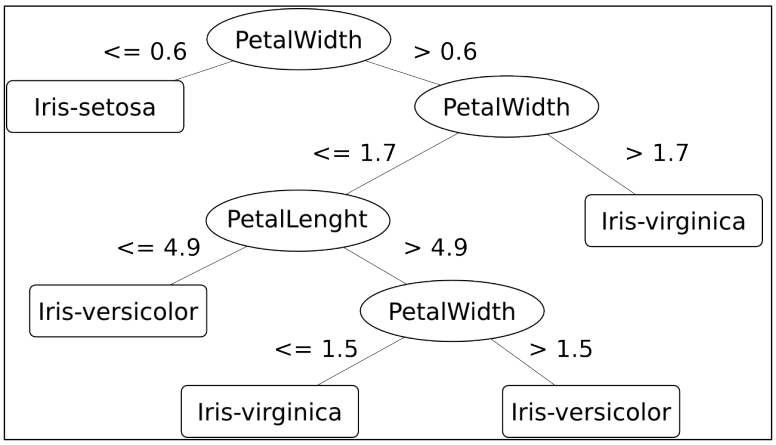


Figura 2.2: Árvore de decisão para classificação de diferentes espécies de plantas

A construção da árvore de decisão, que também é chamada de indução da árvore, é feita com base em dados de treinamento previamente rotulados. Desta forma, busca-se escolher atributos que melhor dividem os dados como raíz da árvore, repetindo o processo recursivamente às sub-árvores, criando modelos compactos e que melhor representam os dados. Tal tarefa é feita, por exemplo, utilizando informações de ganho de informação e entropia. Alguns algoritmos famosos para indução de árvore de desição são ID3, CART, C4.5 e J48 (TAN; STEINBACH; KUMAR, 2005).

Para evitar erros de generalização devido a *overfitting*, os algoritmos de indução de ávores de decisão utilizam estratégias de poda, que evitam que ruídos nos dados de treinamento prejudiquem o modelo criado em tarefas de classificação de dados desconhecidos.

2.8.3 Árvores de Decisão para Regressão (*Regression Trees*)

Quando torna-se necessário prever dados numéricos ao invés de categorias, o mesmo tipo de ávore de decisão pode ser utilizado, entretanto os nós folhas vão possuir o valor médio de todos os valores dos dados de treinamento que se aplicam a este nó folha (WITTEN; FRANK, 2005). Como o que está sendo previsto neste tipo de árvore são dados numéricos, árvores de decisão que possuem valores médios em seus respectivos nós folha são chamadas de árvores de decisão para regressão (*regression trees*).

Outra estratégia adotada para tornar as decisões numéricas ainda mais precisas consiste em combinar equações de regressão com as árvores de decisão para regressão, sendo que ao invés de possuir uma média dos valores nos nós folha da árvore, cada folha possui expressões lineares, que são equações de regressão (WITTEN; FRANK, 2005). Árvores de decisão que adotam esta estratégia são chamadas de árvore de decisão baseadas em modelo (*model trees*).

2.8.4 Árvores de Decisão M5P

As árvores de decisão M5P utilizam o sistema M5 de aprendizagem de máquina, introduzido em (QUINLAN, 1992), para construção de modelos baseados em árvores de decisão. Os modelos são similares aos gerados por outros sistemas de indução de árvore, como o CART por exemplo, 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 (ou seja, dados com muitas *features*). (QUINLAN, 1992).

2.8.5 Redes Neurais Artificiais

Técnicas de aprendizagem de máquina por redes neurais artificiais (RNAs) buscam simular tarefas de aprendizagem com o uso de neurônios artificiais inspirados nos neurônios naturais. Dentre os modelos de neurônios artificiais mais utilizados, se destacam o *Perceptron* de Rosenblatt e o *Adaline*. As RNAs podem ser utilizadas para resolver problemas de classificação e regressão, sendo amplamente utilizadas em tarefas de reconhecimento de padrões em imagens (BISHOP, 1996). Existem diversas formas de organizar os neurônios em forma de uma rede, sendo que esta organização pode variar de acordo com a complexidade do problema a ser resolvido. Basicamente, uma rede neural faz uma combinação linear dos dados de entrada, os quais são multiplicados por seus respectivos pesos, que são determinados na etapa de treinamento da rede, e, em seguida, podem ser transformados por uma função de ativação, caso seja utilizado neurônios do tipo *Perceptron*.

2.8.6 Redes Neurais de Múltiplas Camadas (*Multi-layer Perceptron*)

Redes neurais que possuem apenas um neurônio ou somente uma camada (*layer*) de neurônios possuem limitações no que se diz respeito ao número de funções que tais redes podem aproximar. Assim sendo, quando as redes são organizadas de tal forma a conter pelo menos duas camadas escondidas, conforme ilustrado na Figura 2.3, a rede torna-se capaz de aproximar qualquer tipo de função (BISHOP, 1996). Tais tipos de redes possuindo duas ou mais camadas são comumente chamadas de redes de *Perceptrons* de múltiplas camadas (*Multi-layer Perceptron*, ou apenas MLP), mesmo que o modelo de neurônio utilizado não seja o *Perceptron*.

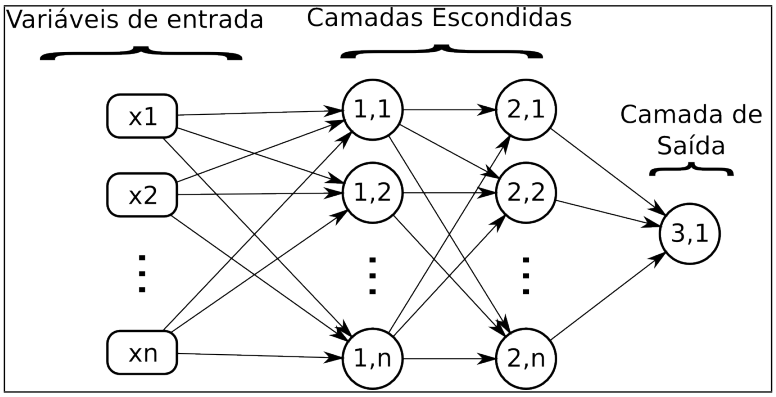


Figura 2.3: Organização de rede de neurônios com duas camadas escondidas.

Durante o treinamento de uma MLP são encontrados todos os pesos das entradas de cada neurônio que compõe a rede. Como os pesos são valores contínuos, encontrar os pesos que melhor se adaptam aos dados de entrada de forma a minizar o erro da rede neural é uma tarefa extremamente complexa. Uma técnica amplamente utilizada para encontrar o conjunto de pesos de forma eficiente é a *backpropagation*, que calcula os pesos baseando-se no erro obtido na saída, o qual é passado no sentido contrário aos dados de entrada para que as camadas intermediárias da rede possam recalcular seus respectivos pesos. A ideia desta técnica é corrigir o conjunto de pesos as entradas dos neurônios na direção contrária ao gradiente da função erro da rede sobre os dados de treinamento.

2.9 *Ensemble Learning*

Ensemble learning é uma técnica de aprendizagem de máquina que busca combinar um conjunto de modelos para criar novos mais precisos. Fazendo uma analogia com o mundo real, quando temos que tomar decisões difíceis, podemos consultar diversos especialistas antes de tomar alguma decisão. Desta forma, em aprendizagem de máquina podemos consultar diversos modelos antes de classificar ou prever um valor numérico de uma instância.

Dentre as técnicas de ensemble mais conhecidas, se destacam as técnicas de *bagging*, randomização e *boosting* (WITTEN; FRANK, 2005; TAN; STEINBACH; KUMAR, 2005). Neste trabalho será utilizado apenas a técnica *boosting* de *ensemble learning*.

2.9.1 *Boosting*

O método *boosting* é um processo iterativo de construção de modelos de aprendizagem de máquina, onde modelos novos são criados baseados na performance de modelos criados anteriormente (WITTEN; FRANK, 2005). A técnica de *boosting* busca criar, iterativamente, modelos que focam nas instâncias que foram mais difíceis de serem classificadas pelos modelos anteriores, ponderando cada modelo gerado de acordo com sua confiança, diferentemente de outras técnicas de *ensemble* que consideram que cada modelo possui o mesmo peso na decisão final.

As técnicas de *boosting* são conhecidas por apresentarem bons resultados mesmo utilizando diferentes tipos de classificadores, característica esta que atrai pesquisadores de diversas áreas de pesquisa a usarem tal abordagem para resolver diversos problemas aproveitando modelos de classificação e regressão já existentes.

2.9.2 Regressão Aditiva (*Additive Regression*)

Quando a técnica de *boosting* é utilizada para prever dados numéricos, tal técnica é denominada regressão aditiva (*additive regression*) (WITTEN; FRANK, 2005). O termo *regressão aditiva* foi escolhido porque pesquisadores da área da estatística descobriram que as técnicas de *boosting* eram apropriadas para construir um modelo aditivo, baseando-se em ideias da estatística que utilizam tal abordagem. Esta técnica ficou conhecida como *forward stagewise additive modeling*, que parte

de um comitê (*ensemble*) vazio de regressores, e, aditivamente, vai acrescentando novos modelos ao comitê. Conforme novos modelos vão sendo acrescentandos ao conjunto, a performance da predição é melhorada sem alterar os modelos que já estão no *ensemble* (WITTEN; FRANK, 2005; TAN; STEINBACH; KUMAR, 2005). Para melhorar a performance do conjunto de regressores, o novo modelo a ser adicionado dá uma atenção especial aos dados de treino nos quais o comitê de regressores obteve piores resultados, dando a tais instâncias um peso maior em etapas de treinamento.

2.10 Validação Cruzada

A validação cruzada, também chamada de *cross-validation*, é uma técnica amplamente utilizada para prever a capacidade de generalização de modelos criados por técnicas de aprendizagem de máquina. A técnica consiste em dividir os dados de treinamento em partições denominadas *folds*, conforme ilustrado na Figura 2.4, as quais são utilizadas iterativamente para treinamento e teste do modelo. Inicialmente, escolhe-se um *fold* para ser utilizado para testar o modelo, e utiliza-se todos os outros para treinamento, avaliando qual foi o erro do modelo para este *fold*. Em etapas seguintes, outro *fold* é escolhido para teste, e os dados utilizados para treinamento são todos os *folds*, exceto o que foi escolhido para teste. O proceso é repetido até que todos os *folds* sejam utilizados para teste e treino, sendo que o erro de generalização do modelo é calculado pela média dos erros obtidos em cada *fold*.

Desta forma, é possível avaliar o comportamento do modelo criado para cada instância dos dados de treinamento, garantindo que os dados que foram utilizados em etapas de treinamento não estejam nos dados de teste a cada iteração, dando uma ideia da capacidade de generalização do modelo criado para dados desconhecidos.

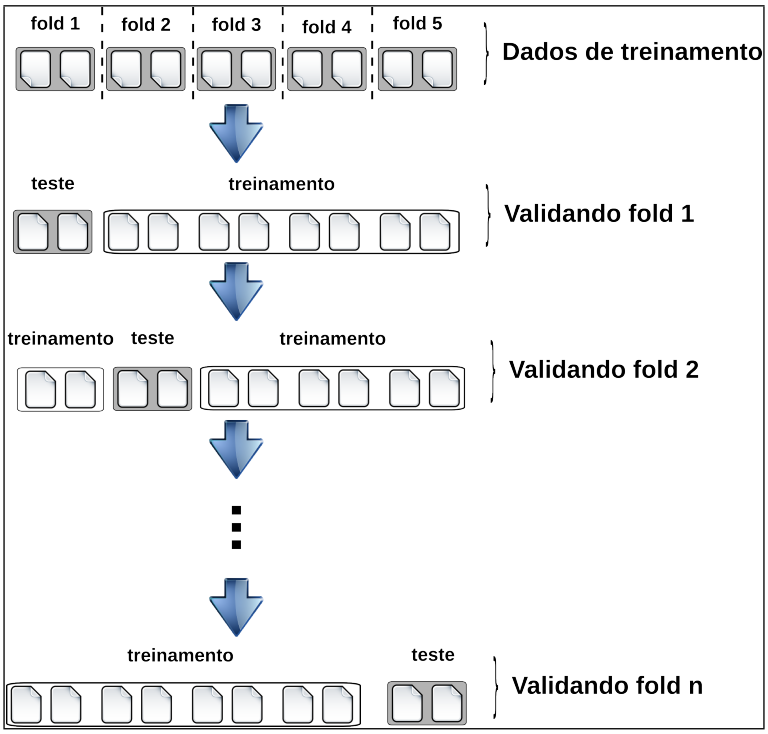


Figura 2.4: Técnica de validação cruzada.

2.11 Trabalhos Correlatos

Trabalhos atuais têm desenvolvido modelos que, por meio de informações disponíveis nos SGBDRs, são capazes de prever o tempo de execução de consultas ao banco de dados. Em pesquisas iniciais, mostrou-se que técnicas utilizando diretamente os custos indicados pelo otimizador de consultas do SGBDR PostgreSQL não se mostraram muito precisas (GANAPATHI *et al.*, 2009). Contudo, em trabalhos recentes (WU *et al.*, 2013), ajustes finos nos parâmetros de configuração do otimizador de consultas do PostgreSQL foram realizados, assim como foi feita uma correlação em estimativas de cardinalidades, obtendo resultados mais precisos.

Uma abordagem, utilizada em (GANAPATHI *et al.*, 2009), (AKDERE *et al.*, 2012) e (LI *et al.*, 2012), faz uso de técnicas aprendizagem de máquina para criação de modelos. Assim sendo, em (GANAPATHI *et al.*, 2009) utilizou-se apenas palavras chave contidas na declaração das consultas em SQL, como *SELECT*, *JOIN*, entre outras, para construção de modelos de predição de tempo e processamento que uma consulta irá demandar. Contudo, tal abordagem não obteve resultados eficientes.

Outra estratégia explorada em (GANAPATHI *et al.*, 2009) e (AKDERE *et al.*, 2012), que também utiliza-se de técnicas de *machine learning*, busca criar modelos individuais para os operadores básicos do SGBDR. Desta forma, ao obter a árvore de operadores gerada pelo otimizador de consultas do SGBDR, modelos específicos para cada operador estimam o tempo combinando a número de vezes que o operador é utilizado e a cardinalidade que cada um é submetido. Assim, combinando as predições para cada operador, é possível fazer uma estimação precisa do tempo e processamento demandado por uma consulta.

Em (LI *et al.*, 2012), técnicas de escalonamento foram adotadas para criar modelos capazes de prever o tempo de execução de consultas em casos que os dados de treinamento se diferem muito dos dados de teste. Desta forma, para cada operador básico dos SGBDRs foi definida uma função de escalonamento específica, o que foi possível a partir do pré-conhecimento do comportamento assintótico destes operadores.

3 METODOLOGIA

Para a execução dos experimentos deste trabalho foi utilizada uma implementação do algoritmo MPJOIN, que faz a junção por similaridade baseando-se em conjuntos. O objetivo dos experimentos foi realizar uma série de execuções do MPJOIN sobre diferentes bases de dados, extraíndo o tempo de execução e caracteríscas destes conjuntos de dados em cada execução, conforme ilustrado na Figura 3.1. A partir destes dados, foram utilizadas técnicas de aprendizagem de máquina para criar modelos de predição do tempo de execução deste algoritmo em diferentes *datasets*.

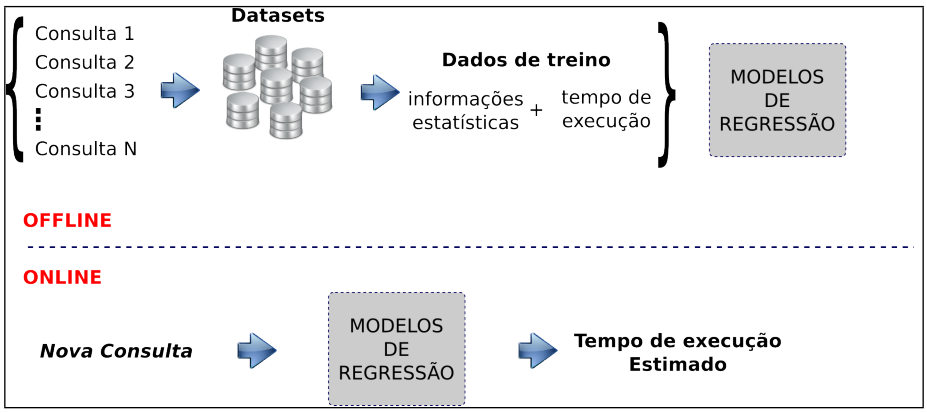


Figura 3.1: Ilustração resumindo os experimentos realizados.

3.1 *Hardware* Utilizado

Para a execução dos experimentos foi utilizado um servidor HP com 8GB de memória RAM e um processador Xeon 3.2 GHz com 4 núcleos (8 simulados), dedicado para fazer experimentos.

3.2 Determinação das Caracteríscas a Serem Extraídas

A partir do pré-conhecimento de estratégias de otimização do algoritmo MPJOIN, exploradas em (RIBEIRO; HÄRDER, 2011), notou-se que as seguintes caracteríscas impactavam significativamente no tempo de execução do MPJOIN:

- Distribuição da cardinalidade dos conjuntos;
- Distribuição do tamanho dos conjuntos;
- Distribuição da frequência dos *tokens*;
- *Threshold* de similaridade fornecido pelo usuário.

Além disso, foi possível notar que a execução do algoritmo difere substancialmente quando utiliza-se *tokens* ponderados utilizando o esquema de pesos *Inverse Document Frequency*. Isto ocorre porque a eficiência dos filtros presentes na etapa de geração de candidatos é substancialmente maior quando se utiliza algum esquema de ponderação de *tokens*, levando a necessidade de criar modelos distintos para casos em que utiliza-se tal abordagem de ponderação e casos em que nenhuma estratégia de ponderação é utilizada.

Relembrando que, nesta pesquisa, o termo cardinalidade de um conjunto se refere ao número de *tokens* que o mesmo possui, enquanto o tamanho de um conjunto é a soma dos pesos de cada *token* que compõe o conjunto, foram definidas:

- 8 *features* para construir modelos de regressão específicos para *datasets* sem nenhum esquema de peso;
- 11 *features* para *datasets* que adotam a estratégia de ponderação IDF.

Basicamente, utilizou-se informações de média, desvio padrão e assimetria dos *datasets* da cardinalidade dos conjuntos, da distribuição da frequência dos

tokens e, no caso dos *datasets* ponderados, informações do tamanho dos conjuntos de *tokens* presentes em cada *dataset*. Nos casos em que não são utilizados *tokens* ponderados, o tamanho e a cardinalidade de um conjunto são iguais, não sendo necessário repetir tais *features* nos dados de treinamento. A seguir, são detalhadas cada uma das features e a classe utilizadas para criação dos dados de treinamento:

1. `cardMean`: Média da cardinalidade dos conjuntos de tokens;
2. `cardStdDeviation`: Desvio padrão da cardinalidade dos conjuntos de tokens;
3. `cardSkewness`: *Skewness* (assimetria) da cardinalidade dos conjuntos de tokens;
4. `setSizeMean`: Tamanho médio dos tamanhos dos conjuntos (apenas para *datasets* com peso);
5. `setSizeStdDeviation`: Desvio Padrão dos tamanhos dos conjuntos (apenas para *datasets* com peso);
6. `setSizeSkewness`: *Skewness* (assimetria) dos tamanhos dos conjuntos normalizados (apenas para *datasets* com peso);
7. `tokenFreqMean`: Média da frequência dos tokens que compõem o dataset;
8. `tokenFreqStdDeviation`: Desvio padrão da frequência dos tokens que compõem o dataset;
9. `tokenFreqSkewness`: *Skewness* (assimetria) da frequência dos tokens que compõem o *dataset*;
10. `numOfUniqueTokens`: Número de tokens únicos que compõem o dataset;
11. `threshold`: limiar de similaridade passado como entrada do algoritmo (valor entre 0 e 1);

12. `execTime (Classe)`: Tempo que o algoritmo levou para executar a junção por similaridade sobre este dataset;

Todas as *features* descritas acima podem ser obtidas antes da execução do MPJOIN, uma vez que em uma etapa denominada criação dos *conjuntos ordenados* é feita uma passagem sobre o *dataset*, conforme ilustrado na Figura 3.2. Nesta etapa, que ocorre antes da execução da junção por similaridade, todos os conjuntos de *tokens* são ordenados por tamanho e, dentro de seus respectivos conjuntos, ordenados por suas frequências, tarefa que, conforme indicado em (BAYARDO; MA; SRIKANT, 2007), é compensada pelo melhor desempenho do algoritmo.

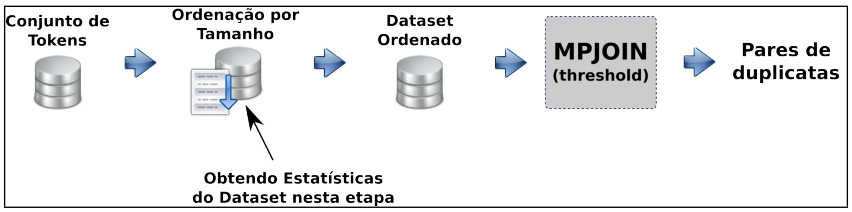


Figura 3.2: Obtendo features na etapa de ordenação dos datasets, antes da execução do MPJOIN

3.3 Criação dos *Datasets*

A tarefa de criação dos *datasets* foi feita utilizando um ambiente de geração de dados semi-sintéticos, no qual é possível extrair informações de um *dataset* estruturado em XML, criando amostras com um número determinado de conjuntos de tokens e duplicatas.

Os *datasets* gerados neste experimento foram retirados da base de dados em XML do DBLP [1] (*Digital Bibliography and Library Project*) e do IMDB[2] (*Internet Movie Database*), sendo que neste último foi necessário uma transformação para um documento XML.

[1]Disponível em `http://dblp.uni-trier.de/`

[2]Disponível em `http://www.imdb.com/`

Para gerar dados com uma grande variabilidade de tamanhos de conjuntos e diferentes distribuições de frequência de *tokens*, informações contidas nos *datasets* do IMDB e do DBLP foram combinadas de diferentes maneiras, utilizando diferentes formas de *tokenização* e números de duplicatas por amostra, conforme ilustrado na Figura 3.3. A geração de duplicatas é feita por uma injeção de erros textuais, como trocas, exclusões e inserções de caracteres nas amostras obtidas, sendo que a quantidade de erros e onde os mesmos serão aplicados são determinados de forma aleatória. Tal estratégia foi adotada buscando cobrir o maior número possíveis de casos de execução do MPJOIN, onde terão consultas de junção por similaridade que variam de tempos inferiores a 1 segundo a tempos superiores a 25 minutos. Em todos os casos, os *datasets* gerados possuem 100 mil tuplas.

Para cada combinação de atributos concatenados, tokenizador e número de duplicatas, foram feitas 5 execuções do gerador de dados, o que foi possível devido ao fato de que as amostragens são feitas de forma aleatória e, a cada vez que os erros aleatórios são aplicados a tais amostragens, diferentes versões dos *datasets* são geradas. Além disso, o processo foi repetido para criar *datasets* sem nenhum esquema de ponderação, e outros que utilizam o esquema de peso *Inverse Document Frequency*. Ao final da geração dos *datasets*, foram criados $concatenacoes * tokenizadores * duplicatas * esquemasdePeso * numerodeExecucoes = 5 * 4 * 3 * 2 * 5 = 600$ *datasets* distintos para cada fonte de dados, ou seja, 600 *datasets* baseados no IMDB e outros 600 para *datasets* baseados no DBLP, sendo metade destes sem nenhum esquema de peso e, a outra metade, utilizando o esquema de ponderação IDF.

3.4 Criação dos Dados de Treinamento

Para a criação dos dados de treinamento, foram feitas execuções do MPJOIN utilizando 5 valores de *threshold* para cada base de dados(300 *datasets* para o DBLP

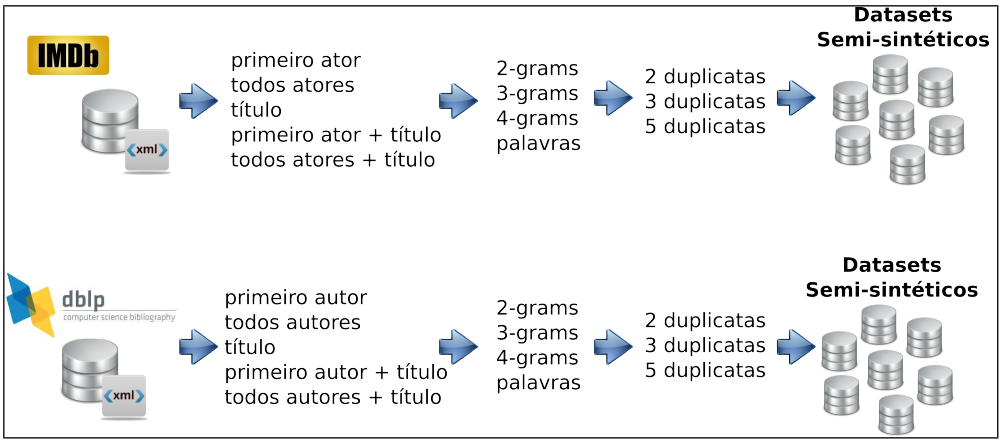


Figura 3.3: Criando datasets sujos para treinamento

sem pesos, 300 para o DBLP com IDF, 300 para o IMDB sem pesos e 300 para o IMDB com IDF), obtendo o tempo de execução e as *features* de cada *dataset*, conforme ilustrado na Figura 3.4. Buscando evitar ruídos nos dados de treinamento, o tempo de execução foi obtido a partir da média de 5 execuções do algoritmo sobre cada base de dados. Desta forma, ao final foram criados 4 arquivos de treinamento, sendo um para cada categoria de *dataset*, ou seja, DBLP sem ponderação de *tokens*, DBLP com ponderação de *tokens IDF*, IMDB sem ponderação de *tokens* e IMDB com podenração de *tokens IDF*.

Para criar dados de treinamento com mais instâncias, foram criados dois arquivos de treino adicionais:

- IMDB+DBLP sem pesos: criado combinando as instâncias dos arquivos de treino gerados a partir do IMDB e do DBLP que não utilizam ponderação de *tokens*, sendo composto por 3000 instâncias de execuções do MPJOIN;
- IMDB+DBLP com esquema de peso IDF: criado pela combinação de instâncias dos arquivos de treino gerados a partir do IMDB e DBLP que utilizam ponderação de *tokens* IDF.

, combinando as instâncias dos arquivos de treino do DBLP e do IMDB (um para dados sem peso, e outra para *datasets* ponderados).

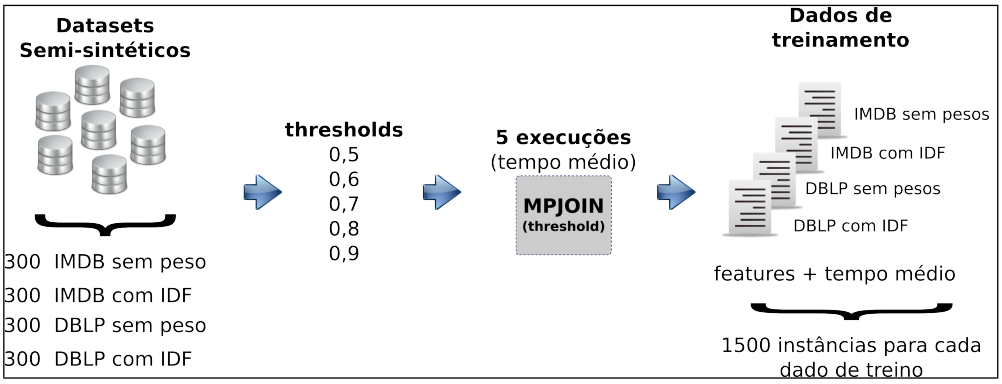


Figura 3.4: Criando dados de treinamento.

3.5 Ferramentas Utilizadas para a Criação dos Modelos

Para a criação dos modelos, foi utilizada a versão 3.7.9 da biblioteca *WEKA* (Waikato Environment for Knowledge Analysis), uma ferramenta *open-source* implementada em Java, que possui diversas técnicas de aprendizagem de máquina já implementadas. O *WEKA* pode ser utilizado tanto por meio de um ambiente gráfico, conforme ilustrado na Figura 3.5, ou integrado ao código em Java de um projeto do usuário.

3.6 Criação dos Modelos de Regressão

Depois de obter os dados de treinamento, conforme descrito na seção 3.4, foram criados modelos de regressão, utilizando a ferramenta *WEKA*, para cada categoria de dados gerados, conforme ilustrado na Figura 3.6. Foram criados modelos utilizando árvores de decisão M5P, redes neurais com múltiplas camadas (*Multi-Layer Perceptron*) e modelos utilizando a técnica de *boosting* para classes numéricas (re-

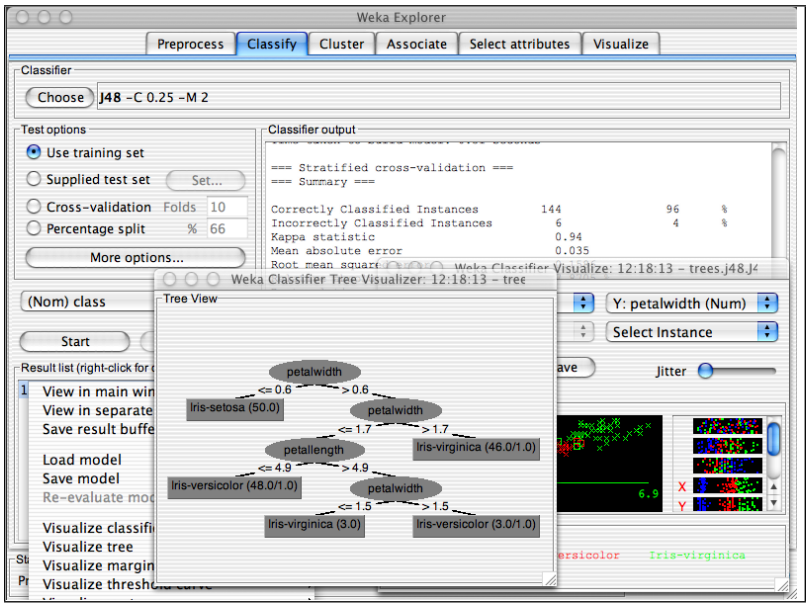


Figura 3.5: Interface gráfica do Weka

gressão aditiva). Os parâmetros de cada modelo foram definidos após uma série de tentativas, sendo que aqueles que apresentaram os melhores resultados foram os escolhidos para criação dos modelos de regressão.

Os modelos criados utilizando árvores de decisão M5P utilizaram os parâmetros:

- número mínimo de instâncias nos nós folha (*minNumInstances*) = 4;
- construir árvore de regressão (*buildRegressionTree*) = false;
- não utilizar suavização (*useUnsmoothed*) = false.

Para modelos gerados por redes neurais multi-camadas (MLP), foi feita uma normalização tanto dos atributos de cada instância como de classe dos dados de treinamento e utilizou-se:

- taxa de aprendizagem (*learningRate*) = 0,2 ;

- taxa de momento (*momentum*) = 0,2 ;
- número de camadas escondidas (*hiddenLayers*) = (número de features + números de classes) / 2;
- número de épocas (*trainingTime*) = 500.

Para o casos em que foi utilizado regressão aditiva (*Additive Regression*), os modelos foram criados utilizando:

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

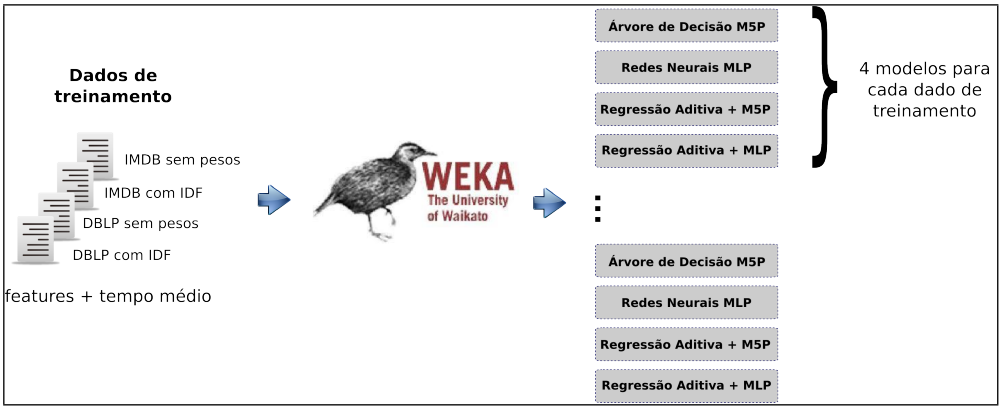


Figura 3.6: Gerando 4 modelos de regressão para cada dado de treinamento.

3.7 Métricas de Erro Utilizadas

Para calcular o erro dos resultados da predição de cada modelo analisado foi utilizada a métrica Erro Médio Relativo Absoluto (*Mean Relative Absolute Error*),

que é definida na equação 3.1, e a métrica Erro Médio Absoluto (*Mean Absolute Error*), que é definida na equação 3.2.

$$EMRA = \sum_{i=1}^{N} \frac{|p_i - e_i|}{|e_i - e_{medio}|} \tag{3.1}$$

$$EMA = \frac{1}{N} \sum_{i=1}^{N} |p_i - e_i| \tag{3.2}$$

Onde p_i e e_i são o tempo previsto e o tempo esperado para a instância i, respectivamente, e e_{medio} é o tempo esperado médio contido nos dados de treinamento (WITTEN; FRANK, 2005).

Em todas as abordagens, a validação dos resultados foi feita utilizando a técnica de validação cruzada, dividindo os dados de treinamento em 10 *folds* (*10-fold cross-validation*).

4 RESULTADOS E DISCUSSÃO

Nesta seção são abordados os resultados obtidos para os modelos utilizados. Inicialmente, são apresentados os resultados utilizando 600 *datasets* semi-sintéticos criados a partir do DBLP utilizando duas técnicas de aprendizagem de máquina. Em seguida, são apresentados os resultados para outras 600 bases de dados semi-sintéticas geradas a partir do IMDB utilizando as mesmas técnicas de aprendizagem de máquina da abordagem anterior. Por último, são apresentados os resultados obtidos pela combinação dos dados de treinamento gerados para os *datasets* do DBLP e do IMDB.

4.1 Tempo para Criação dos Modelos

A Tabela 4.1 traz, de forma resumida, o tempo mínimo e máximo demandados para criação dos modelos de regressão utilizados neste experimento. Como pode-se notar, modelos que utilizam árvore de decisão M5P foram os mais rápidos para serem criados, levando entre 0,04 e 1,18 segundos. Para os modelos que utilizaram redes neurais, percebe-se que o tempo necessário para criação dos modelos foi maior em relação ao tempo para criar as árvores de decisão, sendo necessário de 1,6 até 5,08 segundos para construção da rede.

Quando se utiliza a técnica de regressão aditiva, novos modelos são criados sequencialmente, o que impacta no tempo de criação de modelos utilizando tal abordagem. Novamente nota-se que, quando utilizou-se árvores de decisão em conjunto com a regressão aditiva, o tempo necessário para a construção dos modelos foi menor (de 0,46 a 3,22 segundos) em relação a abordagem utilizando redes neurais (de 15,86 a 31 segundos).

Tabela 4.1: Tempo gasto para construção dos modelos.

Modelo	Tempo mínimo	Tempo máximo
Árvores de Decisão M5P	0,04 segundos	1,18 segundos
Redes Neurais MLP	1,6 segundos	5,08 segundos
Regressão Aditiva + M5P	0,46 segundos	3,22 segundos
Regressão Aditiva + MLP	15,86 segundos	31 segundos

4.2 Resultados para *Datasets* Gerados a partir do DBLP

Nesta seção são detalhados os resultados dos modelos gerados para predição de tempo de execução do algoritmo MPJOIN sobre os *datasets* gerados a partir do DBLP, sendo que tanto para treinamento quanto para testes foram utilizados dados obtidos da execução do algoritmo de junção apenas sobre estes *datasets*.

4.2.1 *Datasets* DBLP sem Esquema de Peso

A Tabela 4.2 e as Figuras 4.1 e 4.2 trazem os resultados das predições de tempo de execução utilizando apenas dados de treinamento das execuções do algoritmo MPJOIN sobre os *datasets* gerados a partir do DBLP que não possuem ponderação de *tokens*.

O uso de regressão aditiva conseguiu melhorar a performance dos dois modelos utilizados, e, mesmo sem o uso de tal abordagem, os resultados já se enquadravam dentro de um intervalo de erro aceitável para o problema (erros inferiores a 6%).

Modelos utilizando árvores M5P obtiveram melhores resultados, sendo que, sem a regressão aditiva, o erro médio relativo absoluto destes foi de 5,04% contra 5,22% dos modelos utilizando redes neurais. Para os casos em que a técnica de regressão aditiva foi utilizada, os erros foram de 3,08% para os modelos basea-

dos em árvores M5P, e de 3,45% , onde novamente o desempenho das árvores de decisão foram superiores.

Outra observação importante, que pode ser notada a partir dos gráficos das Figuras 4.1 e 4.2, é que poucas previsões foram mal estimadas, estando a maior parte destas muito próximos da reta ideal de predições, o que indica que o modelo conseguiu bom desempenho em casos nos quais as consultas são muito rápidas e em casos que demoram muito tempo.

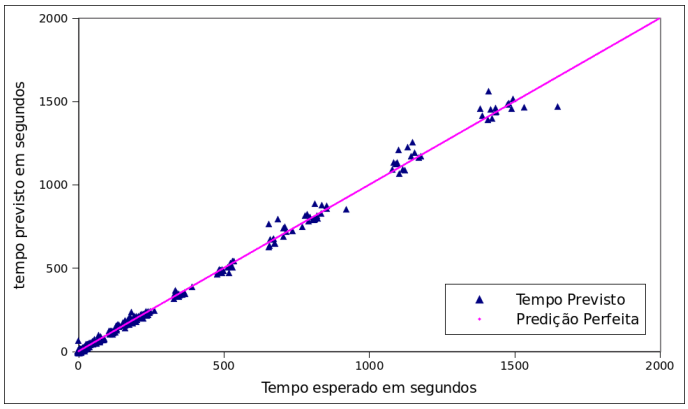
Tabela 4.2: Erros obtidos por cada modelo sobre os datasets gerados a partir do DBLP

Modelo	*Datasets*	Esquema de Peso	EMRA	EMA
M5P	DBLP	nenhum	5,04%	5,27 segundos
MLP	DBLP	nenhum	5,22%	5,46 segundos
RA+M5P	DBLP	nenhum	3,08%	3,22 segundos
RA+MLP	DBLP	nenhum	3,45%	3,60 segundos

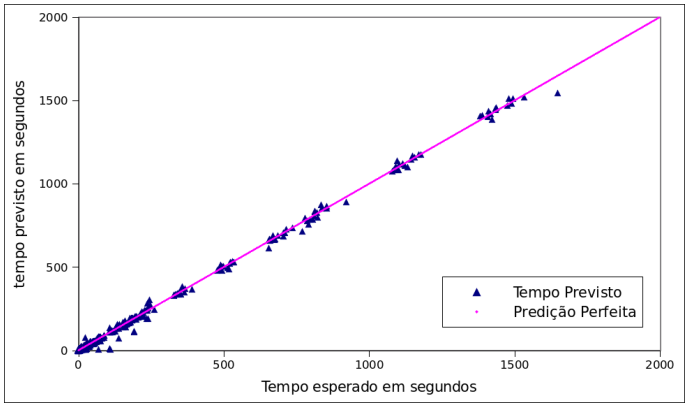
4.2.2 *Datasets* DBLP com Esquema de Peso IDF

A Tabela 4.3 e as Figuras 4.3 e 4.4 trazem os resultados das predições de tempo de execução utilizando apenas dados de treinamento das execuções do algoritmo MPJOIN sobre os *datasets* gerados a partir do DBLP utilizando ponderação de *tokens* IDF.

Novamente, o uso de regressão aditiva conseguiu melhorar a performance dos dois modelos utilizados, sendo que, nestes dados, o ganho foi superior ao de dados sem esquema de ponderação de *tokens*, principalmente para o modelo utilizando árvores M5P (redução de 7,35% para 2,22%). Em geral, os erros de previsão foram, para todos os modelos gerados por dados do DBLP, menores para *datasets* ponderados do que na abordagem sem utilização de pesos.



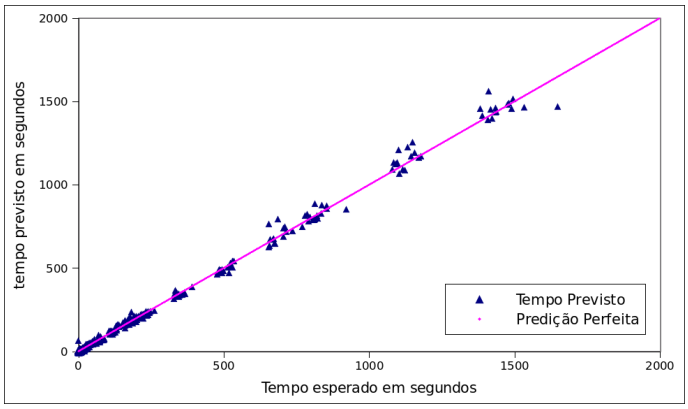
(a) Árvores de Decisão M5P.



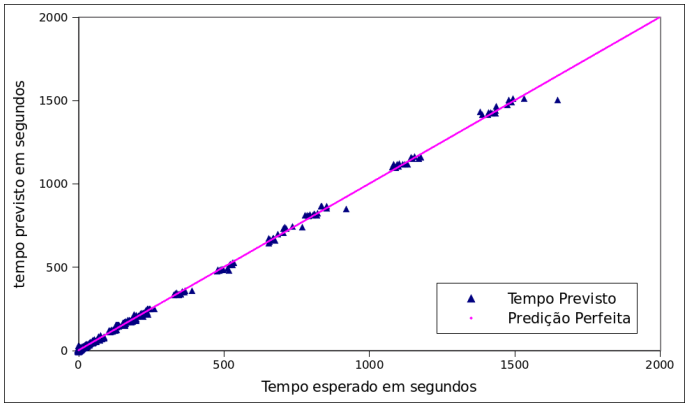
(b) Regressão Aditiva + M5P.

Figura 4.1: Árvores M5P: Tempo Esperado x Tempo Previsto (DBLP sem pesos)

Nesta abordagem, algumas predições foram mal estimadas em todos modelos, ficando distantes da reta ideal mesmo em casos que se utilizou regressão aditiva. Uma conjectura sobre estes erros foi que os dados da execução do algoritmo sobre as bases de dados geradas a partir do DBLP utilizando esquemas de peso IDF possuíram intervalos com pouca representatividade, o que pode ter prejudicado a performance dos modelos para instâncias que pertenciam a estes intervalos. Contudo, na grande maioria das predições ficaram muito próximas do valor ideal, mesmo nos casos em que o MPJOIN possui um tempo maior de execução, compensando erros isolados.



(a) Redes Neurais MLP.

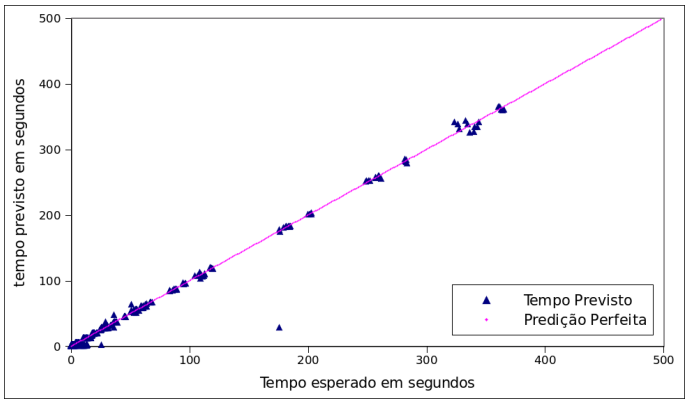


(b) Regressão Aditiva + MLP.

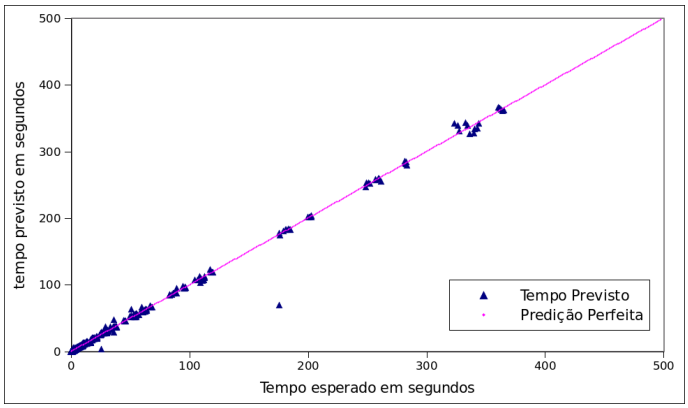
Figura 4.2: Redes Neurais MLP: Tempo Esperado x Tempo Previsto (DBLP sem pesos)

Tabela 4.3: Erros obtidos por cada modelo sobre os dados gerados a partir do DBLP com ponderação de tokens IDF.

Modelo	*Datasets*	Esquema de Peso	EMRA	EMA
M5P	DBLP	IDF	7,35%	1,67 segundos
MLP	DBLP	IDF	6,63%	1,50 segundos
RA+M5P	DBLP	IDF	2,22%	0,51 segundos
RA+MLP	DBLP	IDF	4,44%	1,01 segundos



(a) Árvores de Decisão M5P.

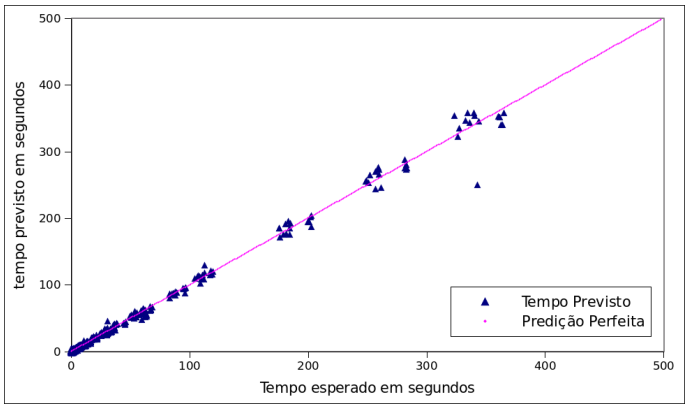


(b) Regressão Aditiva + M5P.

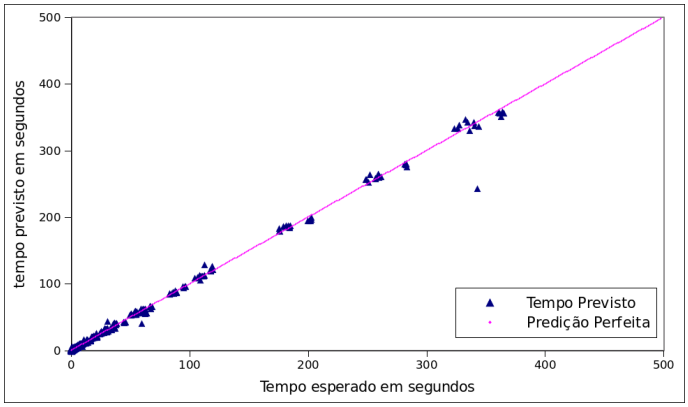
Figura 4.3: Árvores M5P: Tempo Esperado x Tempo Previsto (DBLP + IDF)

4.3 Resultados para *Datasets* Gerados a partir do IMDB

Nesta seção são detalhados os resultados dos modelos gerados para predição de tempo de execução do algoritmo MPJOIN sobre os *datasets* gerados a partir do IMDB, sendo que tanto para treinamento quanto para testes foram utilizados dados obtidos da execução do algoritmo de junção apenas sobre estes *datasets*.



(a) Redes Neurais MLP.



(b) Regressão Aditiva + MLP.

Figura 4.4: Redes Neurais MLP: Tempo Esperado x Tempo Previsto (DBLP + IDF)

4.3.1 *Datasets* IMDB sem Esquema de Peso

A Tabela 4.4 e as Figuras 4.5 e 4.6 trazem os resultados das predições de tempo de execução utilizando apenas dados de treinamento das execuções do algoritmo MPJOIN sobre os *datasets* gerados a partir do IMDB que não possuem ponderação de *tokens*.

Para esta abordagem, uso de regressão aditiva também melhorou a performance dos dois modelos utilizados, principalmente para o modelo utilizando redes neurais MLP (redução do erro de 12,93% para 6,43%). Contudo, as predições so-

bre os *datasets* sem podenração de *tokens* criados com base no IMDB obtiveram erros maiores em comparação com os resultados das execuções sobre os dados gerados a partir do DBLP sem nenhum esquema de peso.

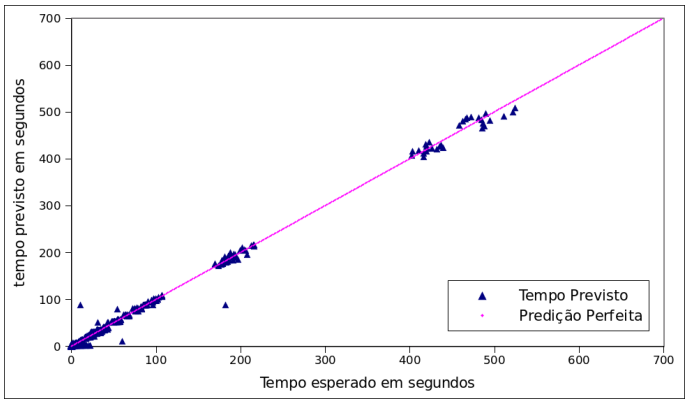
Utilizando redes neurais MLP, praticamente todos as estimações ficaram próximas da reta ideal, sem nenhuma predição muito distante de tal reta. Já para os modelos utilizando M5P, alguns pontos acabaram sendo mal estimados, ficando distantes da reta ideal. Uma possível interpretação para tal resultado é que, em etapas de treinamento, o algoritmo de indução da árvore de decisão faz uma poda de ruídos nos dados de treinamento, o que, combinado com as características dos *datasets* do IMDB, que possuem intervalos com baixa representatividade nos dados de treinamento, pode ter ignorado casos muito específicos da execução do algoritmo. Entretanto, tais casos não prejudicaram significativamente os resultados dos modelos que utilizaram M5P que, nas demais predições, obtiveram resultados melhores do que nos casos que se utilizou redes neurais.

Tabela 4.4: Erros obtidos por cada modelo sobre os datasets gerados a partir do IMDB

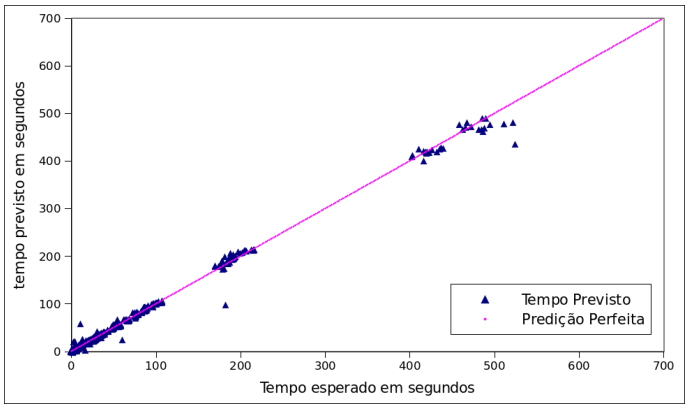
Modelo	*Datasets*	Esquema de Peso	EMRA	EMA
M5P	IMDB	nenhum	5,45%	1,91 segundos
MLP	IMDB	nenhum	12,93%	4,55 segundos
RA+M5P	IMDB	nenhum	4,3%	1,51 segundos
RA+MLP	IMDB	nenhum	6,43%	2,26 segundos

4.3.2 *Datasets* IMDB com Esquema de peso IDF

A Tabela 4.5 e as Figuras 4.7 e 4.8 trazem os resultados das predições de tempo de execução utilizando apenas dados de treinamento das execuções do algoritmo MPJOIN sobre os *datasets* gerados a partir do IMDB utilizando ponderação de *tokens* IDF.



(a) Árvores de Decisão M5P.

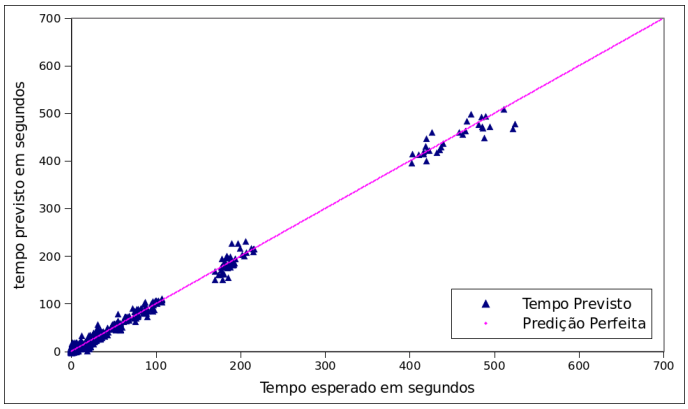


(b) Regressão Aditiva + M5P.

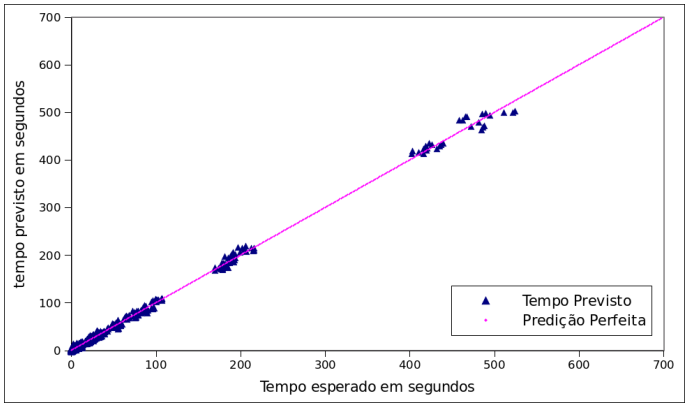
Figura 4.5: Árvores M5P: Tempo Esperado x Tempo Previsto (IMDB sem pesos)

Como podemos notar, o uso de regressão aditiva conseguiu melhorar a performance dos dois modelos utilizados, sendo que, nestes dados, o ganho foi menor ao de dados sem esquema de ponderação de *tokens*. Quando compara-se a performance dos modelos gerados a partir destas bases com as geradas a partir do DBLP que utilizam esquemas de pesos *IDF*, nota-se que, novamente, os erros foram maiores para os dados do IMDB em relação aos do DBLP.

Quando utilizou-se árvores de decisão M5P, novamente algumas predições foram mal estimadas, ficando distantes da reta ideal. Acredita-se que este fato



(a) Redes Neurais MLP.



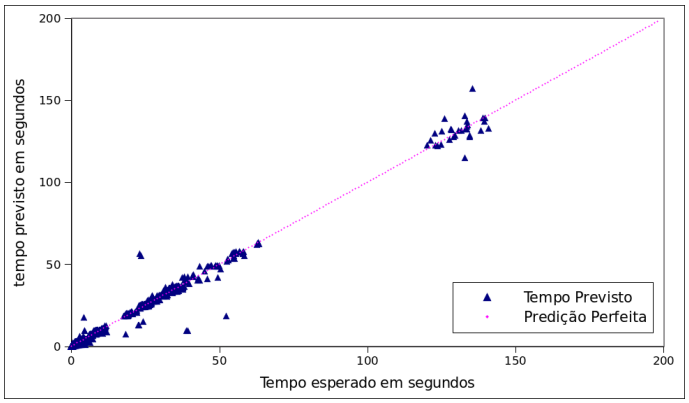
(b) Regressão Aditiva + MLP.

Figura 4.6: Redes Neurais MLP: Tempo Esperado x Tempo Previsto (IMDB sem pesos)

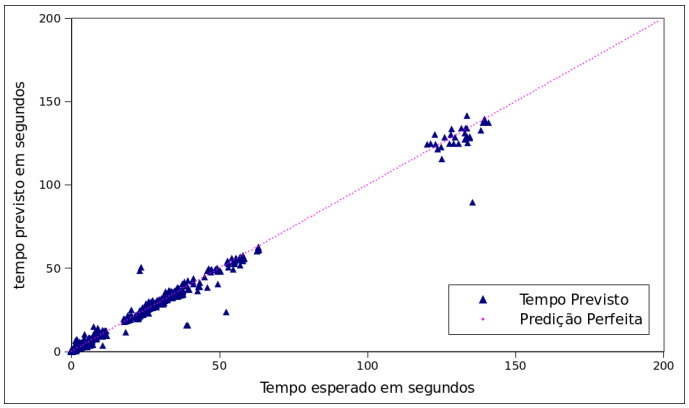
ocorreu devido ao mesmo motivo da abordagem anterior que, combinando características do algoritmo de indução da árvore com especifidades dos *datasets* gerados a partir do IMDB, alguns casos específicos dentro do intervalo de menor representatividade nos dados de treinamento podem ter sido ignorados. Com o uso de regressão aditiva, a distância destes casos específicos em relação a reta ideal foi reduzida. Desconsiderando tais casos específicos, os resultados dos modelos que utilizaram dados das execuções sobre bases do IMDB obtiveram bons resultados em geral, com erros inferiores a 8% quando utilizou-se regressão aditiva.

Tabela 4.5: Erros obtidos por cada modelo sobre os dados gerados a partir do IMDB com ponderação de tokens IDF.

Modelo	*Datasets*	Esquema de Peso	EMRA	EMA
M5P	IMDB	IDF	5,78%	0,85 segundos
MLP	IMDB	IDF	11,86%	1,75 segundos
RA+M5P	IMDB	IDF	4,98%	0,73 segundos
RA+MLP	IMDB	IDF	7,47%	1,11 segundos

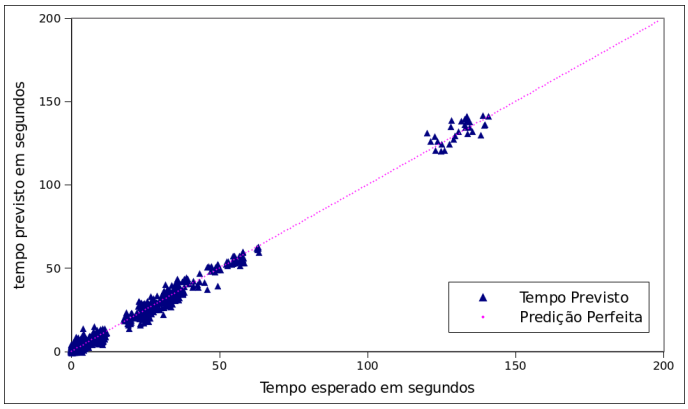


(a) Árvores de Decisão M5P.

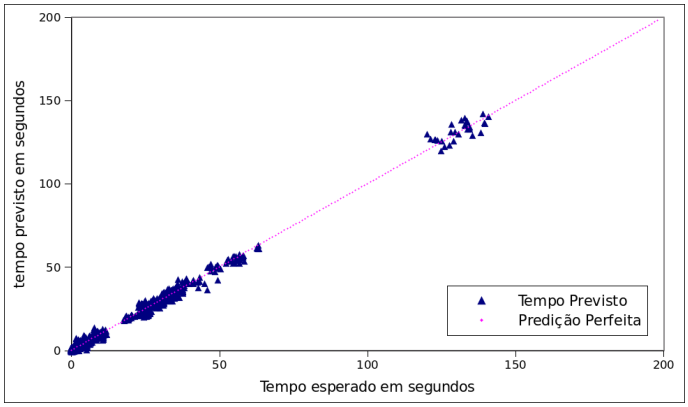


(b) Regressão Aditiva + M5P.

Figura 4.7: Árvores M5P: Tempo Esperado x Tempo Previsto (IMDB + IDF)



(a) Redes Neurais MLP.



(b) Regressão Aditiva + MLP.

Figura 4.8: Redes Neurais MLP: Tempo Esperado x Tempo Previsto (IMDB + IDF)

4.4 Resultados para *Datasets* Gerados a partir do IMDB e do DBLP

Nesta seção são detalhados os resultados obtidos para a predição de tempo de execução do MPJOIN utilizando todos os dados de treino disponíveis das execuções sobre as bases de dados do IMDB e do DBLP.

4.4.1 *Datasets* IMDB+DBLP sem Esquema de peso

A Tabela 4.6 e as Figuras 4.9 e 4.10 trazem os resultados das predições de tempo de execução utilizando apenas dados de treinamento das execuções do algoritmo MPJOIN sobre todos os *datasets* gerados a partir do IMDB e do DBLP que não possuem ponderação de *tokens*.

O uso de regressão aditiva conseguiu, novamente, melhorar a performance dos dois modelos utilizados, principalmente para o modelo utilizando redes neurais MLP (redução do erro de 10,09% para 3,78%). Para os modelos desta abordagem que utilizaram árvores de decisão, o desempenho foi superior em relação aos modelos baseados em árvores gerados com dados das bases do IMDB sem peso, e inferiores aos modelos baseados árvores gerados com dados do DBLP sem ponderação de tokens.

Quando utilizou-se redes neurais para a construção dos modelos, nota-se que os erros foram maiores em relação aos modelos baseados em redes neurais gerados com os dados do DBLP sem pesos, e melhores que os modelos baseados em redes neurais gerados sobre os dados do IMDB sem a regressão aditiva.

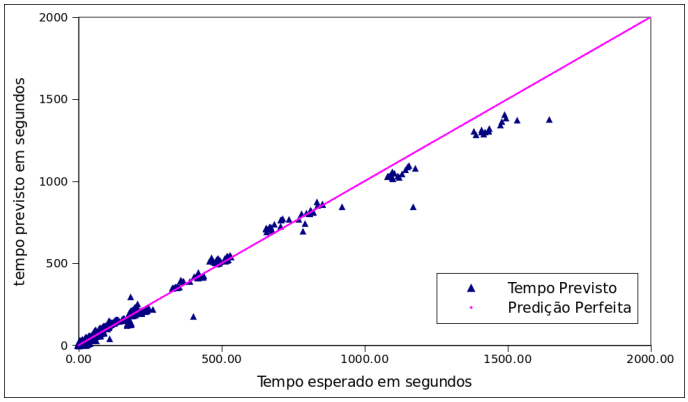
Utilizando redes neurais MLP, praticamente todos as estimações ficaram próximas da reta ideal, sem nenhuma predição muito distante de tal reta. Já para os modelos utilizando M5P, alguns pontos acabaram sendo mal estimados, ficando distantes da reta ideal. Entretanto, tais casos não prejudicaram significativamente os resultados dos modelos que utilizaram M5P, que, nas demais predições, obtiveram resultados melhores do que nos casos que se utilizou-se redes neurais.

4.4.2 *Datasets* IMDB+DBLP com Esquema de Peso IDF

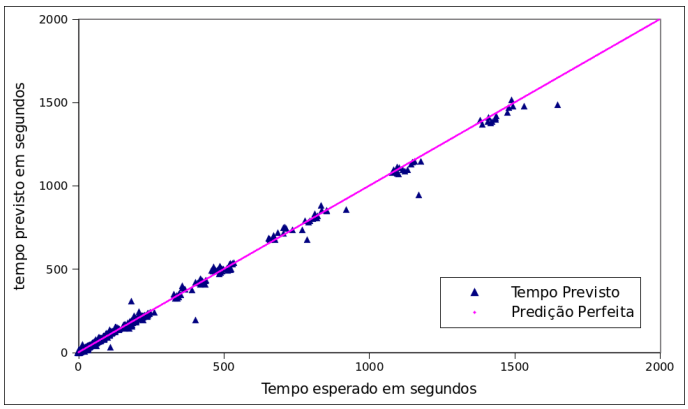
A Tabela 4.7 e as Figuras 4.11 e 4.12 trazem os resultados das predições de tempo de execução utilizando apenas dados de treinamento das execuções do algoritmo

Tabela 4.6: Erros obtidos por cada modelo sobre os datasets gerados a partir do IMDB+DBLP

Modelo	*Datasets*	Esquema de Peso	EMRA	EMA
M5P	IMDB+DBLP	nenhum	10,09%	7,06 segundos
MLP	IMDB+DBLP	nenhum	9,52%	6,66 segundos
RA+M5P	IMDB+DBLP	nenhum	3,78%	2,65 segundos
RA+MLP	IMDB+DBLP	nenhum	6,86%	4,8 segundos



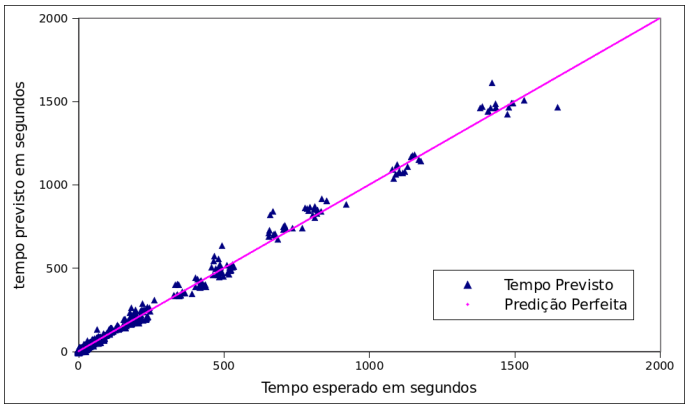
(a) Árvores de Decisão M5P.



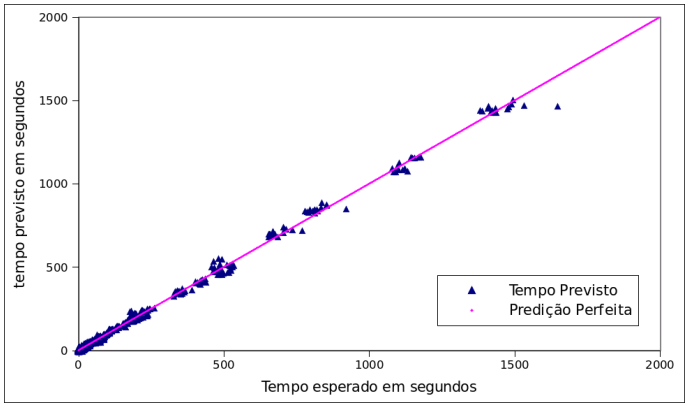
(b) Regressão Aditiva + M5P.

Figura 4.9: Árvores M5P: Tempo Esperado x Tempo Previsto (IMDB+DBLP sem pesos)

MPJOIN sobre os *datasets* gerados a partir do IMDB+DBLP utilizando ponderação de *tokens* IDF.



(a) Redes Neurais MLP.



(b) Regressão Aditiva + MLP.

Figura 4.10: Redes Neurais MLP: Tempo Esperado x Tempo Previsto (IMDB+DBLP sem pesos)

Como podemos notar, o uso de regressão aditiva conseguiu melhorar a performance dos dois modelos utilizados, principalmente para o modelo que utilizou árvore de decisão (redução do erro de 11,19% para 4,56%). Nesta abordagem, utilizando *tokens* ponderados, nota-se que os erros foram superiores quando não se utilizou nenhum esquema de pesos para *tokens*. Dentre os modelos gerados sobre dados ponderados, os modelos criados por esta abordagem obtiveram os resultados, sendo melhores apenas quando comparados com os modelos do IMDB que utilizaram regressão aditiva em conjunto com árvores de decisão M5P.

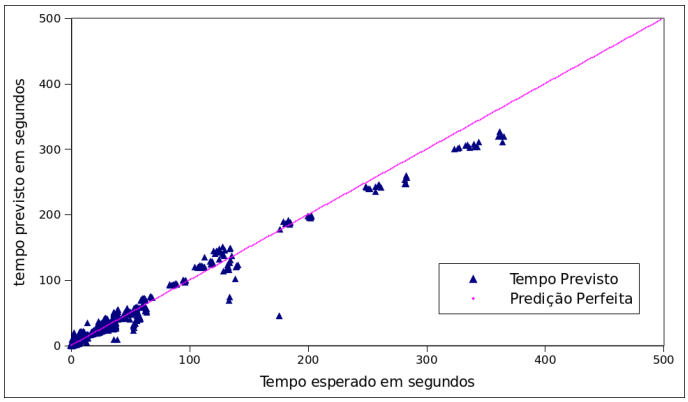
Mesmo com um erro médio absoluto maior, grande parte das predições sobre os dados da execução de duas fontes diferentes ficaram próximos da reta ideal, indicando que os modelos obtidos conseguiram capturar detalhes específicos de cada fonte de dados, fazendo predições mais genéricas que as abordagens anteriores.

Tabela 4.7: Erros obtidos por cada modelo sobre os dados gerados a partir do IMDB+DBLP com ponderação de tokens IDF.

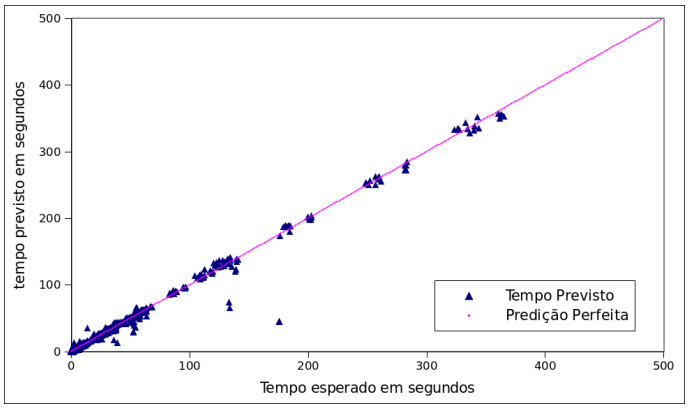
Modelo	*Datasets*	Esquema de Peso	EMRA	EMA
M5P	IMDB+DBLP	IDF	11,19%	2,07 segundos
MLP	IMDB+DBLP	IDF	14,47%	2,68 segundos
RA+M5P	IMDB+DBLP	IDF	4,56%	0,84 segundos
RA+MLP	IMDB+DBLP	IDF	7,92%	1,46 segundos

4.5 Sumário dos Resultados

A Figura 4.13 traz, de forma resumida, os erros obtidos por cada modelo sobre cada classe de conjuntos de dados geradas neste trabalho. Como pode-se notar, todos os modelos criados nestes experimento obtiveram resultados aceitáveis para a tarefa de prever o tempo de execução do algoritmo MPJOIN, sendo que 87,5% dos modelos criados obtiveram erros inferiores a 10%. Os piores resultados foram obtidos pelos modelos que utilizaram redes neurais que não utilizaram a técnica de regressão aditiva, sendo que o modelo que foi gerado sobre os dados de treino combinados do IMDB e do DBLP com esta configuração obteve o pior erro (14,47%). Os melhores resultados foram obtidos por modelos criados utilizando técnicas de regressão aditiva que, em todos os casos em que foi utilizada, reduziu os erros em relação a não utilização de tal abordagem. Assim sendo, quando a regressão aditiva foi adotada em conjunto com árvores de decisão M5P para os *datasets* do



(a) Árvores de Decisão M5P.

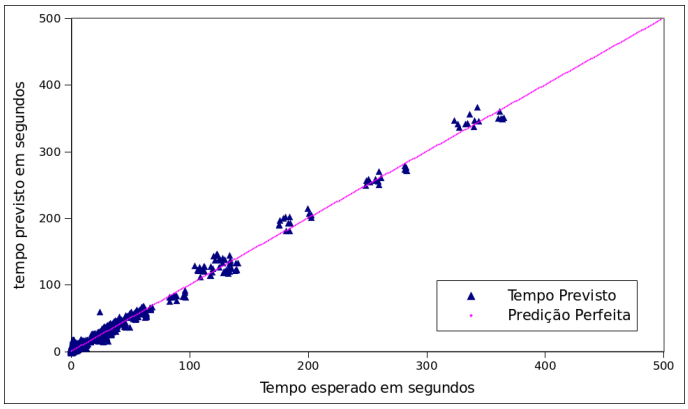


(b) Regressão Aditiva + M5P.

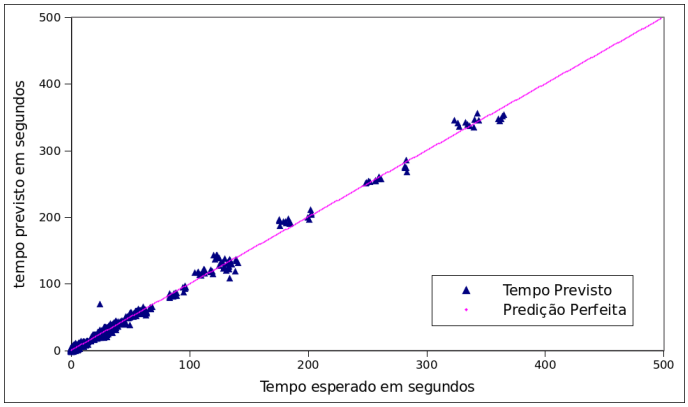
Figura 4.11: Árvores M5P: Tempo Esperado x Tempo Previsto (IMDB+DBLP + IDF)

DBLP utilizando *tokens* ponderados, obteve-se a melhor performance (2,22% de erro).

As execuções do MPJOIN sobre as bases geradas a partir do IMDB sem peso não obtiveram tempos de execução dentro do intervalo [250,400] segundos. Isto ocorreu devido a abordagem de concatenação dos atributos extraídos da base de dadps do IMDB, onde temos casos em que foram concatenados apenas o primeiro ator ou o título de filmes, gerando conjuntos de cardinalidade pequena, e casos em que todos os atores foram concatenados, gerando conjuntos com cardinalidade



(a) Redes Neurais MLP.

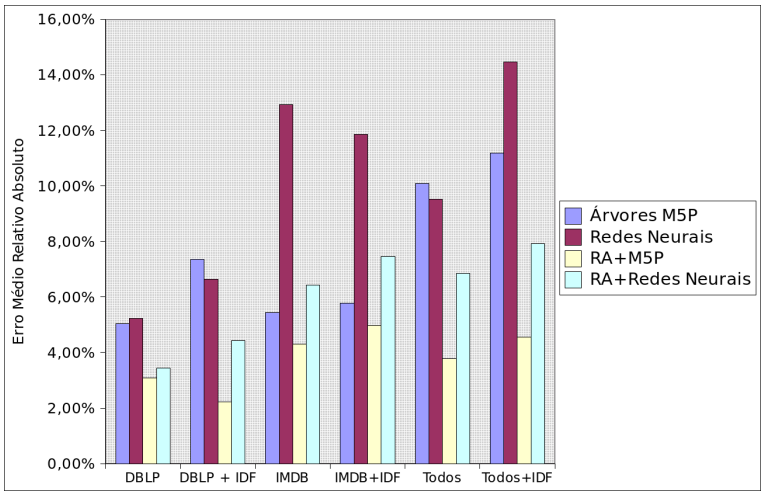


(b) Regressão Aditiva + MLP.

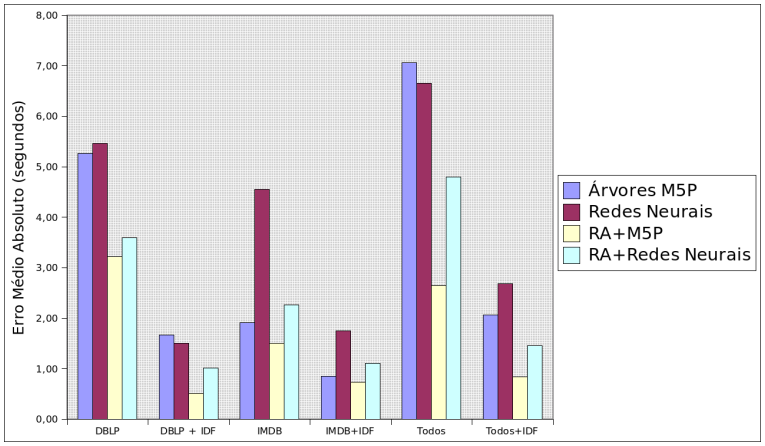
Figura 4.12: Redes Neurais MLP: Tempo Esperado x Tempo Previsto (IMDB+DBLP + IDF)

muito alta. Assim sendo, poucas combinações de concatenação geraram conjuntos com cardinalidade média, ou média-alta. Diante disso, como a cardinalidade dos conjuntos impacta diretamente no tempo de execução do MPJOIN (RIBEIRO; HÄRDER, 2011), tais intervalos vazios de execução ocorreram nos dados de treinamento.

Dentre as técnicas utilizadas, as árvores de decisão M5P foram as mais rápidas no aspecto de tempo para construção dos modelos, mesmo em casos que utilizaram a regressão aditiva. Modelos utilizando redes neurais MLP e regressão aditiva



(a) Resumo do Erro Médio Relativo Absoluto obtido por cada modelo.



(b) Resumo do Erro Médio Absoluto obtido para cada modelo.

Figura 4.13: Erros obtidos por cada modelo para cada classe de *datasets* gerados nos experimentos.

foram os mais que demandaram mais tempo para serem criados, chegando, no pior caso, a demorar 31 segundos para serem gerados.

Nos experimentos utilizando os dados de treinamento das execuções sobre dados de duas fonte diferentes combinados (DBLP e IMDB), é possível notar que os modelos criados foram capazes de generalizar de forma eficiente sobre dados de

diferentes fontes, os quais, devido a própria natureza dos dados, possuem diferentes distribuições de frequência, cardinalidade e tamanhos de conjuntos de tokens.

5 CONCLUSÕES E TRABALHOS FUTUROS

Com a crescente demanda por serviços de computação em nuvem, o planejamento de consultas a serem realizadas sobre as bases de dados assim como *hardware* dedicado a clientes torna-se fundamental. Diante disso, nesta pesquisa foi demostrada uma forma eficiente de prever o tempo de execução de uma operação crucial em serviços de integração de dados e limpeza de dados: as junções baseadas no conceito de similaridade. Modelos utilizando técnicas de regressão aditiva obtiveram resultados mais precisos, indicando que os objetivos iniciais de prever o tempo de execução do operador de junção por similaridade foi atingido.

Entretanto, novas questões surgem quando as cardinalidades dos *datasets*, ou seja, o número de tuplas presentes nos mesmos, sejam diferentes em etapas de treinamento e de teste, alterando significativamente o tempo de execução do algoritmo, mas mantendo as estatísticas similares. Diante disso, trabalhos futuros devem explorar modelos que consigam lidar com tais diferenças para fornecer predições eficientes mesmo em casos que a cardinalidade dos *datasets* se difere substancialmente em etapas de treinamento e de teste. Além disso, estão sendo estudados meios de otimizar a obtenção das estatísticas dos *datasets* utilizando técnicas de amostragem, característica que permitiria que ferramentas de controle de admissão de *hardware*, por exemplo, componham ferramentas que façam a predição do tempo demandado por operações de junção por similaridade sem a necessidade de uma pré-ordenação dos *datasets*.

REFERÊNCIAS BIBLIOGRÁFICAS

AKDERE, M.; ÇETINTEMEL, U.; RIONDATO, M.; UPFAL, E.; ZDONIK, S. B. Learning-based query performance modeling and prediction. In: KEMENTSIETSIDIS, A.; SALLES, M. A. V. (Ed.). *ICDE*. Washington, DC, USA: IEEE Computer Society, 2012. p. 390–401. ISBN 978-0-7685-4747-3.

BAYARDO, R. J.; MA, Y.; SRIKANT, R. Scaling up all pairs similarity search. In: WILLIAMSON, C. L.; ZURKO, M. E.; PATEL-SCHNEIDER, P. F.; SHENOY, P. J. (Ed.). *WWW*. Banff, Alberta, Canada: ACM, 2007. p. 131–140. ISBN 978-1-59593-654-7.

BISHOP, C. M. *Neural Networks for Pattern Recognition*. 1. ed. Oxford University Press, 1996. Paperback. ISBN 9780198538646. Disponível em: <http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/0198538642>.

BRAZIER, K. *KDnuggets:Pools: Data preparation*. October 2003. Disponível em: <http://www.kdnuggets.com/polls/2003/data_preparation.htm>.

GANAPATHI, A.; KUNO, H. A.; DAYAL, U.; WIENER, J. L.; FOX, A.; JORDAN, M. I.; PATTERSON, D. A. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In: IOANNIDIS, Y. E.; LEE, D. L.; NG, R. T. (Ed.). *ICDE*. [S.l.]: IEEE, 2009. p. 592–603. ISBN 978-0-7695-3545-6.

GRAVANO, L.; IPEIROTIS, P. G.; JAGADISH, H. V.; KOUDAS, N.; MUTHUKRISHNAN, S.; SRIVASTAVA, D. Approximate string joins in a database (almost) for free. In: APERS, P. M. G.; ATZENI, P.; CERI, S.; PARABOSCHI, S.; RAMAMOHANARAO, K.; SNODGRASS, R. T. (Ed.). *VLDB*. [S.l.]: Morgan Kaufmann, 2001. p. 491–500. ISBN 1-55860-804-4.

LI, J.; KÖNIG, A. C.; NARASAYYA, V. R.; CHAUDHURI, S. Robust estimation of resource consumption for sql queries using statistical techniques. *PVLDB*, v. 5, n. 11, p. 1555–1566, 2012.

QUINLAN, J. R. Learning with continuous classes. In: SINGAPORE. *Proceedings of the 5th Australian joint Conference on Artificial Intelligence*. [S.l.], 1992. v. 92, p. 343–348.

RIBEIRO, L. A.; HÄRDER, T. Generalizing prefix filtering to improve set similarity joins. *Information Systems*, v. 36, n. 1, p. 62–78, 2011. ISSN 0306-4379. Selected Papers from the 13th East-European Conference on Advances in Databases and Information Systems (ADBIS 2009). Disponível em: <http://www.sciencedirect.com/science/article/pii/S0306437910000657>.

ROBERTSON, S. E.; JONES, K. S. Relevance weighting of search terms. *Journal of the American Society for Information science*, Wiley Online Library, v. 27, n. 3, p. 129–146, 1976.

SARAWAGI, S.; KIRPAL, A. Efficient set joins on similarity predicates. In: *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 2004. (SIGMOD '04), p. 743–754. ISBN 1-58113-859-8. Disponível em: <http://doi.acm.org/10.1145/1007568.1007652>.

TAN, P.-N.; STEINBACH, M.; KUMAR, V. *Introduction to Data Mining, (First Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005. ISBN 0321321367.

TVERSKY, A. Features of similarity. *Psychological review*, American Psychological Association, v. 84, n. 4, p. 327, 1977.

UKKONEN, E. Approximate string matching with q-grams and maximal matches. *Theor. Comput. Sci.*, v. 92, n. 1, p. 191–211, 1992.

WITTEN, I. H.; FRANK, E. *Data Mining: Practical Machine Learning Tools and Techniques*. Second. Morgan Kaufmann, 2005. Paperback. (Morgan Kaufmann Series in Data Management Sys). ISBN 0120884070. Disponível em: <http://www.amazon.fr/exec/obidos/ASIN/0120884070/citeulike04-21>.

WU, W.; CHI, Y.; ZHU, S.; TATEMURA, J.; HACIGÜMÜS, H.; NAUGHTON, J. F. Predicting query execution time: Are optimizer cost models really unusable? In: *ICDE*. Brisbane, Australia: [s.n.], 2013.

XIAO, C.; WANG, W.; LIN, X.; YU, J. X.; WANG, G. Efficient similarity joins for near-duplicate detection. *ACM Trans. Database Syst.*, ACM, New York, NY, USA, v. 36, n. 3, p. 15:1–15:41, ago. 2011. ISSN 0362-5915. Disponível em: <http://doi.acm.org/10.1145/2000824.2000825>.