UNIVERSIDADE FEDERAL DO ACRE

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

ANÁLISE DE VARIAÇÕES DO MÉTODO DE AVALIAÇÃO JANELA DESLIZANTE EM MODELOS PREDITIVOS: UM ESTUDO DE CASO NO CONTEXTO DE *PULL REQUESTS*

RIO BRANCO

2018

BRUNO MAIA DA COSTA

ANÁLISE DE VARIAÇÕES DO MÉTODO DE AVALIAÇÃO JANELA DESLIZANTE EM MODELOS PREDITIVOS: UM ESTUDO DE CASO NO CONTEXTO DE *PULL REQUESTS*

Projeto de TCC apresentado como exigência parcial para obtenção do grau de Bacharel em Sistemas de Informação da Universidade Federal do Acre.

Prof. Orientador: Manoel Limeira de Lima Júnior Almeida.

RIO BRANCO

2018

TERMO DE APROVAÇÃO

BRUNO MAIA DA COSTA

ANÁLISE DE VARIAÇÕES DO MÉTODO DE AVALIAÇÃO JANELA DESLIZANTE EM MODELOS PREDITIVOS: UM ESTUDO DE CASO NO CONTEXTO DE *PULL REQUESTS*

Esta monografia foi apresentada como trabalho de conclusão de Curso de Bacharelado em Sistemas de Informação da Universidade Federal do Acre, sendo aprovado pela banca constituída pelo professor orientador e membros abaixo mencionados.

Compuseram a banca:

_____

Prof. Manoel Limeira de Lima Júnior Almeida, Dr.
Curso de Bacharelado em Sistemas de Informação

_____

Daricélio Moreira Soares, Dr.
Curso de Bacharelado em Sistemas de Informação

_____

Raoni Simões Ferreira, Dr.
Curso de Bacharelado em Sistemas de Informação

Rio Branco, 20 de agosto de 2018.

Dedico esta pesquisa a minha mãe e minha namorada, os pilares que me mantiveram em pé, seguindo em frente.

AGRADECIMENTOS

Agradeço em primeiro lugar a minha mãe, Maria Irene Ferreira Maia por sempre ter me apoiado e me dado forças nessa jornada. Em segundo lugar à minha namorada Karoliny da Silva Matos, que esteve ao meu lado durante toda a batalha contra o monstro temível chamado TCC, me dando a motivação necessária para não desistir.

Ao professor Manoel Limeira de Lima Júnior Almeida, dedico os meus mais profundos agradecimentos, sem a sua orientação e o tempo disponibilizado para me auxiliar nesta pesquisa, nunca teria chegado tão longe.

Aos meus colegas de graduação, em especial, José Marcos Lopes Damasceno, Alberto Ottaviano e Álvaro Lopes Rios, que me auxiliaram sempre que precisei.

Agradeço também ao grande mangaká Masashi Kishimoto, graças ao seu personagem chamado Naruto pude superar uma época muito difícil da minha vida (a morte de meu pai), além de aprender a sempre lutar até o fim e nunca desistir dos meus sonhos.

Por fim, agradeço a Universidade Federal do Acre por me propiciar a experiência de fazer parte do corpo discente do curso de Bacharelado em Sistemas de Informação.

"Nada pode ser obtido sem algum tipo de sacrifício. É sempre preciso oferecer algo em troca de valor equivalente"
(Edward Elric - Full Metal Alchemist)

RESUMO

A constante evolução das tecnologias da informação aumenta a cada dia o número de dados gerados e armazenados. Esses dados podem ser transformados em informações úteis e gerar conhecimento para uma organização ou equipe. O número elevado de dados torna inviável a análise manual, para contornar este problema, analistas utilizam a mineração de dados, uma das etapas do processo de KDD para explorar grandes bases de dados, permitindo assim interpretar os resultados e gerar conhecimento. Algumas bases de dados apresentam características temporais, onde a utilização dos métodos clássicos avaliação de algoritmos de classificação não refletem o que ocorre no mundo real. Nesse sentido, uma extensão da ferramenta WEKA *Training Test Sliding Validation* visa tornar a avaliação dos algoritmos de classificação mais coerente em bases de dados temporais, mantendo a ordem cronológica dos seus dados. Entretanto, a extensão original conta apenas com um método de manter a ordem dos dados durante a avaliação, por porcentagem de instâncias. Esta pesquisa desenvolveu e experimentou dois novos métodos de avaliação em bases temporais, por quantidade absoluta de instâncias e por intervalo de datas. Nos experimentos, foram utilizadas bases de dados sobre *pull requests*, com o intuito de identificar os parâmetros mais adequados para a avaliação dos algoritmos de classificação.

Palavras-chave: Avaliação de algoritmos de classificação, conjuntos, mineração de dados, bases de dados temporais

ABSTRACT

The constant evolution of information technology increases the number of data generated and stored each day. This data can be transformed into useful information and generate knowledge for an organization or team. The high number of data makes manual analysis unviable, to circumvent this problem, analysts use data mining, one of the steps in the KDD process to explore large databases, thus allowing interpretation of the results and generate knowledge. Some databases present temporal characteristics, where the use of the classical methods evaluation of data mining classification algorithms do not reflect what occurs in the real world. In this sense, an extension of the WEKA Training Test Sliding Validation tool aims to make the evaluation of classification algorithms more coherent in temporal databases, keeping the chronological order of their data. However, the original extension has only one method of maintaining the order of the data during the evaluation, by percentage of instances. This research developed and experimented with two new methods of evaluation on a time basis, by absolute number of instances and by interval of data. In the experiments, we used data bases on pull requests, in order to identify the most adequate parameters for the evaluation of classification algorithms.

Key-words: Evaluation of classification algorithms, sets, data mining, temporal databases

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A constante utilização das tecnologias da informação gera diariamente imensos volumes de dados. Esses dados fornecem para as organizações, por exemplo, a possibilidade de entender melhor o seu funcionamento e auxiliar no processo de tomada de decisão, revelando informações úteis, ou seja, conhecimento para a organização. Essa grande quantidade de dados é armazenada em diversas bases, gerando um grande volume de dados e tornando difícil a sua análise de forma manual, sendo necessário a utilização de técnicas e ferramentas automatizadas.

Para que seja possível contornar o problema da análise manual, analistas de dados utilizam o processo de descoberta de conhecimento em bases de dados (KDD – *Knowledge-Discovery in Databases*), um processo utilizado para extrair informações de bases de dados e encontrar informações úteis através da sua análise. Segundo Fayyad; Piatetsky-shapito e Smythy (1996), o processo traz diversas vantagens para uma organização, tais como: melhoria na velocidade de obtenção das informações, facilidade de comunicação entre os gestores, maximização dos resultados e ganhos da organização, bem como, fornecimento de suporte ao processo de tomada de decisão.

Segundo Hand, Mannila e Smyth (2001), existem diversas tarefas que podem ser realizadas com a mineração de dados, tais como: tarefas preditivas, tarefas descritivas, representação por conteúdo e analise exploratória dos dados. Dentre essas categorias, aquela que será utilizada ao longo deste trabalho é a tarefa preditiva de classificação, que tem como

objetivo prever informações, onde o valor de um atributo é sugerido por um algoritmo com base nos valores de outros atributos previamente conhecidos, com isso seria possível, por exemplo, definir promoções, recomendar produtos e diversas outras situações, dependendo dos objetivos a serem alcançados pelo usuário responsável pela mineração e dos resultados obtidos.

Em geral, as tarefas preditivas costumam dividir a base de dados com o objetivo de avaliar o poder preditivo dos algoritmos. Assim, uma parte da base é utilizada para o treinamento do algoritmo e outra para testes, de acordo com Han, Kamber e Pei (2011), alguns dos métodos mais tradicionais de avaliação são o método *holdout* e *k-fold cross.* No método *holdout*, o conjunto total de dados é divido aleatoriamente em dois subconjuntos exclusivos, um para treinamento e o outro para testes. Já o método *k-fold cross,* divide o conjunto total de dados em vários subconjuntos exclusivos e com o mesmo tamanho, em seguida, cada subconjunto é selecionado individualmente para o teste e os subconjuntos restantes são utilizados para o treino.

Algumas bases de dados apresentam características temporais[1], tal característica torna incompatível a utilização dos métodos clássicos de avaliação dos algoritmos preditivos. No método *holdout,* a formação do conjunto de treinamento pode conter instâncias com datas superiores às instâncias que fazem parte do conjunto de testes, em uma aplicação real isso não seria possível. O mesmo acontece no método *k-fold cross*, que também tentará predizer um atributo, utilizando valores futuros.

Para realizar um processo de avaliação mais coerente, Lima Júnior (2017) propôs um método capaz de lidar com bases de dados temporais, mantendo a ordem cronológica dos dados, método este que será explicado em detalhes no capítulo 3 desta pesquisa. Em resumo o método chamado de janela deslizante, utiliza duas janelas (uma para treino e uma para teste) com tamanhos em porcentagem, que são definidos pelo usuário no momento da utilização do método. O tamanho das janelas define o número de instâncias que irão compor os conjuntos de treino e teste, com isso a base de dados é verificada mantendo sua ordem cronológica. Lima (2017), implementou uma interface para o método em uma extensão para a ferramenta WEKA denominada *Training Test Sliding Validation*. No entanto, para realizar avaliações mais específicas, o método de avaliação pode ser incrementado para ser utilizado de forma customizada em bases de dados temporais.

1Dados que possuem o registro de data e hora da sua ocorrência, por exemplo, em uma base de dados sobre o clima ao longo de 1 ano, pode existir um atributo que armazena a data e hora do registro sobre a temperatura.

1.1 PROBLEMA DA PESQUISA

A janela deslizante é um método sensível ao tamanho das janelas e que depende diretamente da qualidade dos dados usados para o treinamento, dependendo de tais dados, ocorrem variações nas medidas de avaliação o que pode melhorar ou piorar os resultados obtidos pelo método de avaliação.

Os usuários do método utilizarão diversas bases de dados e a mineração em cada uma destas bases provavelmente será realizada com diversos objetivos, por conta disso, a divisão original em porcentagens utilizada nas janelas, limita as variações de distribuição dos dados de treinamento, já que o método sempre dividirá uma porcentagem dos dados, independente do número de dados que compõe essa porcentagem, ou de sua distribuição ao longo do tempo dentro da base de dados.

Essa limitação pode não atender as necessidades de todos os usuários da ferramenta, sendo necessário fornecer variações ao método para flexibilizar a sua utilização, tornando o método útil para um maior número de usuários, possibilitando por exemplo, que um geólogo marinho possa analisar dados sobre variações em correntes marinhas ao longo de 1 ano, gerando modelos preditivos com base em intervalos de semanas ou meses na base de dados, ou que uma equipe de desenvolvimento possa observar seu ritmo de integração de *pull requests* ao longo de uma certa faixa de tempo, analisando um número fixo de instâncias em cada modelo preditivo, definindo por exemplo, qual desenvolvedor da equipe é mais recomendado para fechar um *pull request*.

Neste contexto, questiona-se: Quais variações podem ser adicionadas no método de janela deslizante de forma a proporcionar uma avaliação coerente com bases que apresentem a característica temporal?

1.2 OBJETIVOS DA PESQUISA

Esta seção descreve o objetivo geral da pesquisa, bem como seus objetivos específicos.

1.2.1 OBJETIVO GERAL

Desenvolver duas variações para o método de avaliação de janela deslizante para algoritmos de classificação na extensão *Training Test Sliding Validation* e realizar experimentos preditivos em bases de dados temporais.

1.2.2 OBJETIVOS ESPECÍFICOS

Para que seja possível atingir o objetivo geral desta pesquisa, os seguintes objetivos específicos devem ser completados:

a. Especificar os requisitos para as variações de implementação do método de avaliação da janela deslizante;
b. Analisar o código fonte do método de avaliação;
c. Implementar variações a serem adicionadas no método de avaliação;
d. Realizar experimentos preditivos em bases de dados temporais e testar as novas variações;

1.3 JUSTIFICATIVA DA PESQUISA

Como já abordado no problema de pesquisa, os métodos de avaliação tradicionais se mostram incompatíveis com bases de dados temporais, pois, em algum momento dados do futuro podem fazer parte do conjunto de treino, o que não faz sentido, já que não poderia acontecer no mundo real. Nesse contexto, é necessário fornecer um método capaz de lidar com bases de dados temporais, garantindo que a ordem cronológica dos dados seja mantida durante toda a avaliação. Isso assegura que o conhecimento gerado seja confiável e mais fiel ao mundo real.

Esta pesquisa pretende incrementar variações para o método da janela deslizante,

pois através deste método é possível garantir que a característica temporal das bases de dados seja mantida, desta forma, o treinamento e teste realizado pelos algoritmos serão aplicados nos dados sempre seguindo sua ordem cronológica, e, como já demonstrado na tese de Lima Júnior (2017), acredita-se que esse é um dos métodos mais adequados a serem utilizados na avaliação de algoritmos de classificação em bases de dados temporais.

A implementação de variações permite expandir o método, trazendo novos parâmetros para sua utilização e com isso, um maior número de usuários poderão alcançar seus objetivos ao utilizá-lo, tornando o método ainda mais útil no tratamento de bases de dados temporais, visto que existirá um maior número de possibilidades de customização das configurações utilizadas para dividir a base, podendo assim gerar modelos preditivos mais precisos.

1.4 METODOLOGIA

De acordo com Wazlawick (2014), esta pesquisa pode ser classificada em três critérios.

a. Quanto a sua natureza: original, pois a pesquisa busca trazer um novo conhecimento, expandindo um método de mineração preditiva já existente chamado janela deslizante.
b. Quanto ao seu objetivo: Consiste de uma pesquisa empírica, pois ela busca desenvolver, testar e apresentar uma nova abordagem para um método de avaliação de algoritmos preditivos já existente, realizando assim uma expansão desse método.
c. Quanto ao procedimento técnico: a pesquisa se classifica como experimental, pois consiste em desenvolver, testar e apresentar uma expansão de um método de avaliação para algoritmos preditivos.

Quanto à forma como esta pesquisa foi conduzida, pode-se organizá-la em 5 etapas:

a. A primeira etapa consistiu em construir uma fundamentação teórica sobre os principais conceitos, métodos e ferramentas que servem de base para esta pesquisa, nesta fundamentação é abordado sobre a descoberta de conhecimento,

mineração de dados, principais técnicas preditivas, avaliação de algoritmos preditivos, métricas de avaliação de algoritmos preditivos, métodos de avaliação de algoritmos preditivos, ferramenta WEKA, gerência de configuração de software e *pull request*.

b. Na segunda etapa, foi desenvolvido o documento de requisitos das variações a serem implementadas.
c. A terceira etapa consistiu da implementação das variações do método, onde o código fonte original foi analisado e compreendido, para então se implementar as modificações na extensão, utilizando a linguagem de programação Java.
d. Na quarta etapa, foram realizados experimentos com algoritmos preditivos, usando a extensão com as variações implementadas, com o intuito de atingir o objetivo proposto no problema de pesquisa.
e. Por fim, na última etapa foi realizada a análise e interpretação dos resultados obtidos e do processo realizado.

1.5 ORGANIZAÇÃO DA PESQUISA

Além do Capítulo 1, onde foram apresentadas introdução, problema da pesquisa, justificativa, metodologia e organização, a pesquisa está organizado nos seguintes capítulos. No capítulo 2, é apresentada uma fundamentação teórica sobre os conceitos, métodos e ferramentas que servem de base para esta pesquisa, apresentando a descoberta de conhecimento, mineração de dados, principais técnicas preditivas, avaliação de algoritmos preditivos, métricas de avaliação de algoritmos preditivos, métodos de avaliação de algoritmos preditivos, ferramenta WEKA, gerência de configuração de software e *pull request*. No capítulo 3, é apresentada a extensão *Training Test Sliding Validation*, sua utilidade, o funcionamento do seu método e a implementação das variações. No capítulo 4, as bases de dados utilizadas são descritas e os experimentos realizados com o uso da extensão modificada são demonstrados. Por fim, o Capítulo 5 desta pesquisa trata da sua conclusão, onde são dadas as considerações finais e recomendações.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta o referencial teórico desta pesquisa, com cada seção se baseando na literatura existente. Estão descritos neste capítulo os conceitos referentes a descoberta de conhecimento em bases de dados na seção 2.1; definições sobre mineração de dados na seção 2.2; uma breve descrição sobre as principais técnicas preditivas na seção 2.3; avaliação de algoritmos preditivos na seção 2.4; métricas de avaliação de algoritmos preditivos na seção 2.4.1; métodos de avaliação de algoritmos preditivos na seção 2.4.2; ferramenta WEKA na seção 2.5; gerencia de configuração de software na seção 2.6; *pull request* na seção 2.7 e considerações sobre o capítulo na seção 2.8.

2.1 DESCOBERTA DE CONHECIMENTO

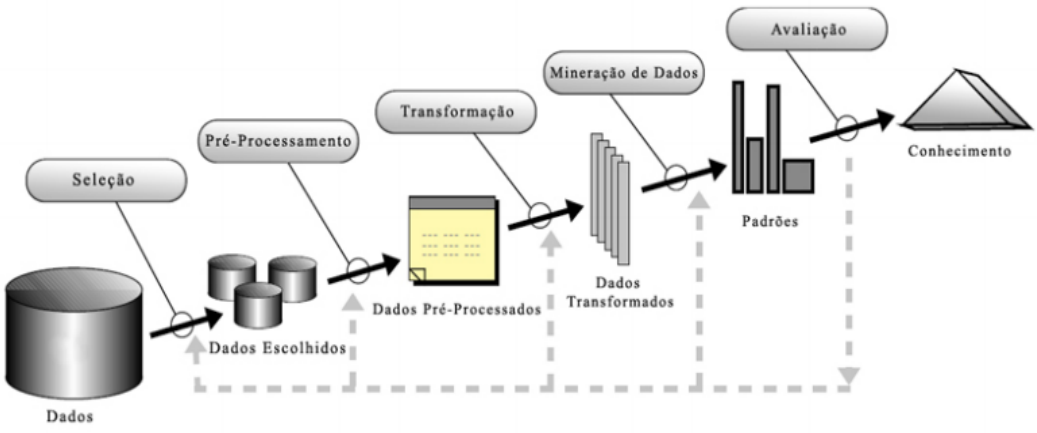
Segundo Fayyad, Piatetsky-shapito e Smythy (1996) a "descoberta de conhecimento em bancos de dados (KDD – *Knowledge Discovery in Databases*) é um processo, não trivial, de extração de informações implícitas, previamente desconhecidas e potencialmente úteis, a partir dos dados armazenados em um banco de dados". A característica de "não trivial" deixa claro que existe uma técnica de busca ou interferência, enquanto a característica de "previamente desconhecida", significa que a informação obtida deve ser nova e potenciamente útil, possuindo valor e trazendo benefícios para aqueles interessados nela.

Ainda de acordo com Fayyad, Piatetsky-shapito e Smythy (1996), o processo de KDD compreende uma sequência de 5 etapas, correlacionadas a fim de produzir o melhor resultado, as etapas são:

a. **Seleção**: Etapa inicial do processo, onde é realizada a escolha do conjunto de dados que será analisado.
b. **Pré-processamento**: Nesta etapa ocorre a limpeza dos dados, utilizando vários processos para reduzir dados irrelevantes ou ruins a fim de filtrar os dados com a melhor qualidade possível.
c. **Transformação**: Nesta etapa é realizada a manipulação dos dados de forma a enriquecer e transformá-los em dados adequados para a mineração.
d. **Mineração de dados**: Métodos são aplicados nesta etapa sobre os dados para se realizar a descoberta de padrões e/ou relacionamentos úteis.
e. **Interpretação/avaliação**: Etapa final onde o usuário responsável pela mineração analisa os resultados obtidos, interpretando se os resultados foram úteis ou não.

A Figura 1 mostra uma visão geral das etapas do processo de KDD, vale observar a ligação entre cada etapa do processo (demonstrada através da direção das setas), onde é possível notar, por exemplo, que a partir de qualquer etapa é possível retornar a uma das etapas anteriores.

Figura 1 - Etapas do processo de KDD.



Fonte: Camilo e Silva (2009).

Em termos gerais, não importa qual seja a área onde o processo de KDD é aplicado, contanto que exista a necessidade se analisar um grande volume de dados, essa área será enormemente beneficiada (SILVA, 2016). A etapa principal é conhecida como mineração de

dados, seus conceitos, funcionamento e técnicas serão explicadas na seção a seguir.

2.2 MINERAÇÃO DE DADOS

Segundo Fayyad, Piatetsky-shapito e Smythy (1996), a mineração de dados é uma parte do processo de descoberta de conhecimento em bancos de dados (KDD). A mineração de dados é composta por várias ferramentas e técnicas, que ao utilizar algoritmos são capazes de explorar conjuntos de dados dos mais diversos tamanhos, extraindo ou então auxiliando no ato de evidenciar padrões nestes dados, realizando assim, a descoberta de conhecimento. O conhecimento adquirido pode então ser apresentado de diversas formas, como: regras, árvores, grafos ou agrupamentos. Berry e Linoff (1997), também citam que "mineração de dados é a exploração e a análise, por meio automático ou semiautomático, de grandes quantidades de dados, a fim de descobrir padrões e regras significativas". Por mais que as definições da mineração de dados levem a crer que seja um processo automático, isto não é totalmente verdade. De acordo com Camilo e Silva (2009), a mineração de dados ainda é um processo que depende da intervenção humana, pois os dados gerados por ela precisam passar por uma análise, realizada por um humano com o intuito de descobrir se os objetivos iniciais que levaram a mineração de dados foram alcançados.

Existem diversos tipos de tarefas possíveis a se realizar com a mineração de dados, de acordo com Cios *et al*. (2007), essas tarefas são geralmente classificadas em duas categorias:

a. **Tarefas descritivas (não supervisionado)**: Esta categoria tem o objetivo de encontrar relacionamento entre os dados, sua principal característica possivelmente é o fato de que não é preciso realizar uma categorização nos dados, pois as tarefas descritivas não precisam de um atributo alvo, ao invés disso para realizar a mineração nesta categoria é comumente realizada medidas de similaridade entre os dados, as principais tarefas nesta categoria são as regras de associação e o agrupamento.
b. **Tarefas preditivas (supervisionado)**: Tem o objetivo de prever o valor de um

> atributo alvo, utilizando como base os valores de outros atributos, para conseguir isto as tarefas desta categoria possuem um conjunto de dados com uma variável alvo definida, desta forma durante a mineração se busca classificar os dados com base nesta variável alvo, os principais tipos de tarefas preditivas são: classificação e regressão.

O tipo de tarefa a ser explorada depende diretamente do objetivo a ser alcançado com o processo de mineração e também do domínio do negócio relacionado. Os experimentos realizados nesta pesquisa exploraram as tarefas do tipo preditiva. Neste cenário, duas técnicas podem ser utilizadas, a classificação quando a variável dependente é categórica e regressão quando a variável dependente possui valores numéricos (HAND; MANNILA; SMYTH, 2001). Tais tarefas serão detalhadas na seção a seguir.

2.3 PRINCIPAIS TÉCNICAS PREDITIVAS

Assim como existem vários tipos de tarefas de mineração de dados, existem também diversas técnicas para cada tarefa, ao longo desta seção, algumas das principais técnicas preditivas serão descritas.

A técnica conhecida como **regressão**, consiste em modelar o relacionamento de uma variável independente e uma variável dependente. Essa técnica é usada quando o dado é identificado por um valor numérico e não um valor categórico, a variável independente é um atributo do dado e a variável dependente é a resposta a qual se quer alcançar, aquilo que se deseja predizer. Com isso, se torna possível, por exemplo, analisar uma base de dados contendo dados referentes a compras de clientes ao longo de 2 semanas, indicando os gastos de cada um e suas preferências. Assim, após a mineração, o modelo gerado se tornaria capaz de predizer qual será o valor gasto por um novo cliente (CAMILO; SILVA, 2009).

A regressão pode assumir dois tipos, **regressão linear** e **regressão não-linear**, a mesma assume uma forma linear quando a relação entre a variável independente e dependente segue uma forma linear, nestes casos é possível traçar um gráfico onde o valor de **y** será uma

função linear de **x**. Já a forma não linear é assumida quando a relação entre a variável independente e dependente não segue uma forma linear. Outros tipos também são encontrados na literatura: *Logistic Regression*, *Poisson Regression* e *Log-Linear Models* (CAMILO; SILVA, 2009).

A **classificação** é uma técnica com o objetivo de prever os valores assumidos por variáveis categóricas, desta forma é possível agrupar os dados em categorias. O funcionamento da classificação se dá analisando os dados fornecidos ao algoritmo, sendo que cada dado já possui uma indicação de qual classe pertence, desta forma o algoritmo "aprende", com base nos dados existentes, como realizar a classificação de um novo dado. Se a técnica for utilizada em uma Empresa de Empréstimos, com o objetivo de avaliar seus clientes, ao usar a classificação seria possível analisar todos os clientes atuais da Empresa, observando aqueles que possuem uma renda fixa, analisando seus perfis e com isso classificando quais clientes devem ou não ter seus empréstimos aceitos, criando um modelo para classificar cada cliente dentro da base de dados (CAMILO; SILVA, 2009).

De acordo com Camilo e Silva (2009), dentro da literatura, diversas técnicas diferentes de classificação são citadas, alguns dos principais paradigmas serão brevemente descritos a seguir:

A **Classificação Bayesiana é** uma técnica estatística baseada no teorema de Thomas Bayes. O teorema demonstra como é possível descobrir a probabilidade de um evento ocorrer, tomando como base a probabilidade de um evento que já ocorreu, para isso o teorema define a seguinte Equação 1 a seguir (HAN; KAMBER; PEI, 2011).

$$P(A/B)=\frac{P(B/A)P(A)}{P(B)} \quad (1)$$

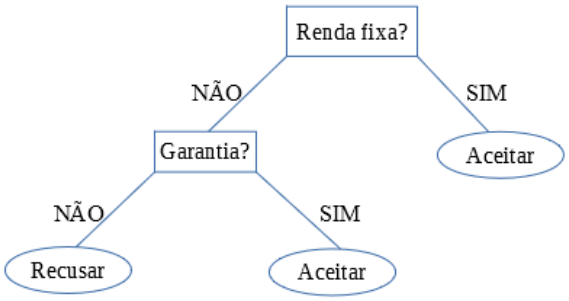
Os algoritmos baseados neste teorema carregam o nome *bayes* como *Naive bayes, Bayesian network*, entre outros. São algoritmos bastante simples, possuem um alto poder preditivo e segundo Zhang (2004), este alto poder e simplicidade os torna um dos algoritmos mais utilizados na classificação. O princípio fundamental do algoritmo é que não exista uma relação de dependência entre os atributos, entretanto, podem existir casos onde esta dependência ocorra, para atender estes casos, existe a variação da técnica chamada *Bayesian Network* (CAMILO; SILVA, 2009).

Outra técnica é conhecida como **árvores de decisão,** nesta técnica o resultado da mineração é organizado como um fluxograma em forma de árvore, onde cada nó indica o

teste de um atributo e cada folha indica uma classe, desta forma, para classificar um novo dado basta seguir o fluxo da árvore, percorrendo sua extensão a partir do nó raiz ate uma de suas folhas, com base nos atributos do dado a ser classificado (GAMA, 2004).

Um exemplo de árvore de decisão para a empresa de empréstimos citada no exemplo da classificação é mostrado na Figura 2. Cada nó da árvore possui um atributo que um cliente pode ou não possuir e as folhas representam os resultados que classificam se o empréstimo deve ser aceito ou não.

Figura 2 - Exemplo de uma árvore de decisão genérica para o exemplo da empresa de empréstimo.



Fonte: autoria própria.

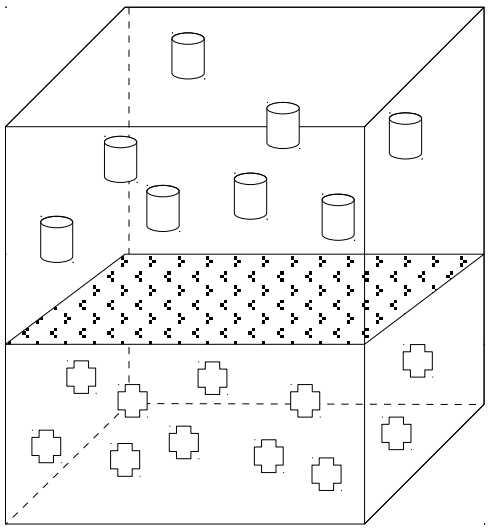
Um algoritmo da família das árvores de decisão, que é bastante utilizado e conhecido e também é um dos principais algoritmos utilizados nesta pesquisa é o *randomForest.* Por conta de sua estrutura, árvores de decisão também podem ser convertidas em regras de classificação do tipo "se-então", tornando mais fácil sua compreensão, por exemplo, na árvore da última figura, poderíamos tirar uma regra como "SE tem renda fixa ENTÃO aceitar" (GAMA, 2004). Quando o foco da mineração está em realizar a extração e análise de regras "se-então" citadas anteriormente, se origina uma outra técnica conhecido como **Classificação Baseada em Regras**, tal técnica normalmente é utilizada, quando árvores muito grandes são geradas, pois, interpretar os resultados apenas pela análise da árvore seria muito difícil.

Máquinas de vetores de suporte (SVM) é uma técnica que tenta classificar dados de entrada entre duas classes. Para isso, o SVM utiliza um modelo desenvolvido com as entradas de treinamento, as mapeando em um espaço multidimensional e utilizando regressão para encontrar um hiperplano[2], que melhor separe duas classes de entradas. Após o SVM ser treinado, ele se torna capaz de avaliar novos dados, relacionando-os ao hiperplano divisor e desta forma os classificando entre as duas classes (CUNHA, 2013).

2Hiperplano é uma superfície em um espaço de n dimensões que o separa em duas partes/metades.

Uma forma de tornar mais claro os SVM, é conceituá-los através de uma imagem como a Figura 3, onde os dados estão distribuídos em um espaço 3D, sendo divididos pelo hiperplano ao centro em duas classes qualquer, uma representada por cilindros e outra por cruzes.

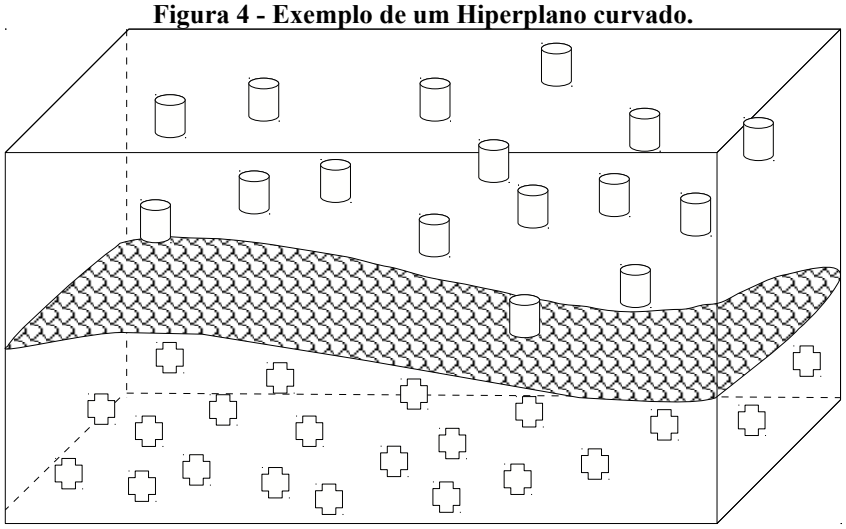
Figura 3 - Exemplo de um SVM no espaço.



Fonte: Adaptação Cunha, 2013.

Vale ressaltar que muito dificilmente uma base de dados poderia ser divida por um hiperplano como o da Figura 3, pois os dados muito dificilmente estarão organizados de uma forma tão Linear, sendo assim, na maioria dos casos o hiperplano assume uma forma mais curvada para comportar os dados, a Figura 4 apresenta um o exemplo de uma possível curvatura para o hiperplano (CUNHA, 2013).

Figura 4 - Exemplo de um Hiperplano curvado.



Fonte: Adaptação Cunha, 2013.

A técnica do **K-NN** foi proposta por Fukunaga e Narendra (1975), foi um dos primeiros algoritmos a tentar solucionar o problema do caixeiro viajante[3]. Segundo Pacheco (2017), é um dos mais simples classificadores a ser implementado, além de possuir uma fácil compreensão e obter bons resultados, o objetivo do K-NN é determinar a classe de um dado, baseado nos dados vizinhos vindos de um conjunto de treinamento.

Por exemplo, em uma situação onde um novo dado seja adicionado a base de dados, os dados vizinhos mais próximos serão utilizados para descobrir a qual classe o dado pertence, se definirmos que existem 7 dados vizinhos mais próximos, 4 deles possuem a classe “aceitar empréstimo” e 3 possuem a classe “recusar empréstimo”, desta forma, após o treinamento o algoritmo classificará o novo dado como “aceitar empréstimo”, por possuir mais vizinhos com essa classe.

2.4 AVALIAÇÃO DE ALGORITMOS PREDITIVOS

Ao utilizar algoritmos preditivos, o objetivo é gerar o melhor modelo preditivo possível, minimizando possíveis erros na previsão. No entanto, nem sempre os modelos preditivos acertam a previsão, por conta disso, medir a capacidade preditiva do modelo é importante, assim é possível definir a confiabilidade do modelo em aplicações reais (ANDREONI, 2014). As seções a seguir tratarão de dois aspectos importantes na medição da capacidade preditiva de um modelo: as métricas e os métodos de avaliação.

2.4.1 MÉTRICAS DE AVALIAÇÃO DE ALGORITMOS PREDITIVOS

De acordo com Hand, Mannila e Smyth (2001), o item fundamental utilizado para

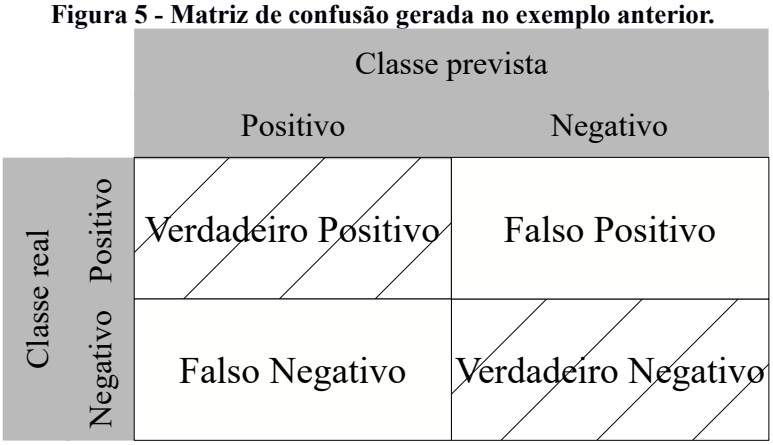
3Problema que tenta determinar a menor rota para percorrer uma série de cidades (visitando uma única vez cada uma delas), retornando à cidade de origem. Ele é um problema de otimização NP-difícil, inspirado na necessidade dos vendedores em realizar entregas em diversos locais (as cidades), percorrendo o menor caminho possível, reduzindo o tempo necessário para a viagem e os possíveis custos com transporte e combustível.

medir a capacidade preditiva de um modelo é a matriz de confusão. Clésio (2014) explica que quando algoritmos preditivos são usados, seu objetivo é prever a qual classe os dados pertencem. Por exemplo, digamos que em uma base de dados sobre algum teste qualquer, os dados possam ser classificados em duas classes, positivo ou negativo, neste caso, de acordo com Han, Kamber e Pei (2011) existem quatro valores que servem de base para as métricas usadas na classificação da capacidade preditiva do modelo gerado, são elas:

a. **Verdadeiros positivos:** dados classificados como positivos e que realmente possuem o rótulo de classe positivos;
b. **Verdadeiros negativos:** dados classificados como negativos e que realmente possuem a classe negativos;
c. **Falsos positivos:** Dados classificados de forma errada como positivos, já que possuem o rotulo de classe negativos;
d. **Falsos negativos:** Dados classificados de forma errada como negativos, já que possuem o rotulo de classe positivos;

Cada uma destas medidas pode então ser colocada em uma tabela, conhecida como tabela de contingência ou matriz de confusão (HAN; KAMBER; PEI, 2011, HAND; MANNILA; SMYTH, 2001). Tal matriz será demonstrada na Figura 5.

Figura 5 - Matriz de confusão gerada no exemplo anterior.



Fonte: adaptação de Andreoni (2014).

Caso o número de classes aumente, uma nova coluna e linha para cada classe é adicionada a matriz, mantendo sempre o mesmo formato, vale notar que as verificações verdadeiras sempre estarão na diagonal principal da matriz (HAND; MANNILA; SMYTH,

2001). Com a matriz completa é possível derivar outras métricas como a acurácia, taxa de erro, sensibilidade, especifidade, eficiência, precisão, *recall* e *f-score*, cada uma dessas métricas serão definidas a seguir.

A acurácia é a taxa de acerto (verdadeiros positivos e verdadeiros negativos), em relação a todos os dados presentes no modelo. Vale ressaltar que a acurácia é uma métrica extremamente suscetível ao desbalanceamento das classes no conjunto de dados, o que pode induzir a uma interpretação errada dos resultados (HAN; KAMBER; PEI, 2011). O valor da acurácia é calculado pela Equação 2:

$$Acurácia = \frac{Verdadeiros\ Positivos + Verdadeiros\ Negativos}{Positivos + Negativos} \quad (2)$$

O oposto da acurácia é a **taxa de erro**, que mostra a taxa de erros (falsos positivos e falsos negativos) em relação a todos os dados no modelo. A Equação 3 demonstra a fórmula (HAN; KAMBER; PEI, 2011).

$$Taxa\ de\ erro = \frac{Falsos\ Positivos + Falsos\ Negativos}{Positivos + Negativos} \quad (3)$$

A **sensibilidade** é a proporção do número de verdadeiros positivos que foram corretamente classificados. A sensibilidade demonstra a capacidade do modelo de predizer corretamente uma classe para dados que realmente possuem essa classe. A Equação 4 a seguir demonstra a fórmula (HAN; KAMBER; PEI, 2011).

$$sensibilidade = \frac{Verdadeiros\ Positivos}{Positivos} \quad (4)$$

A **especificidade** é a proporção de verdadeiros negativos que foram corretamente classificados e demonstra a capacidade do modelo em predizer de forma correta a ausência da classe para casos que realmente não a têm (HAN; KAMBER; PEI, 2011). A Equação 5 demonstra a fórmula da especificidade.

$$especificidade = \frac{Verdadeiros\ Negativos}{Negativos} \quad (5)$$

A sensibilidade e especificidade variam em direções opostas dentro do modelo, sendo muito difícil conseguir um modelo de predição perfeito, onde seja possível alcançar 100% em ambas as métricas, desta forma uma outra métrica conhecida como **eficiência** auxilia na identificação de um melhor balanço entre ambas, através de uma média aritmética da sensibilidade e especificidade, como mostra a Equação 6 a seguir (CLÉSIO, 2014).

$$eficiência = \frac{Sensibilidade + Especifidade}{2} \quad (6)$$

Além de algumas das métricas já descritas, Han; Kamber e Pei (2011) também descreve a precisão, *recall* e *f-score*. A **precisão** é uma medida de exatidão, deixando claro que as porcentagens de predições positivas são realmente isso, já ***recall*** é uma medida de completude, dizendo qual porcentagem de predições positivas são realmente positivos, sendo o mesmo que a sensibilidade. Suas formulas estão nas Equações 7 e 8 respectivamente.

$$precisão = \frac{Verdadeiros\ positivos}{Verdadeiros\ positivos + Falsos\ positivos} \quad (7)$$

$$recall = \frac{Verdadeiros\ positivos}{Verdadeiros\ positivos + Falsos\ negativos} \quad (8)$$

Por fim, ***F-Score*** é uma alternativa ao uso do *recall* e precisão, combinando ambas em uma única medida, gerando um meio harmônico entre precisão e sensibilidade como mostra a Equação 9.

$$F - Score = \frac{2\ x\ precisão\ x\ recall}{precisão + recall} \quad (9)$$

2.4.2 MÉTODOS DE AVALIAÇÃO DE ALGORITMOS PREDITIVOS

Os algoritmos preditivos sempre utilizam os conjuntos de treino e teste. No primeiro, uma parte da base é utilizada para treinar o algoritmo e, no segundo são realizados testes. Para proporcionar um cálculo confiável da sua capacidade preditiva é preciso garantir que os dados testados não sejam iguais aos dados utilizados para treinar o algoritmo, já que utilizar os mesmos dados em ambos não é uma boa forma de verificar o seu desempenho, para garantir que isso não ocorra, a divisão da base é realizada (REZENDE; ABREU, 2003).

Na literatura existem diversos métodos tradicionais para realizar a avaliação de modelos preditivos, dentre os métodos tradicionais dois são bastante utilizados e populares, sendo eles os métodos *Holdout* e *k-fold cross validation.* O método *Holdout* é bastante utilizado quando o objetivo é gerar um único modelo de conhecimento, que pode ser aplicado posteriormente a um sistema de apoio a decisão. O método divide aleatoriamente os dados de

uma base em dois conjuntos com porcentagens fixas de tamanho, onde um conjunto é utilizado para realizar o treinamento do algoritmo e outro para realizar os testes, normalmente cerca de 2/3 da base são alocados para treinamento e 1/3 para teste, além disso, sempre a quantidade de dados alocado para o treinamento é maior que metade da base de dados. Após a divisão, a geração do modelo e o cálculo de erros é realizada (REZENDE; ABREU, 2003).

O método *Holdout* é indicado para grandes bases de dados, seu uso em pequenas bases pode gerar uma grande variação no cálculo do erro. Para mitigar esses problemas existe um outro método chamado *k-fold cross validation,* o método realiza a divisão de uma base de dados em *k* subconjuntos com o mesmo tamanho e mutuamente exclusivos, após isso um dos subconjuntos é selecionado para teste dos dados e os subconjuntos restantes são agrupados para serem utilizados no treinamento do algoritmo, assim o algoritmo vai testar *k* subconjuntos e gerar *k* modelos, alternando o subconjunto de teste a cada novo modelo gerado (BERGMEIR; HYNDMAN; KOO, 2017).

O método *k-fold cross validation* gera uma investigação completa da variação do modelo em relação aos dados, garantindo uma acurácia precisa, entretanto por conta disso, o seu custo operacional é muito elevado, tornando-o recomendável apenas para pequenas bases de dados. (BERGMEIR; HYNDMAN; KOO, 2017).

Ambos os métodos *k-fold* e *Holdout* possuem limitações quanto ao seu uso em bases de dados temporais, por conta da divisão aleatória realizada pelo método *holdout*, dados do futuro podem ser utilizados no conjunto de treino, o que não é correto, pois o método tentaria predizer os dados já estando ciente dos seus valores futuros*,* já no método *k-fold cross*, a não ser que o subconjunto selecionado seja composto de forma a possuir apenas instâncias que correspondam aos últimos dados (em ordem cronológica) da base, o método também tentará predizer dados já estando ciente dos seus valores futuros (LIMA JÚNIOR, 2017).

Como solução para esse problema, Lima Júnior (2017) propôs o método da janela deslizante, o método divide a base mantendo sua ordem cronológica, onde duas janelas são definidas, com base no tamanho em porcentagem de cada janela os conjuntos de treino e teste são formados, com isso, o método desliza os limites dos conjuntos por toda a base de dados, indo das instâncias mais antigas cronologicamente na base até as instâncias mais atuais.

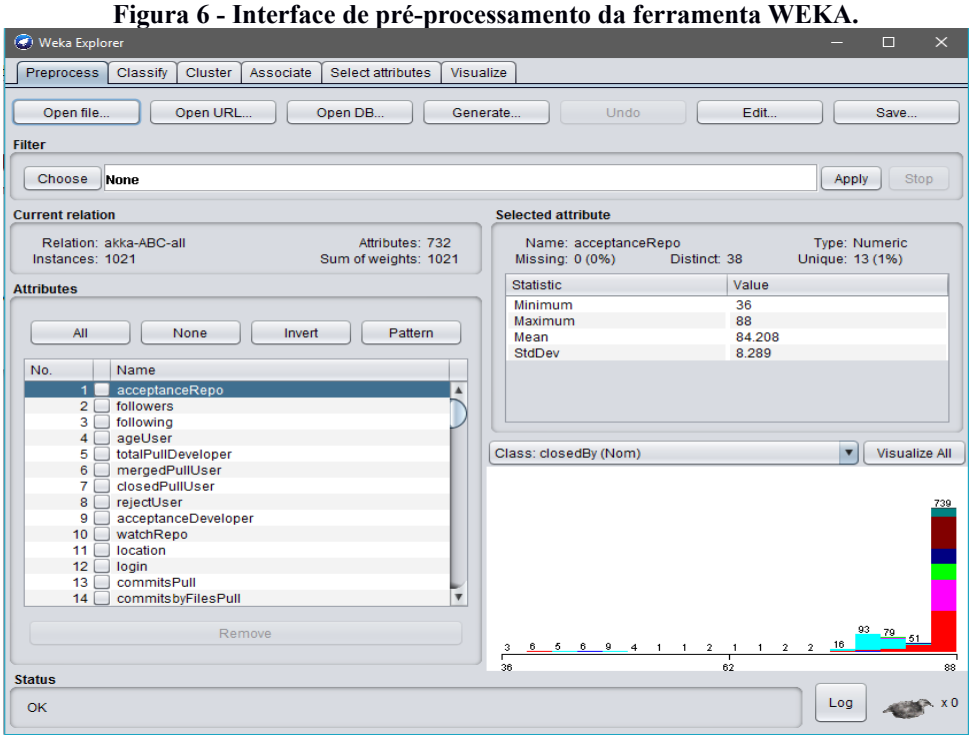
2.5 FERRAMENTA WEKA

A ferramenta *Waikato Environment for Knowledge Analysis* (WEKA[4]) é um software livre escrito na linguagem de programação Java e foi desenvolvido pela Universidade de Waikato, Nova Zelândia. O programa é composto por uma coleção bastante robusta de algoritmos para mineração de dados podendo executar diversas técnicas diferentes, além de conter vários recursos para o pré-processamento dos dados e *plugins* desenvolvidos pela comunidade que expandem ainda mais a ferramenta (HALL *et al*., 2009).

A ferramenta WEKA possui uma interface bastante simples e modular, isso permite que os algoritmos de aprendizagem e as várias ferramentas utilizadas para a transformação dos dados sejam aplicadas facilmente, não havendo a necessidade de escrever nenhum código, o formato padrão da ferramenta é o *Attribute-Relation File Format* (ARFF), formato especifico que é composto por uma tabela relacional. A própria ferramenta permite transformar qualquer base de dados para o formato ARFF (HALL *et al*., 2009).

Na Figura 6, é mostrada a interface de pré-processamento da ferramenta WEKA, que é utilizada para realização da visualização e pré-processamento dos dados.

Figura 6 - Interface de pré-processamento da ferramenta WEKA.



Fonte: Autoria própria.

4 Disponível em: <https://www.cs.waikato.ac.nz/ml/index.html> Acesso em: 02 de junho de 2018

A ferramenta WEKA possui uma extensa documentação, além disso no seu site oficial é disponibilizada uma API[5], permitindo que seus algoritmos e componentes possam ser reutilizados em códigos escritos na linguagem Java (HALL *et al*., 2009).

Os experimentos realizados nesta pesquisa utilizaram bases de dados temporais no contexto de *pull request*, neste aspecto, as seções 2.6 e 2.7 a seguir tratam de descrever a gerência de configuração de software (processo que gera os *pull request*), seguindo com a explicação do que é um *pull request* e seu funcionamento.

2.6 GERÊNCIA DE CONFIGURAÇÃO DE SOFTWARE

Garantir o controle das modificações realizadas em um software é uma das tarefas da Gerência de Configuração de Software (GCS). Segundo Ieee (1990), a GCS é uma disciplina da Engenharia de Software (ES) que visa a qualidade do software, aplicando procedimentos técnicos e administrativos para identificar as características físicas e funcionais de um Item de Configuração[6] (IC), também controla todas as alterações realizadas nos IC, armazena e trata o processamento das modificações dos IC, avalia e revisa o estágio dos IC e gerência a liberação e entrega do software.

Muitos sistemas de software se encontram em constante evolução. Atualmente a realização de modificações em códigos já anteriormente desenvolvidos (artefato), consome cerca de 75% dos custos de um software ao longo de todo o seu ciclo de vida, tornando a manutenção o principal custo do desenvolvimento de software na era atual, neste aspecto a GCS é de suma importância, já que todas as modificações realizadas no software devem ser estritamente controladas para evitar inconsistências e perda de configuração, permitindo assim maximizar a produtividade e minimizar os erros (FILHO, 2003; ARAÚJO; SPÍNOLA, 2009).

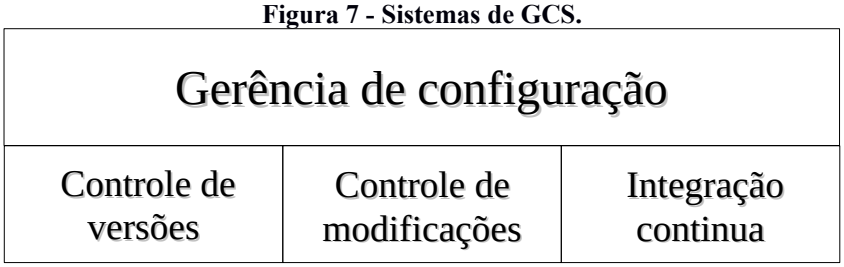
A GCS não define formas de executar modificações nos artefatos de software, sua função é auxiliar o controle e acompanhamento de tais modificações, controlando e

5 Conjunto de rotinas e padrões de programação para acesso a um aplicativo de software ou plataforma baseado na Web

6Qualquer componente que necessita ser configurado com o objetivo de se entregar um serviço de TI, como, por exemplo, um documento de requisitos ou trecho de código

notificando todas as correções, adições e modificações realizadas durante o ciclo de vida de um software, permitindo assim, um processo rastreável e sistemático (MURTA, 2006). De acordo com Murta (2006), GCS pode ser dividida em 3 sistemas como demonstra a Figura 7.

Figura 7 - Sistemas de GCS.



Fonte: Adaptação de DEVMEDIA 2009.

Como somente controle de versões e controle de modificações possuem relevância para esta pesquisa, por se tratarem dos sistemas que geram os *pull request*, somente estes serão explicados no restante deste capítulo. Segundo Asklund *et al*. (2001), **os sistemas de controle de versões (SCV)** é a parte central de um GCS e a funcionalidade principal na maioria das ferramentas relacionadas a GCS. Para Pressman (2002), o SCV tem a função de administrar as diversas versões dos IC que são criados durante a Engenharia Software, utilizando para este fim combinações de procedimentos e ferramentas.

Todo SCV é composto por duas partes: o **repositório** e a **área de trabalho**. A parte que fica responsável por armazenar o histórico contendo toda evolução realizada no projeto é o repositório, os desenvolvedores não possuem acesso direto aos arquivos armazenados no repositório, não podendo assim modificá-los diretamente. Para que seja possível realizar modificações no repositório, os desenvolvedores precisam utilizar uma área de trabalho, um espaço de armazenamento local que possuí uma cópia de todos os IC do projeto e constantemente se mantêm monitorando e identificando todas as mudanças realizadas nos IC, gerando *feedback* ao desenvolvedor (DIAS, 2016).

Para enviar modificações realizadas nos IC ao repositório é preciso sincronizar a área de trabalho (local onde as modificações foram realizadas) com o repositório (local onde as modificações serão armazenadas). Segundo Dias (2016), para realizar isso é preciso utilizar os comandos *commit* e *update*. Ao executar o comando de c*ommit* o desenvolvedor envia ao repositório um pacote que contém todas as modificações na área de trabalho. Quanto ao comando *update* a sua função é realizar o inverso, desta forma, ao executá-lo todas as novas modificações contidas no repositório são enviadas para a área de trabalho, sendo necessário

sincronizar a área de trabalho (em casos onde novos *commits* tenha sido realizados após o último *update* do usuário) antes de realizar um *commit*, garantindo que todos os IC (não modificados) na área de trabalho sejam iguais aos IC armazenados no repositório (NAGEL, 2005; DIAS, 2016).

Quanto aos **sistemas de controle de modificações** (SCM), são sistemas encarregados de realizar o controle da configuração de um software automaticamente, neles todas as mudanças geram informações, que são armazenadas e direcionadas às pessoas interessadas (MURTA, 2006).

Segundo Lima Júnior (2017), o processo de funcionamento de um SCM se inicia no momento que uma nova funcionalidade precisa ser criada ou um defeito é encontrado no sistema. Uma solicitação de modificação (*issue)* precisa ser enviada a equipe principal, descrevendo o que foi modificado ou adicionado no IC. A equipe então verifica cada um dos *issue*, decidindo quais devem ser adicionados ao repositório principal e quais devem ser rejeitados, a ordem que isso será feito e qual desenvolvedor ficará encarregado de cada *issue* (CAVALCANTI *et al*. apud MURTA, 2006, 2014).

Ainda segundo Lima Júnior (2017), no geral existem 3 fases a qual um *issue* pode passar, solicitação, modificação e verificação. Solicitação é a fase onde os dados sobre o *issue* são preenchidos pelo solicitante[7], como o nome, título, descrição, entre outros. Modificação é a fase quando o *issue* pode realmente ser aceito no repositório, dependendo da sua relevância para o projeto. Verificação é a última fase, onde se garante que o *issue* foi corretamente integrado ao repositório.

2.7 *PULL REQUEST*

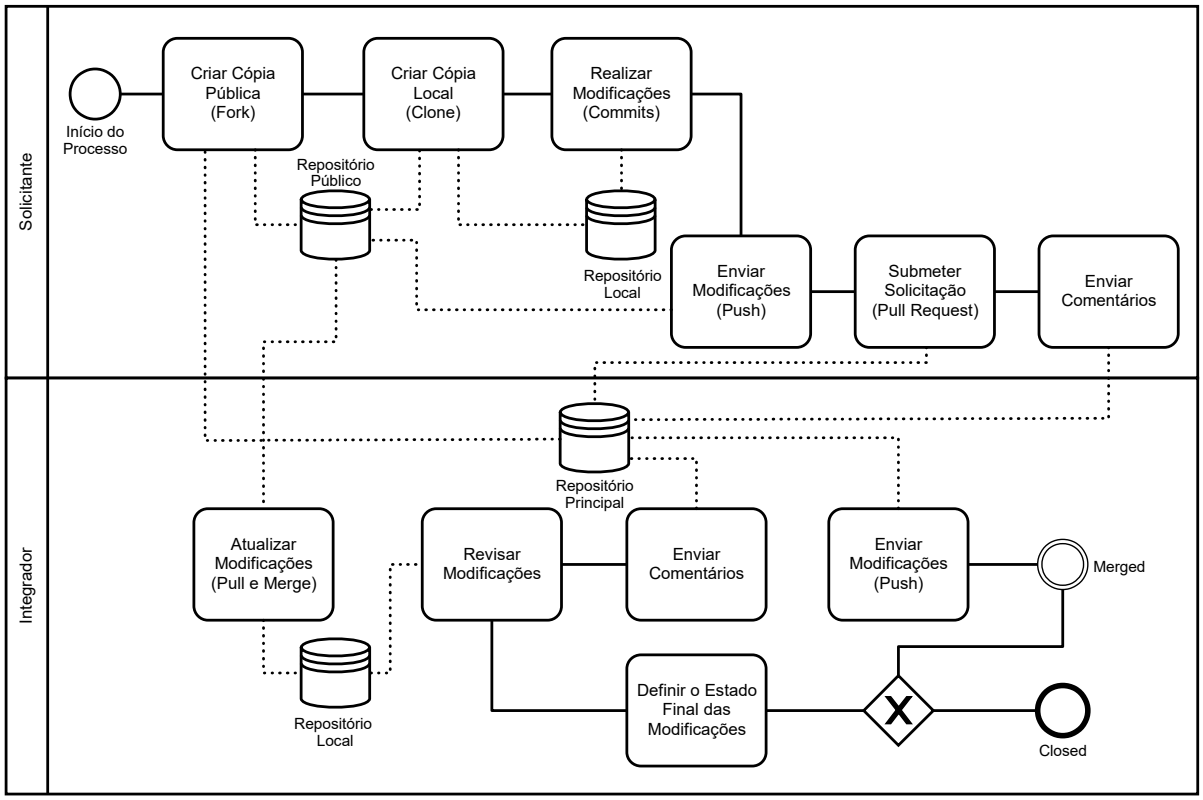
Em vários projetos, principalmente projetos *open souce* ou projetos que utilizem a gerência de configuração de software existe um grande número de desenvolvedores contribuindo para criação do software. Vários destes desenvolvedores não fazem parte da equipe principal de desenvolvimento e não tem acesso de escrita direto ao repositório

7Desenvolvedor que faz a solicitação do *issue.*

principal. Em alguns casos, mesmo em projeto fechados, onde somente existe a equipe principal em atuação, tal característica de escrita se mostra presente, garantindo um maior controle sobre as modificações realizadas (CHACON; STRAUB, 2014).

Para enviar suas contribuições ao repositório principal é preciso realizar um *inssue* denominado *pull request*, basicamente, uma forma de indiciar que uma contribuição está pronta e pode ser anexada ao código final. Inicialmente o usuário cria uma cópia publica do repositório, para então cloná-lo no seu repositório local, todas as modificações são acompanhadas pelo repositório local, e então tais modificações são enviadas ao repositório publico, gerando assim o *pull request*, Ao se criar um *pull request,* um registro é criado no servidor indicando todas as diferenças, entre o código atual modificado pelo desenvolvedor e o código alvo armazenado no repositório principal, isso permite aos responsáveis da equipe principal revisarem o código com maior facilidade, para só então decidir se o *pull request* deve ser aceito ou recusado (CHACON; STRAUB, 2014), a Figura 8 demonstra um esquema do processo de criação do *pull request* citado neste paragrafo.

Figura 8 – Esquema de funcionamento do *pull request.*



Fonte: Lima Júnior (2018).

Um *pull request* pode assumir 3 estados, **aberto** indica um *pull request* que ainda não

recebeu um estado final, **aceito** que indica um *pull request* revisado e integrado ao código principal e **fechado**, que indica um *pull request* que foi recusado. Quando um *pull request* é revisado e considerado ineficiente o mesmo é recusado, ao recusar os responsáveis pela análise normalmente escrevem um comentário indicando o motivo pelo qual o *pull request* foi recusado, permitindo assim ao criador ter um *feedback* que pode ser usado para melhorar o código a fim de receber a aceitação em um futuro *pull request,* caso o *pull request* seja considerado útil para o projeto o mesmo é aceito, tendo o seu código integrado ao código presente no repositório principal (LIMA JÚNIOR, 2017).

2.8 CONSIDERAÇÕES SOBRE O CAPÍTULO

A mineração de dados se destaca entre as etapas do processo de KDD, necessitando de técnicas e ferramentas específicas para a sua utilização, como esta é uma pesquisa referente ao desenvolvimento de variações para um método de avaliação de algoritmos preditivos, este capítulo apresentou uma revisão bibliográfica sobre vários aspectos da mineração de dados, permitindo assim embasar cientificamente os conceitos que sustentam essa pesquisa, dando ênfase a mineração preditiva, suas técnicas, métricas e modelos de avaliação Além disso, a GCS e *pull request,* também foram apresentados, deixando claro o que são e como funcionam, pois, o experimento realizado no capítulo 4 desta pesquisa utiliza bases de dados no contexto de *pull request*.

O capítulo a seguir apresenta a extensão original desenvolvida por Lima (2017) para a ferramenta WEKA, além das variações implementadas nesta pesquisa.

3 IMPLEMENTAÇÃO DE VARIAÇÕES NO MÉTODO DE AVALIAÇÃO DA EXTENSÃO TRAINING TEST SLIDING VALIDATION (TTSV)

A ferramenta WEKA permite o desenvolvimento e a integração de novas extensões, incorporando funcionalidades originalmente inexistentes na ferramenta. As extensões criadas são distribuídas através de pacotes instaláveis, gerenciados dentro da própria ferramenta, permitindo que outras pessoas tenham acesso às extensões e possam utilizá-las.

No ano de 2017, na Universidade Federal do Acre (UFAC), o trabalho de conclusão de curso do aluno Max Wilian de Lima implementou uma extensão para a ferramenta WEKA, denominada *Training Test Sliding Validation*. A extensão se baseia no método da janela deslizante desenvolvido na pesquisa de doutorado do seu orientador, Manoel Limeira de Lima Júnior Almeida, como uma proposta mais coerente com o mundo real de realizar o processo de avaliação de algoritmos de classificação em bases de dados temporais.

Este capítulo está organizado da seguinte forma: a seção 3.1 aborda o funcionamento do método de janela deslizante implementado na extensão original, a seção 3.2 descreve a implementação das variações por intervalo entre datas e por número absoluto de instâncias na extensão e uma conclusão é desenvolvida na seção 3.3.

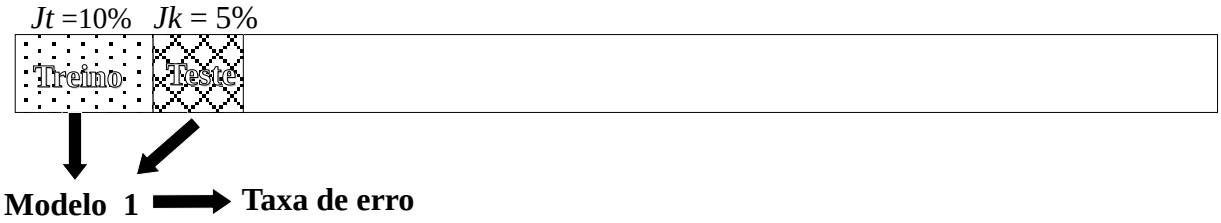
3.1 A EXTENSÃO

De acordo com Lima Júnior (2017), o grupo de técnicas de mineração que compõem

as tarefas de previsão, passa por um processo de geração dos modelos preditivos, para que seja possível gerar modelos confiáveis, tradicionalmente uma parte da base de dados deve ser dividida para realizar o treinamento e outra parte para a realização dos testes.

Segundo Lima (2017), o método implementado define duas janelas *Jt* e *Jk,* onde *Jt* define o tamanho em porcentagem do conjunto de instâncias (*t*) que será utilizada para criação do treino e *Jk* define o tamanho em porcentagem do conjunto de instancias(*k*) para realização dos testes. Considerando que a base de dados está organizada em ordem cronológica, utiliza-se as duas porcentagens previamente configuradas para "deslizar" por toda a base. A Figura 9 demonstra um exemplo de como o processo de criação dos modelos preditivos funciona, *Jt* possuirá 10% das instâncias da base e *Jk* possuirá 5%.

Figura 9 - Geração do primeiro modelo.

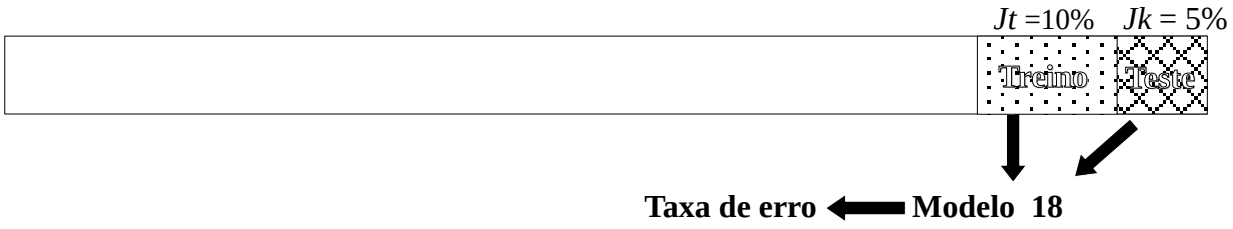


Fonte: Adaptação dos exemplos de Lima (2017).

Para gerar o primeiro modelo preditivo as instâncias que compõem o conjunto *t* são analisadas, o conhecimento adquirido é utilizado para gerar um modelo preditivo, o modelo gerado é submetido às instâncias de teste e uma taxa de erro é calculada. Após a geração do primeiro modelo, os limites dos conjuntos *t* e *k* "deslizam" *Jk* por cento dentro da base, sendo deslocados para a direita, se movendo das instâncias cronologicamente mais antigas na base, para as instâncias mais atuais.

Os próximos *Jk* por cento de instâncias na base são utilizados para compor o novo conjunto de testes *k* e os *Jt* por cento de instâncias anteriores ao novo conjunto *k*, compõem o novo conjunto de treino *t.* O processo é repetido até que o conjunto *k* alcance o fim da base ou até que não reste instâncias para formar um novo conjunto *k.* A Figura 10 demonstra a geração do último modelo preditivo.

Figura 10 - Geração do último modelo.



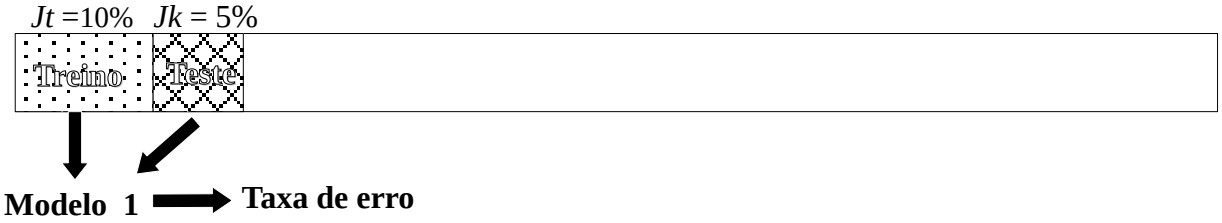
Fonte: Adaptação dos exemplos de Lima (2017).

Com a geração do último modelo, o método chega ao final da base e não existem mais instâncias para formar um novo conjunto *k,* desta forma, a média entre todas as taxas de erro é calculada conforme a quantidade de modelos gerados, a quantidade de modelos gerados é definido pela Equação 10.

$$Modelos\,gerados = \frac{100 - t}{k} \tag{10}$$

O método possui uma variação para casos onde em que é necessário treinar com todas as instâncias anteriormente utilizadas no conjunto *k*, ocorrendo a acumulação das instâncias de treinamento, a geração do primeiro modelo pode ser visto na Figura 11.

Figura 11 - Geração do primeiro modelo com acumulação.



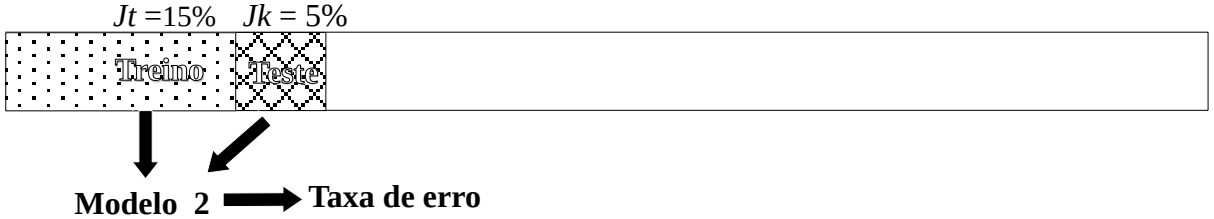
Fonte: Adaptação dos exemplos de Lima (2017).

O primeiro modelo gerado é idêntico ao exemplo anterior, a diferença da acumulação ocorre após a geração do segundo modelo, onde a medida que os limites dos conjuntos *t* e *k* deslizam pela base, *t* englobará o conjunto *k* anteriormente testado. Assim, o treinamento ocorre de forma acumulada, onde a cada vez que os limites dos conjuntos deslizarem as instâncias do novo conjunto *t* resultam da Equação 11.

$$novo\,conjunto\,t = ta + ka \tag{11}$$

Onde *ta* é o conjunto *t* atual e *ka* é o conjunto *k* atual. A geração do segundo modelo é demonstrado na Figura 12.

Figura 12 - Geração do segundo modelo com acumulação.



Fonte: Adaptação dos exemplos de Lima (2017).

Assim como no primeiro exemplo, o ciclo continuará repetindo, até que não reste instâncias o bastante para formar um novo conjunto *k*, ou até o método chegar ao fim da base. O método por porcentagem não participou de nenhum experimento desta pesquisa, pois já foi anteriormente experimentado e avaliado no trabalho de Lima (2017).

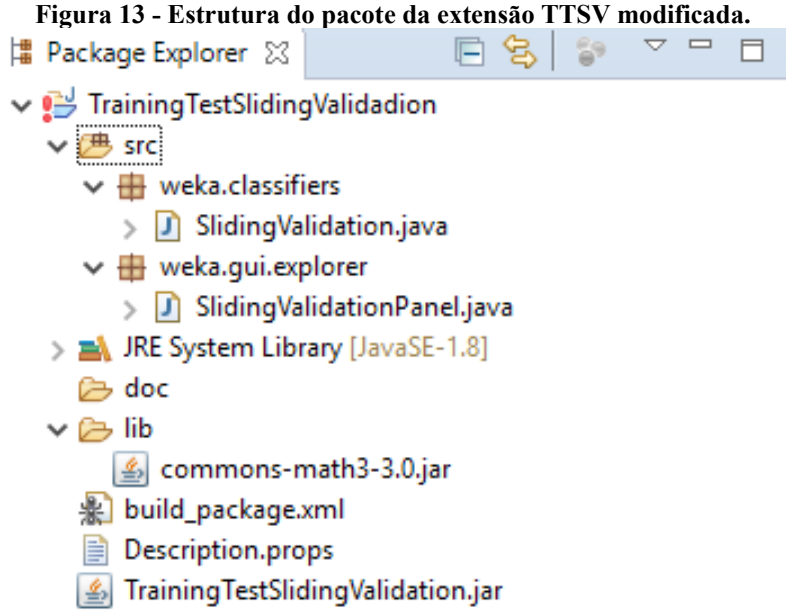
3.2 IMPLEMENTAÇÃO DE VARIAÇÕES PARA A EXTENSÃO TTSV

Ao utilizar a extensão TTSV pode ocorrer casos onde o usuário responsável pela mineração deseje realizar experimentos mais específicos, como por exemplo, treinar *N* instâncias e testar de *N* em *N*, ou treinar e testar diretamente com o número total de instâncias existentes em um intervalo de datas. Estes objetivos não podem ser alcançados apenas com o uso do método por porcentagem, tornando necessário fornecer novas opções ao usuário.

Esta pesquisa desenvolveu duas variações para o método original, a primeira variação gera os modelos preditivos através do número de instâncias, funcionando como uma flexibilização do método por porcentagem, dando maiores opções para o tamanho das janelas de treino e teste. A segunda variação gera os modelos preditivos com base em um intervalo de datas entre as instâncias existentes na base.

Ambas variações foram implementadas na linguagem de desenvolvimento JAVA, por se tratar da linguagem utilizada na ferramenta WEKA e também a linguagem utilizada no desenvolvimento da extensão original. A estrutura do pacote da extensão modificada continuou igual à extensão original de Lima (2017) e pode ser vista na Figura 13 a seguir.

Figura 13 - Estrutura do pacote da extensão TTSV modificada.



Fonte: Elaboração própria.

Nenhuma mudança foi realizada na classe *SlidingValidation.java*, responsável pela aplicação dos algoritmos da ferramenta WEKA e métodos para geração da matriz de confusão, calculo kappa, entre outros. Somente a classe *SlidingValidationPanel.Java* foi

modificada, classe responsável pela implementação dos objetos gráficos da extensão, recebimento das janelas *t* e *k*, preparo dos dados e geração dos conjuntos *t* e *k*, fazendo reúso do método original de recebimento e organização dos dados para geração dos conjuntos (*startClassifier*), originalmente o arquivo possuía 881 linhas de código e após a implementação das variações o número aumentou para 2061 linhas de código.

As implementações consistiram da adição de duas novas telas e novos métodos para o funcionamento das variações implementadas, como por exemplo, o método *startClassifierInstances* da Figura 14, que é acionado após o usuário clicar sobre o botão *start* na interface da extensão (Figura 14 A), o método apaga o botão *start* (evitar duplos cliques) e ativa o botão *stop* (permitindo parar a avaliação a qualquer momento), Em seguida, o método zera as principais variáveis utilizadas para o cálculo da matriz de confusão e armazenamento de instâncias (Figura 14 B) daí, inicia os preparativos para a avaliação.

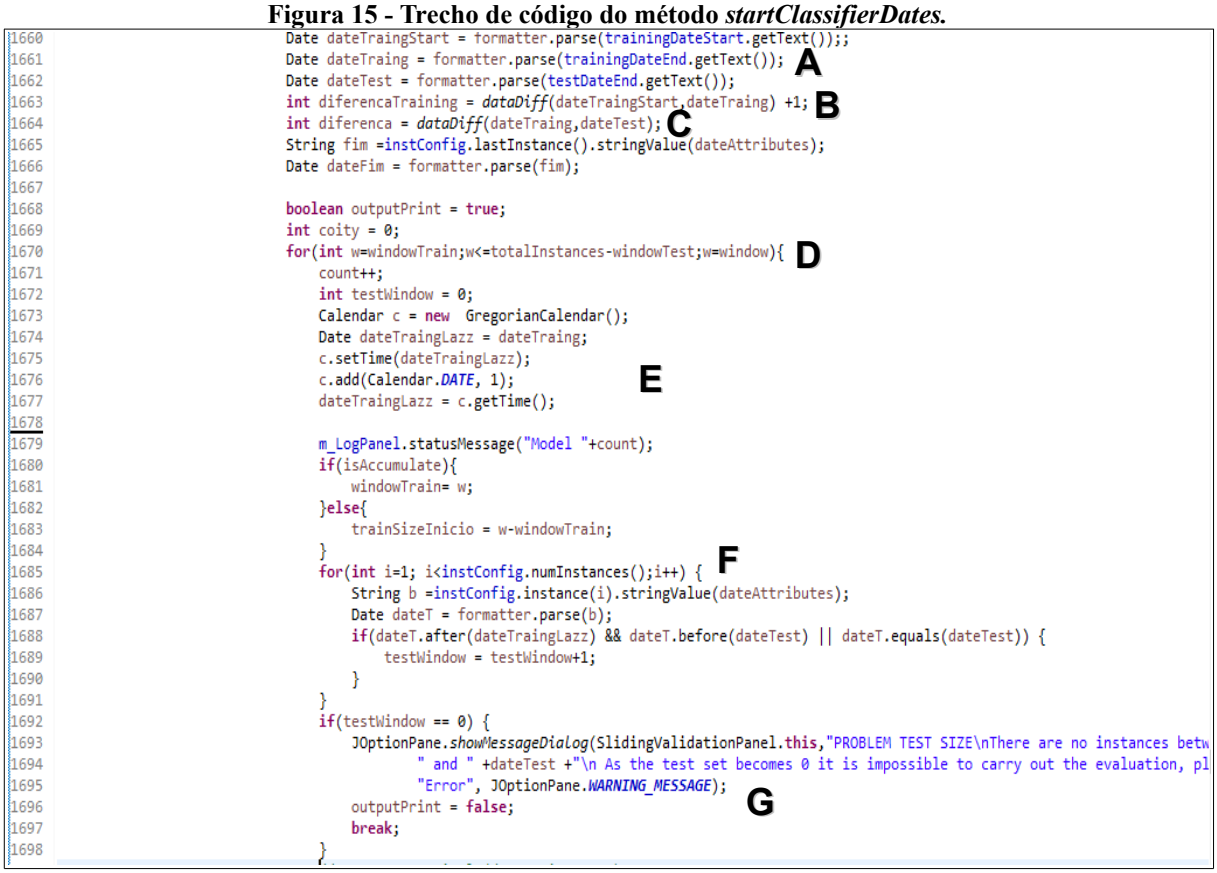
Figura 14 - Trecho de código do método *startClassifierInstances*.

```
1263    protected void startClassifierInstances() {
1264        if (m_RunThread == null) {
1265            synchronized (this) {
1266                btnStart.setEnabled(false);     A
1267                btnStop.setEnabled(true);
1268            }
1269            m_RunThread = new Thread() {
1270                @Override
1271                public void run() {
1272
1273                    if (m_LogPanel instanceof TaskLogger) {
1274                        ((TaskLogger) m_LogPanel).taskStarted();
1275                    }
1276                    m_ClassCombo.setEnabled(false);
1277                    // Copy the current state of things
1278                    m_LogPanel.statusMessage("Setting up...");
1279                    CostMatrix costMatrix = null;
1280                    Instances inst = new Instances(m_Instances);
1281                    DataSource source = null;                          B
1282                    Instances userTestStructure = null;
1283                    ClassifierErrorsPlotInstances plotInstances = null;
1284
1285                    // for timing
1286                    long trainTimeStart = 0, trainTimeElapsed = 0;
1287                    long testTimeStart = 0, testTimeElapsed = 0;
1288
1289                    int classIndex = m_ClassCombo.getSelectedIndex();
1290                    inst.setClassIndex(classIndex);
1291                    Classifier classifier = (Classifier) m_ClassifierEditor.getValue();
1292                    Classifier template = null;
1293                    try {
```

Fonte: Elaboração própria.

Outro exemplo é um trecho do método *startClassifierDates* na Figura 15.

Figura 15 - Trecho de código do método *startClassifierDates*.



```
1660                    Date dateTraingStart = formatter.parse(trainingDateStart.getText());;
1661                    Date dateTraing = formatter.parse(trainingDateEnd.getText());
1662                    Date dateTest = formatter.parse(testDateEnd.getText());
1663                    int diferencaTraining = dataDiff(dateTraingStart,dateTraing) +1;
1664                    int diferenca = dataDiff(dateTraing,dateTest);
1665                    String fim =instConfig.lastInstance().stringValue(dateAttributes);
1666                    Date dateFim = formatter.parse(fim);
1667
1668                    boolean outputPrint = true;
1669                    int coity = 0;
1670                    for(int w=windowTrain;w<=totalInstances-windowTest;w=window){
1671                        count++;
1672                        int testWindow = 0;
1673                        Calendar c = new  GregorianCalendar();
1674                        Date dateTraingLazz = dateTraing;
1675                        c.setTime(dateTraingLazz);
1676                        c.add(Calendar.DATE, 1);
1677                        dateTraingLazz = c.getTime();
1678
1679                        m_LogPanel.statusMessage("Model "+count);
1680                        if(isAccumulate){
1681                            windowTrain= w;
1682                        }else{
1683                            trainSizeInicio = w-windowTrain;
1684                        }
1685                        for(int i=1; i<instConfig.numInstances();i++) {
1686                            String b =instConfig.instance(i).stringValue(dateAttributes);
1687                            Date dateT = formatter.parse(b);
1688                            if(dateT.after(dateTraingLazz) && dateT.before(dateTest) || dateT.equals(dateTest)) {
1689                                testWindow = testWindow+1;
1690                            }
1691                        }
1692                        if(testWindow == 0) {
1693                            JOptionPane.showMessageDialog(SlidingValidationPanel.this,"PROBLEM TEST SIZE\nThere are no instances betw
1694                                    " and " +dateTest +"\n As the test set becomes 0 it is impossible to carry out the evaluation, pl
1695                                    "Error", JOptionPane.WARNING_MESSAGE);
1696                            outputPrint = false;
1697                            break;
1698                        }
```


Fonte: Elaboração própria.

O trecho de código recebe os valores inseridos nas janelas do método por número absoluto de instâncias (Figura 15 A) e os converte de *String* para Data, em seguida, calcula o número de dias para a janela *t* (Figura 15 B) e o número de dias para a janela *k* (Figura 15 C). Na sequência, um laço é iniciado (Figura 15 D) para verificar toda a base de dados, gerando os conjuntos *t* e *k,* a cada repetição do laço, até que o método chegue ao final. Em seguida, algumas variáveis são instanciadas e precisam ser reinicializadas a cada ciclo de repetição (Figura 15 E).

Um novo laço é criado (Figura 15 F), onde todas as instâncias são verificadas, aquelas que possuem datas dentro da faixa de tempo definida para a janela de teste, incrementam o valor da variável *testWindow,* variável responsável por definir quantas instâncias serão testadas no ciclo de repetição atual. Por fim, caso nenhuma instância possua datas dentro da faixa definida, *testWindow* permanece com valor 0 e uma mensagem de erro é informada ao usuário (Figura 15 G).

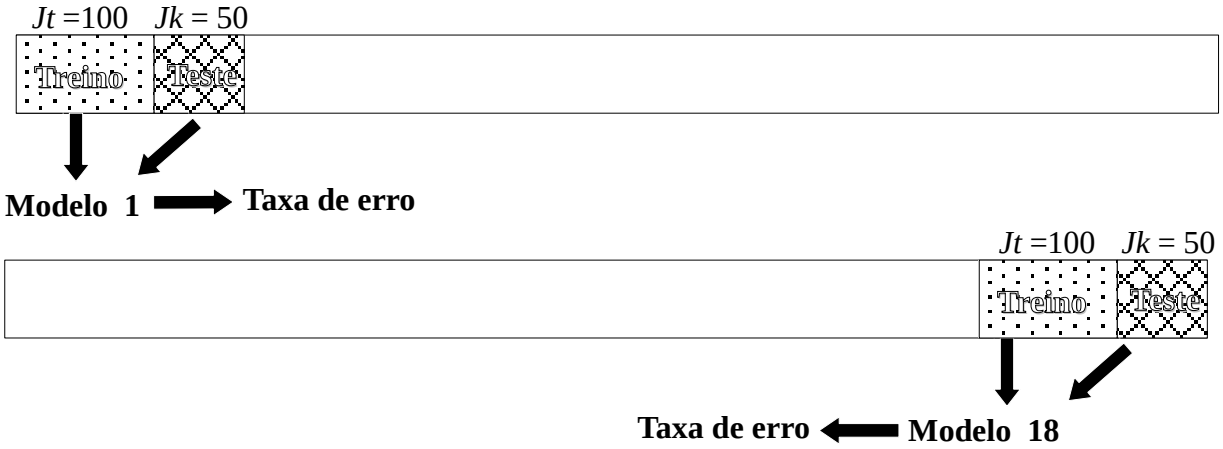
3.2.1 MÉTODO POR NÚMERO ABSOLUTO DE INSTÂNCIAS

A variação por número absoluto de instâncias é bastante parecida com o método original por porcentagens, sua diferença está nos seus campos de entrada de dados que aceitam valores inteiros em vez de porcentagens, permitindo definir janelas de treino e teste muito mais específicas e como existem mais opções, é possível atender um número maior de situações.

A variação também permite criar um número maior de modelos preditivos. Enquanto o método por porcentagem é capaz de gerar $(100-t)/k$ modelos preditivos, o método por número absoluto de instâncias pode gerar $(N-t)/k$ modelos preditivos, onde N é igual ao número total de instâncias da base de dados.

O funcionamento da variação é igual ao método por porcentagem, executando o mesmo ciclo de geração dos modelos, calculo da taxa de erros e deslizamento dos conjuntos *t* e *k* para a direita, além da capacidade de executar a avaliação acumulando ou não acumulando o conjunto *t,* um exemplo da geração de modelos sem acumulação será demonstrado na Figura 16, para este exemplo será definida uma base com 1.000 instâncias, onde *Jt* = 100 instâncias e *Jk* = 50 instâncias.

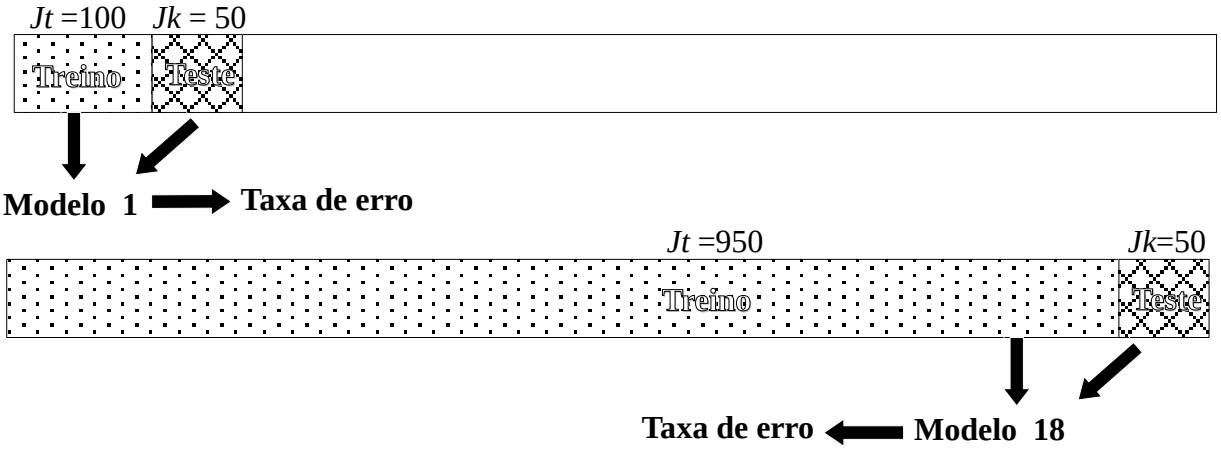
Figura 16 - Geração do modelo inicial e final do método por número absoluto de instâncias sem acumulação.



Fonte: Elaboração própria.

Assim como no método original, a variação continua executando até alcançar o fim da base, gerando 18 modelos preditivos. Na Figura 17 a geração de modelos com acumulação será demonstrada.

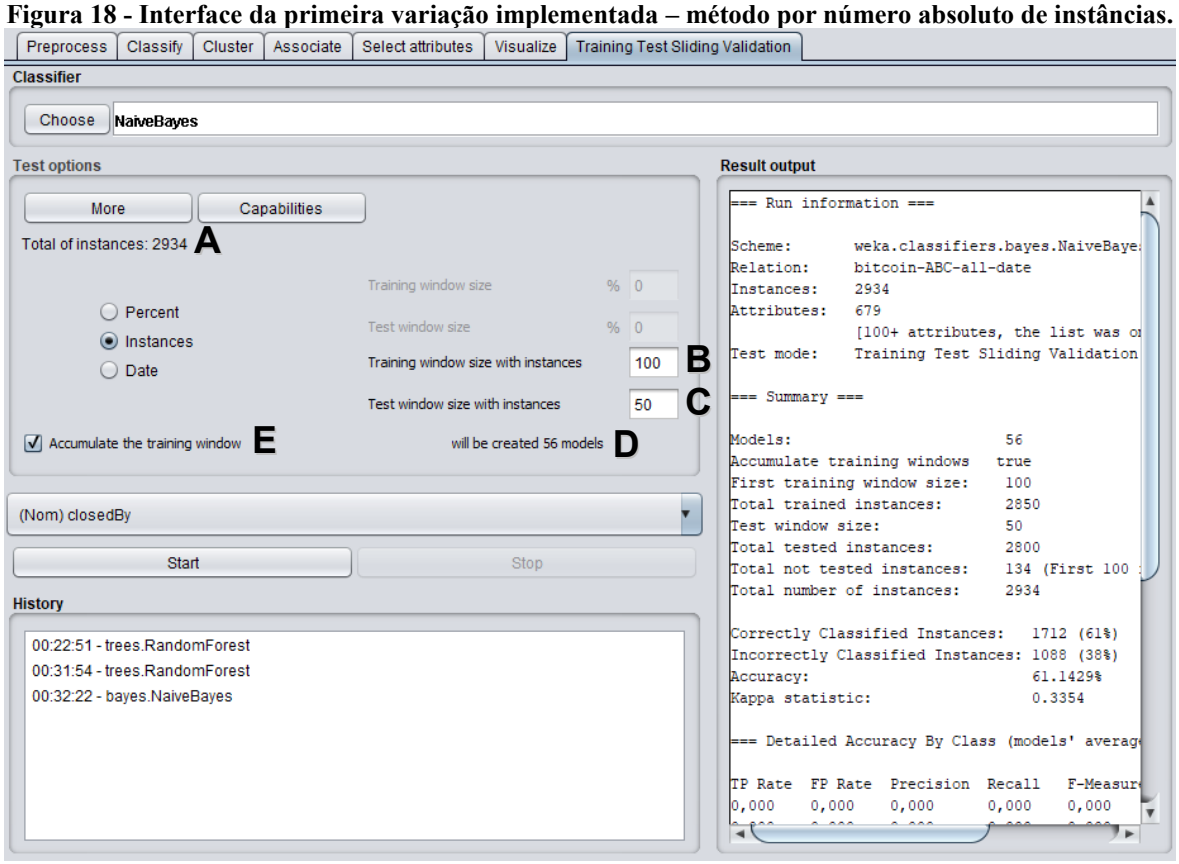
Figura 17 - Geração do modelo inicial e final do método por número de instâncias com acumulação.



Fonte: Elaboração própria.

Para que seja possível conhecer o número total de instâncias da base de dados, o campo *total of instances* (Figura 18 A) é exibido na extensão ao abrir qualquer base de dados na ferramenta WEKA. Ao inserir o valor das janelas *Jt* (Figura 18 B) e *Jk* (Figura 18 C), a quantidade de modelos que será gerada (Figura 18 D) é mostrada logo abaixo, e a caixa de seleção (Figura 18 E) é utilizada para definir se o treino vai ou não acumular as instâncias *t*. A interface da primeira variação do método de avaliação pode ser vista na Figura 18.

Figura 18 - Interface da primeira variação implementada – método por número absoluto de instâncias.



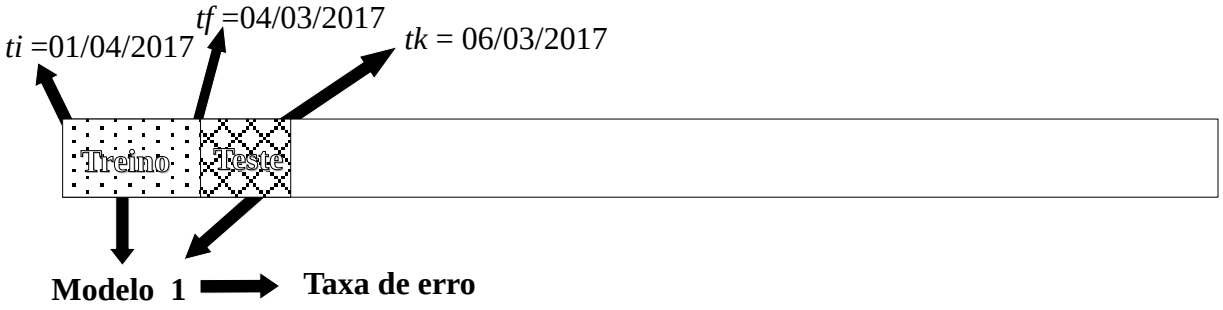
Fonte: Elaboração própria.

3.2.2 MÉTODO POR INTERVALO DE DATAS

Assim como o método por número absoluto de instâncias, no método por intervalo de datas os ciclos de geração dos modelos, o cálculo da taxa de erro e o deslizamento para a direita são os mesmos, além de que o método também é capaz de executar suas avaliações acumulando e não acumulando instâncias do conjunto *t*. A diferença neste método ocorre no tamanho dos conjuntos *t* e *k.* Em vez de valores fixos inseridos pelo usuário, o tamanho é gerado dinamicamente ao longo da execução do método, para isso o usuário insere uma data inicial para a janela de treino (*ti)*, uma data final para a janela de treino (*tf)* e uma data limite para a janela de testes (*tk)*, dessa forma as **instâncias que possuem datas entre ou iguais a *ti* e *tf* irão compor o conjunto *t* e as instâncias com datas um dia após *tf* e igual ou anteriores a *tk* compõem o conjunto *k***.

O funcionamento do método será demonstrado na Figura 19, como exemplo será definida uma base de dados com a data mais antiga sendo 01/03/2017 e a data mais recente sendo 01/04/2017. Para *ti* será escolhida a data 01/03/2017, para *tf* será escolhida a data 04/03/2017, enquanto que *tk* receberá 06/03/2017.

Figura 19 - Geração do primeiro modelo de data.



Fonte: Elaboração própria.

Após a geração do modelo inicial, ambos os conjuntos deslizam para a direita, a distância deste deslocamento é definido pelo número de dias entre *tf* e *tk,* para definir as novas datas das variáveis *ti, tf* e *tk* 3 fórmulas são seguidas. Inicialmente *tf* recebe a data de *tk*, seguindo a Equação 10.

$$tf = tk \tag{10}$$

Em seguida, *ti* recebe a data da nova variável *tf* subtraída pelo número de dias existentes entre *ti* e *tf* antes da aplicação da Equação 10, como mostra a Equação 11, sendo *tfO* e *tiO* as datas antes da modificação causada pela Equação 10.

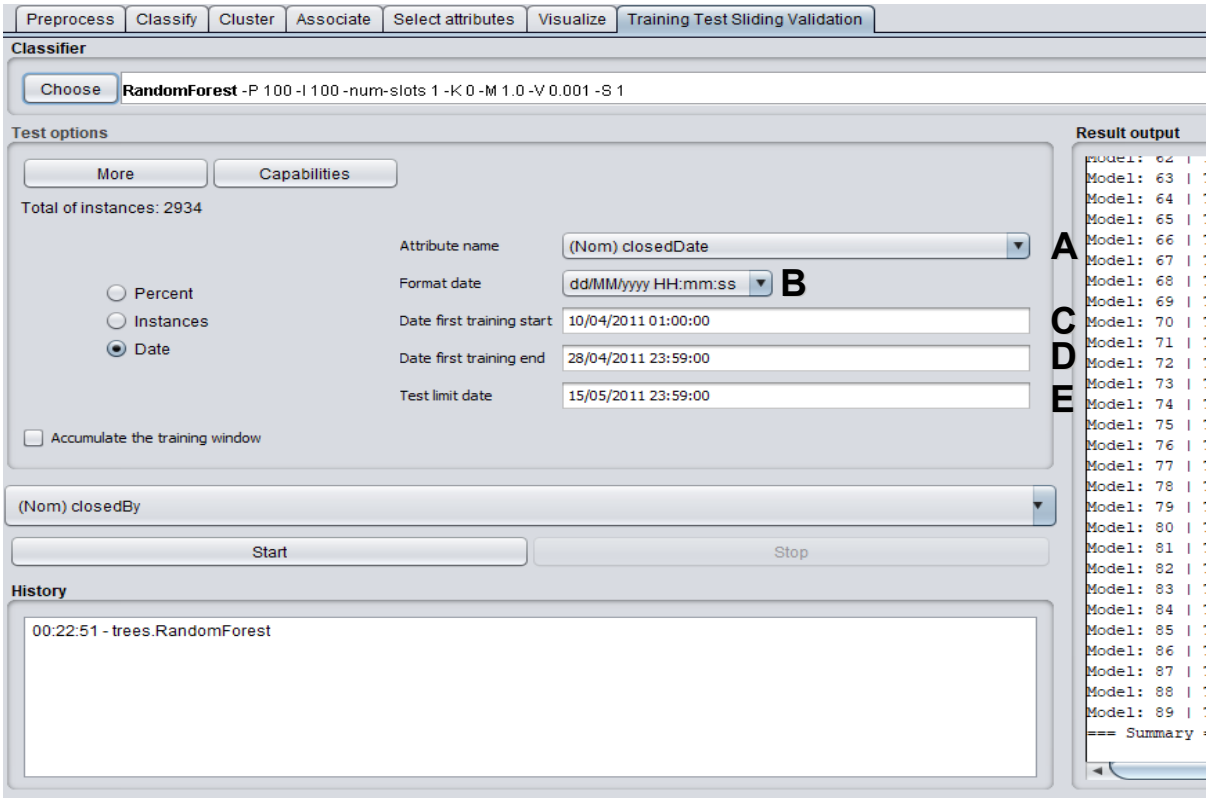
$$ti = tf - (tfO - tiO) \tag{11}$$

Por fim, a data de *tk* é incrementada com número de dias entre *tf* e *tk*, como mostra a Equação 12, sendo *tkO* e *tfO* as datas antes da modificação causada pela Equação 10 e Equação 11.

$$tk = tk + (tkO - tfO) \tag{12}$$

O método continua gerando modelos e se deslocando até alcançar a última data da base, ou até não haver dias para formar um novo conjunto de testes. Para utilizar a variação por intervalo de datas, é preciso que o usuário tenha um maior conhecimento sobre a base de dados sendo utilizada, pois é preciso escolher qual atributo da base de dados fornece a característica que a torna temporal, ou seja, registra as datas e horas, essa escolha é feita em uma caixa de seleção (Figura 20 A). Uma segunda caixa de seleção (Figura 20 B) é utilizada para formatar os campos de inserção das datas, selecionando o formato que o atributo de data original da base de dados possui, garantido que a data inserida nos campos possua o mesmo formato da data contida na base de dados, evitando possíveis erros, os 3 últimos campos *date first training start* (*ti*) (Figura 20 C), *date first training end* (*tf*) (Figura 20 D) e *test limit date* (*tk*) (Figura 20 E) são utilizados para a inserção das datas para avaliação, a interface da segunda variação pode ser vista na Figura 20.

Figura 20 - Interface da segunda variação implementada – método por intervalo de datas.



Fonte: Elaboração própria.

Vale lembrar que ao utilizar a extensão não é necessário fazer uso da opção clássica

de classificação da ferramenta WEKA, sendo possível realizar todos os procedimentos diretamente pela interface da extensão.

3.3 CONSIDERAÇÕES SOBRE O CAPÍTULO

Neste capítulo foi abordado o funcionamento do método original, demonstrando o seu deslocamento e a sua geração de modelos preditivos. Além disso, o capítulo descreve detalhes sobre as duas variações implementadas e suas diferenças em relação ao método original. Também foi mostrado uma visão geral das variações implementadas e trechos dos códigos desenvolvidos.

O próximo capitulo destina-se a demonstrar os experimentos preditivos realizados na com as novas variações da extensão.

4 EXPERIMENTOS COM AS NOVAS VARIAÇÕES

Com o objetivo de avaliar as variações implementadas na extensão, dois experimentos foram realizados em seis bases de dados temporais, compostas por dados sobre *pull request* de diferentes projetos hospedados no GitHub, nenhum motivo técnico levou a escolha de bases sobre *pull request*, apenas este foi o contexto das bases utilizadas na tese de Lima Júnior (2017), tornando assim uma escolha mais prática e de fácil acesso.

O capítulo está organizado da seguinte forma, a seção 4.1 descreve brevemente as bases de dados. A seção 4.2 apresenta os experimentos realizados. A seção 4.3 descreve e discute os resultados dos experimentos no método por intervalo de datas. A seção 4.4 trata da avaliação do método por número absoluto de instâncias e uma conclusão sobre os experimentos é apresentada na seção 4.5.

4.1 BASES DE DADOS UTILIZADAS

Nesta etapa da pesquisa procurou-se bases de dados que apresentassem a característica temporal. Lima Júnior (2017) categorizou diversas bases sobre *pull requests,* colhidas com o uso da ferramenta GHTorrent e classificadas em 3 grupos diferentes. Os grupos se baseiam nas características das equipes de cada projeto, utilizando para isso o número de *pull requests* que foram integrados em relação ao número de desenvolvedores da

equipe que realizaram as integrações. Para a realização dos experimentos preditivos propostos nesta pesquisa, foram selecionados dois projetos de cada grupo utilizado na tese de Lima Júnior (2017), tais projetos são apresentados na Quadro 1.

Quadro 1 - Visão geral das bases de dados.

Projeto	**Número de *pull requests***	**Descrição**
titanium_mobile	6.044	Aplicativo que usa a linguagem JavaScript como padrão para o desenvolvimento mobile, permitindo desenvolver para qualquer plataforma mobile com apenas uma linguagem.
katello	1.066	Software livre que ajuda os administradores de sistemas a gerenciar servidores durante todo o seu ciclo de vida.
akka	1.021	Conjunto de bibliotecas de código aberto para projetar sistemas resilientes e escalonáveis que abrangem núcleos de processador e redes.
infinispan	3.114	Software de armazenamento de dados NoSQL de cache distribuído, que pode ser usado como serviço remoto através do TCP/IP.
bitcoin	2.934	Moeda descentralizada que permite transações financeiras sem intermediários, mas, verificadas por todos usuários (nodos) da rede, que são gravadas em um banco de dados distribuídos, chamado de blockchain.
netty	907	O Netty é uma estrutura de servidor cliente de E/S sem bloqueio para o desenvolvimento de aplicativos de rede Java, como servidores de protocolo e clientes.

Fonte: Elaboração própria.

Todas as bases utilizadas nesta pesquisa possuem instâncias que representam cada uma um *pull request* realizado no projeto, cada instância contém 24 atributos principais, além de alguns outros atributos secundários gerados através da aplicação de filtros e pré-processamento realizado na tese de Lima Júnior (2017). Um atributo importante para os experimentos desta pesquisa é *closedDate,* originalmente este atributo foi removido no pré-processamento feito por Lima Júnior (2017). Por se tratar do atributo central para o funcionamento da variação por intervalo de datas, foi realizado um novo processamento

manual em cada uma das bases, onde o atributo foi colhido nas bases originais extraídas pelo GHTorrent e adicionado novamente as bases de Lima Júnior (2017). Os 24 atributos estão listados no Quadro 2, listando o nome e descrevendo a característica.

Quadro 2 - Descrição dos atributos.

Atributo	**Descrição**
acceptanceRepo	Taxa de aceitação de *pull request* no repositório.
followers	Indica se o solicitante é seguido pelo integrador.
following	Indica se o integrador é seguido pelo solicitante.
ageUser	Tempo do solicitante como usuário do GitHub
totalPullDeveloper	Quantidade total de *pulls* realizados.
mergerdPullUser	Quantidade total de *pulls* que foram aceitos.
closedPullUser	Quantidade total de *pulls* que foram recusados.
rejectUser	Quantidade de *pull request* rejeitados.
aceptanceDeveloper	Quantidade de solicitantes aceitos no repositório.
watchRepo	Booleano que indica se o solicitante marcou a opção acompanha o repositório (opção *Watching* do GitHub).
location	Indica o local do planeta de origem da solicitação de *pull request.*
login	Indica o nome de usuário do solicitante no GitHub.
closedDate	Indica a data completa (dd/mm/yyyy HH:mm:ss) que o *pull request* foi fechado.
commitsPull	Quantidade de *commits* do *pull request.*
commitsbyFilesPull	Número de *commits* alterado nos arquivos do *pull request.*
authorMoreCommits	Indica os integradores com maior número de *commits* dos arquivos alterados pelo *pull request* nos últimos 7 dias.
additionsLines	Quantidade de linhas de código adicionadas pelo *pull request.*
deletionsLines	Quantidade de linhas de código removidas pelo *pull request.*
totalLines	Quantidade total de linhas de código alteradas do *pull request.*
changedFiles	Quantidade de arquivos alterados pelo *pull request.*
typeDeveloper	Indica se o solicitante é interno da equipe ou externo.
requesterFlowCoreTeam	Indica se o solicitante segue a equipe principal.
coreTeamFlowRequester	Indica se a equipe principal segue o solicitante.
closedBy	Indica qual membro da equipe fechou o *pull request.*

Fonte: Elaboração própria.

Nos experimentos, é importante destacar as características de dois atributos, *ClosedBy* que é utilizado para predizer os desenvolvedores mais indicados para integrar um *pull request*, sendo ele o atributo preditivo utilizado nos experimentos, buscando identificar qual o desenvolvedor mais recomendado para fechar um *pull request* e *ClosedDate,* utilizado no método por intervalo de datas para identificar todos os registros de datas da base de dados,

calculando o tamanho dos conjuntos *t* e *k*, direcionando os ciclos de criação de modelos preditivos do método.

4.2 EXPERIMENTOS

Para demonstrar as modificações realizadas na ferramenta e alcançar o objetivo desta pesquisa, dois tipos de experimentos foram realizados. No primeiro experimento, observou-se a **variação de tamanho dos modelos preditivos** gerados, utilizando um algoritmo e diferentes parâmetros para compor as janelas de treino e teste. No segundo experimento, foi observado o **impacto na acurácia**, utilizando cinco algoritmos diferentes e o mesmo parâmetro para compor as janelas.

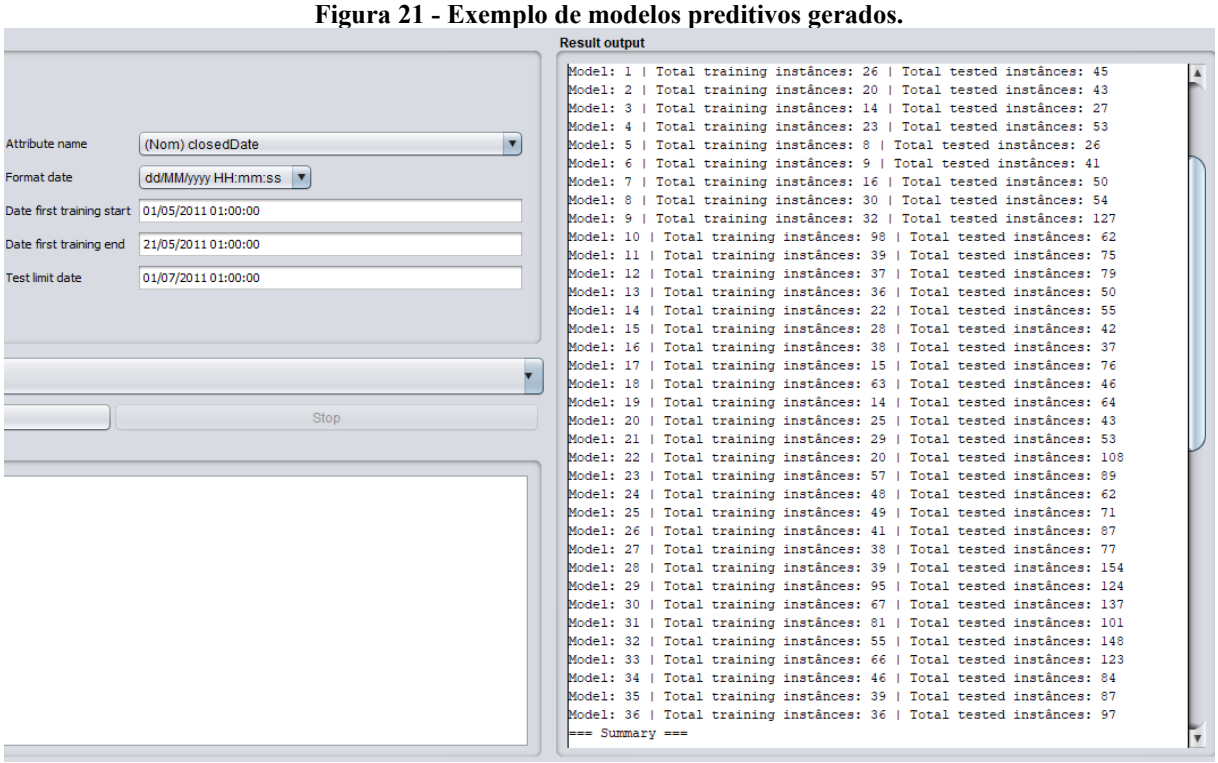
Para o primeiro experimento, foi escolhido o algoritmo *randomForest*. Durante o experimento foram analisadas as variações dos modelos preditivos gerados, observando os tamanhos dos conjuntos *t* e *k* ao longo da execução do método e somente o método por intervalo de datas será avaliado no primeiro experimento. Pelo fato do método por número absoluto de instâncias possuir janelas com valores fixos, quando não está acumulando o conjunto *t*, o método sempre gerará os mesmos valores nos conjuntos *t* e *k,* em todos os modelos preditivos e quando o método está acumulando o conjunto *t*, o tamanho do conjunto é previsível seguindo um padrão em todos os modelos, onde o tamanho do conjunto *t* no próximo modelo será igual ao conjunto *t* atual somado com o conjunto k atual.

No segundo experimento foram selecionados 5 algoritmos diferentes, cada algoritmo representa uma abordagem técnica de *classifiers* da ferramenta WEKA, sendo *ramdomForest* para árvores de decisão, *Naive Bayes* representando o teorema de bayes, SMO para máquina de vetores de suporte, *decisionTable* para regras, e IBK como representante do *k*-NN. Os dois métodos por intervalo entre datas e por número absoluto de instâncias foram utilizados e para ambos foram demonstrados os resultados com e sem acumulação. Estes experimentos e seus resultados serão demonstrados nas seções 4.3 e 4.4 a seguir.

4.3 VARIAÇÃO DOS MODELOS PREDITIVOS

Ao fim da execução, o método por intervalo de datas imprime em um painel (*result output*) na lateral direita da interface, o número de instâncias que compõem os conjuntos *t* e *k* de cada modelo preditivo, como mostra a Figura 21.

Figura 21 - Exemplo de modelos preditivos gerados.



Fonte: Elaboração própria.

É importante que o usuário busque gerar modelos preditivos com a menor variação possível no tamanho dos seus conjuntos *t* e *k*, e que o tamanho dos conjuntos *t* seja em sua maioria, superior ao tamanho dos conjuntos *k* a cada modelo gerado, refletindo assim em uma avaliação mais coerente com o mundo real.

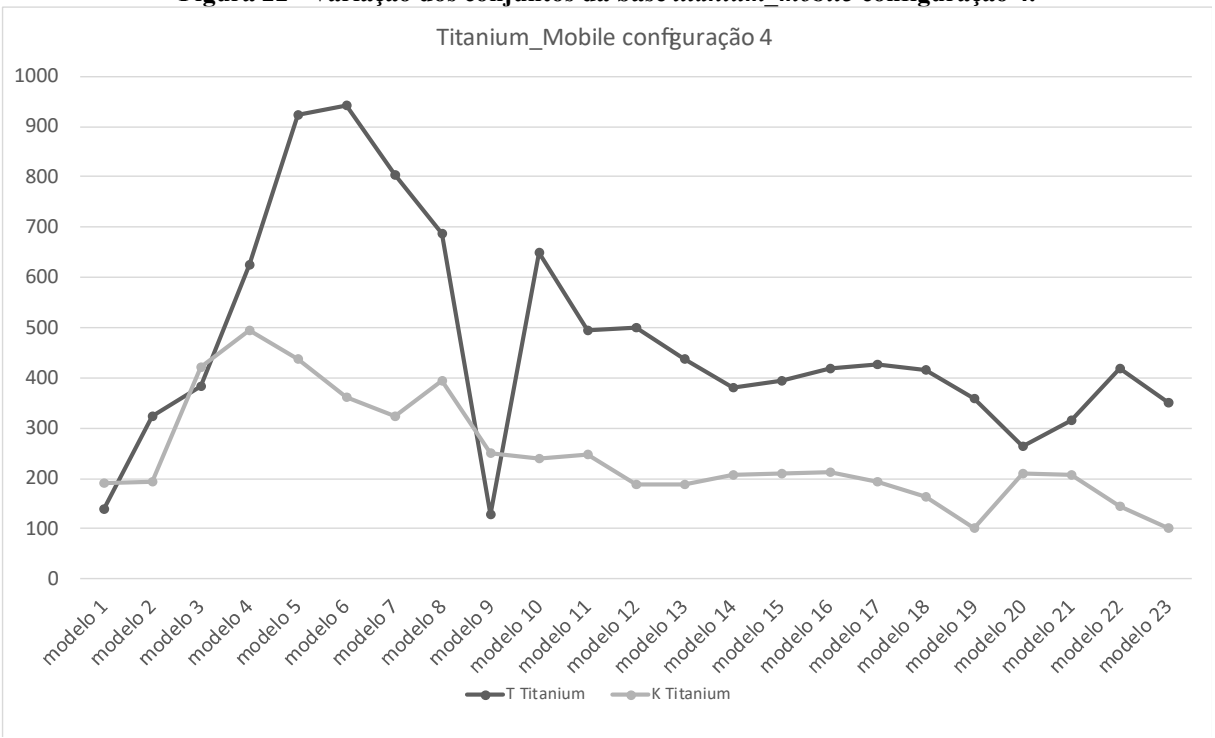
Entretanto, como o método possui uma característica dinâmica na geração dos modelos, é impossível garantir que todas as bases de dados possuirão conjuntos com um tamanho igual, ou mesmo aproximado, em todos os modelos gerados, sendo necessário um bom conhecimento da base utilizada e uma configuração cuidadosa das suas janelas, buscando um equilíbrio entre os objetivos do usuário e a menor variação possível dos modelos preditivos. Este experimento buscou então demonstrar uma situação neste contexto, e as possíveis opções a serem utilizadas para configurar as janelas.

No experimento, as variações dos modelos preditivos foram observadas a partir do primeiro *pull request* integrado no projeto, utilizando parâmetros de entrada para a configuração do tamanho das janelas do modelo de avaliação. Para comparar o tamanho dos conjuntos em cada modelo preditivo, utilizou-se a medida estatística coeficiente de variação

(CV). Como o CV é uma medida relativa de variabilidade, sua utilização torna possível comparar em porcentagens a variação nos conjuntos entre cada uma das bases. Para o contexto das bases coletadas foram definidos três intervalos para avaliar o CV: 0 – 20% indica uma boa variação, não ocorrendo muita dispersão no número instâncias em cada conjunto, 21-40% indica uma dispersão mediana e acima de 40% indica uma alta dispersão.

Além da variação, também é necessário comentar sobre o tamanho dos conjuntos *t*, em relação aos conjuntos *k* em cada modelo preditivo. A motivação para isso pode ser melhor visualizada através na Figura 22, a linha preta demonstra o tamanho dos conjuntos *t,* e a linha cinza o tamanho dos conjuntos *k*.

Figura 22 - Variação dos conjuntos da base *titanium_mobile* configuração 4.



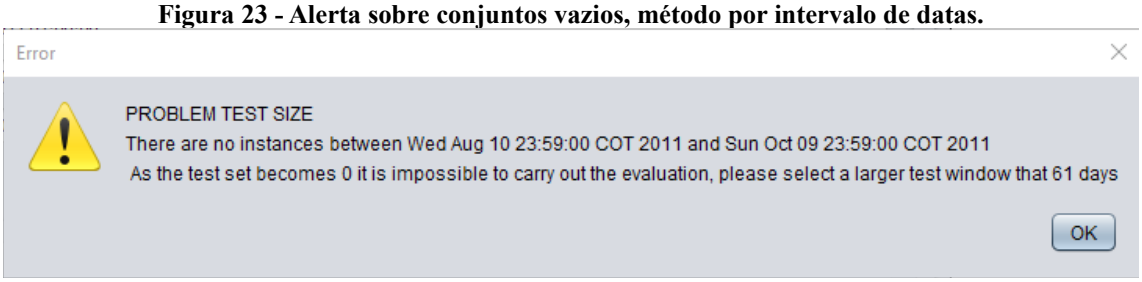
Fonte: Elaboração própria.

A Figura 22 demonstra as variações por modelo da base *titanium_mobile*, onde é possível perceber mais claramente a influência da característica dinâmica de geração dos conjuntos, sobre o tamanho dos próprios conjuntos gerados no método. Por exemplo, mesmo com uma janela de treino configurada com o dobro de dias da janela de teste, é possível notar que os modelos preditivos 1, 3 e 9 gerados, possuem conjuntos *t* com um número menor de instâncias que os conjuntos *k*, tornando estes modelos ruins, já que com um número menor de instâncias para o treinamento, pode ser que o algoritmo erre mais.

Inicialmente o parâmetro de composição das janelas foi definido em 7 dias para *k* e 15 dias para *t*, pois representam semanticamente uma semana e duas semanas, o que torna a

escolha intuitiva e interessante para uma aplicação real, onde um gerente de projeto pode prever quais desenvolvedores podem integrar os *pull requests* nos próximos dias. Com a aplicação do primeiro parâmetro notou-se um comportamento anormal nas bases de dados, para alguns modelos, os conjuntos *t* e/ou *k* não possuem instâncias. Como o tamanho dos conjuntos é gerado com base nos intervalos entre as datas inseridas nas janelas, dependendo da base de dados pode existir modelos onde nenhuma instância foi encontrada na base, caso o tamanho em dias, definido para as janelas seja menor que uma faixa de tempo vazia na base, nenhuma instância poderá ser capturada. Como não é possível treinar ou testar com nenhuma instância, o método para, informando ao usuário o local da base onde ocorreu o problema e que conjunto precisa de um tamanho maior. A Figura 23 mostra um exemplo deste aviso.

Figura 23 - Alerta sobre conjuntos vazios, método por intervalo de datas.



Fonte: Elaboração própria.

Os resultados para cada base podem ser vistos no Quadro 3, demonstrando o nome da base, número de modelos gerados e uma descrição sobre o problema.

Quadro 3 - Experimento por intervalo de datas t = 15 dias k = 7 dias.

Nome da base	Descrição
titanium_mobile	Não existem instâncias com datas iguais ou após 14/04/2011 e iguais ou anteriores a 20/04/2011, gerando um conjunto de teste vazio no modelo 2.
katello	Não existem instâncias com datas iguais ou após 22/12/2012 e iguais ou anteriores a 28/12/2012, gerando um conjunto de teste vazio no modelo 35.
akka	Não existem instâncias com datas iguais ou após 02/05/2011 e iguais ou anteriores a 08/05/2011, gerando um conjunto de teste vazio no modelo 2.
infinispan	Não existem instâncias com datas iguais ou após 24/08/2011 e iguais ou anteriores a 30/08/2011, gerando um conjunto de teste vazio no modelo 21.
bitcoin	Não existem instâncias com datas iguais ou após 22/04/2011 e iguais ou anteriores a 28/04/2011, gerando um conjunto de teste vazio no modelo 4.
netty	Não existem instâncias com datas iguais ou após 17/05/2011 e iguais ou anteriores a 23/05/2011, gerando um conjunto de teste vazio no modelo 3.

Fonte: Elaboração própria.

Todas as bases geraram conjuntos de teste sem instâncias em algum momento da execução do método, impossibilitando sua avaliação completa com o parâmetro escolhido, demonstrando que, pelo menos, no contexto destas bases, é preciso utilizar janelas maiores.

Desta forma, novos parâmetros foram configurados, de acordo com o seguinte padrão: dobrar o valor para a janela de treino e teste, até alcançar um parâmetro onde todas as bases possam ser avaliadas, desta forma foram geradas as configurações 2 (treino = 30 dias/teste = 15 dias), 3 (treino = 60 dias/teste = 30 dias), 4 (treino = 120 dias/teste = 60 dias) e 5 (treino = 240 dias e teste = 120 dias). Estas configurações e seus resultados estão sumarizados na Tabela 1, onde "Nº de modelos" indica a quantidade de modelos gerados ao fim da execução do método e "Modelo não gerado" indica o modelo onde o intervalo de data definido para treino ou teste não pode capturar nenhuma instância, pois nenhuma instância na base possui registro de data referente a esse intervalo, desta forma o método é parado.

Tabela 1 – Resultados para as configurações de avaliação usando intervalo de datas.

	Configuração 2		Configuração 3		Configuração 4		Configuração 5	
Nome da Base	**Nº de modelos**	**Modelo não gerado**	**Nº de modelos**	**Modelo não gerado**	**Nº de modelos**	**Modelo não gerado**	**Nº de modelos**	**Modelo não gerado**
Titanium mobile	103		50		23		9	
katello		30	28		11		6	
akka		1		2	22		9	
infinispan	103		50		23		10	
bitcoin	103		50		23		10	
netty		2		1	23		10	

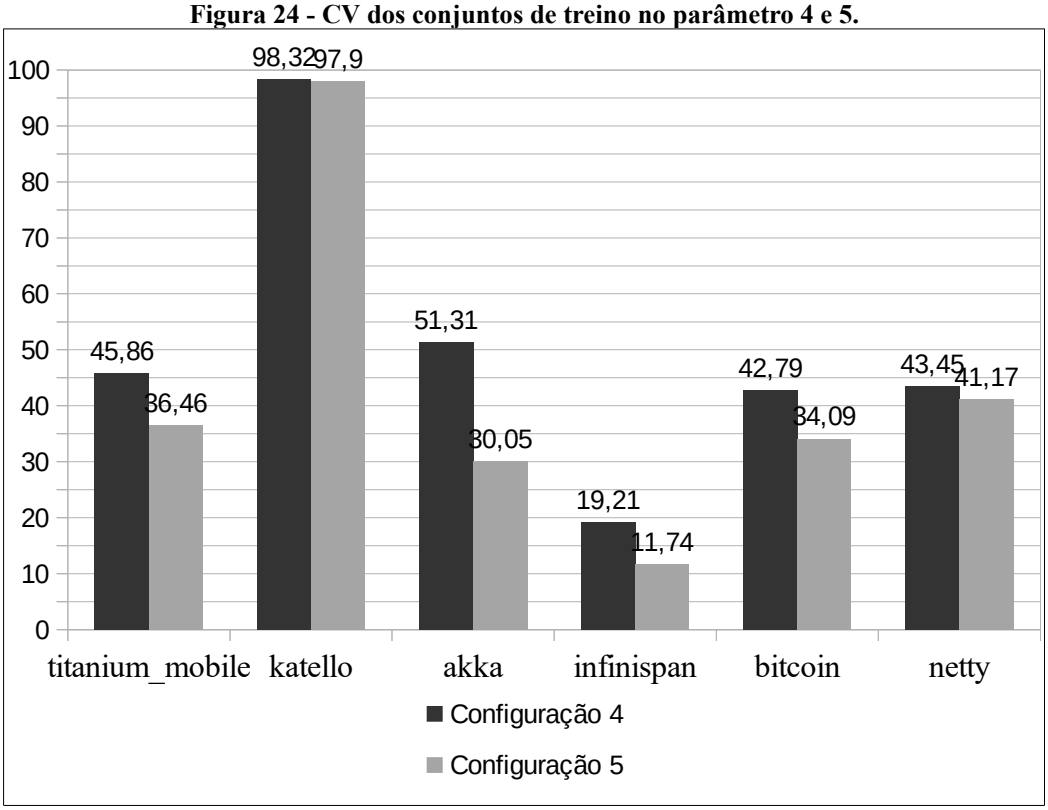
Fonte: Elaboração própria.

A configuração 2 não capturou instâncias em 3 bases de dados: *katello*, *akka* e *netty*, a configuração 3 conseguiu capturar instâncias em todos os modelos na base *katello*, entretanto não capturou nas bases *akka* e *bitcoin*. Como a partir da configuração 4 todas as bases foram avaliadas sem a ocorrência de nenhum modelo incapaz de capturar instâncias, tanto a configuração 4, como a configuração 5 foram selecionadas como parâmetros para avaliar a variação dos conjuntos *t* e *k* em cada uma das bases. Os demais gráficos para cada base podem ser consultados no "APÊNDICE B GRÁFICOS DOS MODELOS".

Os tamanhos dos conjuntos em cada modelo preditivo, foram colhidos e agrupados em uma planilha, com isso, o CV foi aplicado sobre os dados. Com o intuito de facilitar a visualização dos dados dois gráficos foram criados, apresentando o CV dos conjuntos de treino e teste respectivamente. No eixo dos Y estão as porcentagens e no eixo dos X estão os nomes de cada base de dados utilizada. Cada base possui duas barras verticais, uma preta e uma cinza, representando a configuração 4 e 5 respectivamente, na Figura 24, o CV dos

conjuntos de treino será apresentado.

Figura 24 - CV dos conjuntos de treino no parâmetro 4 e 5.



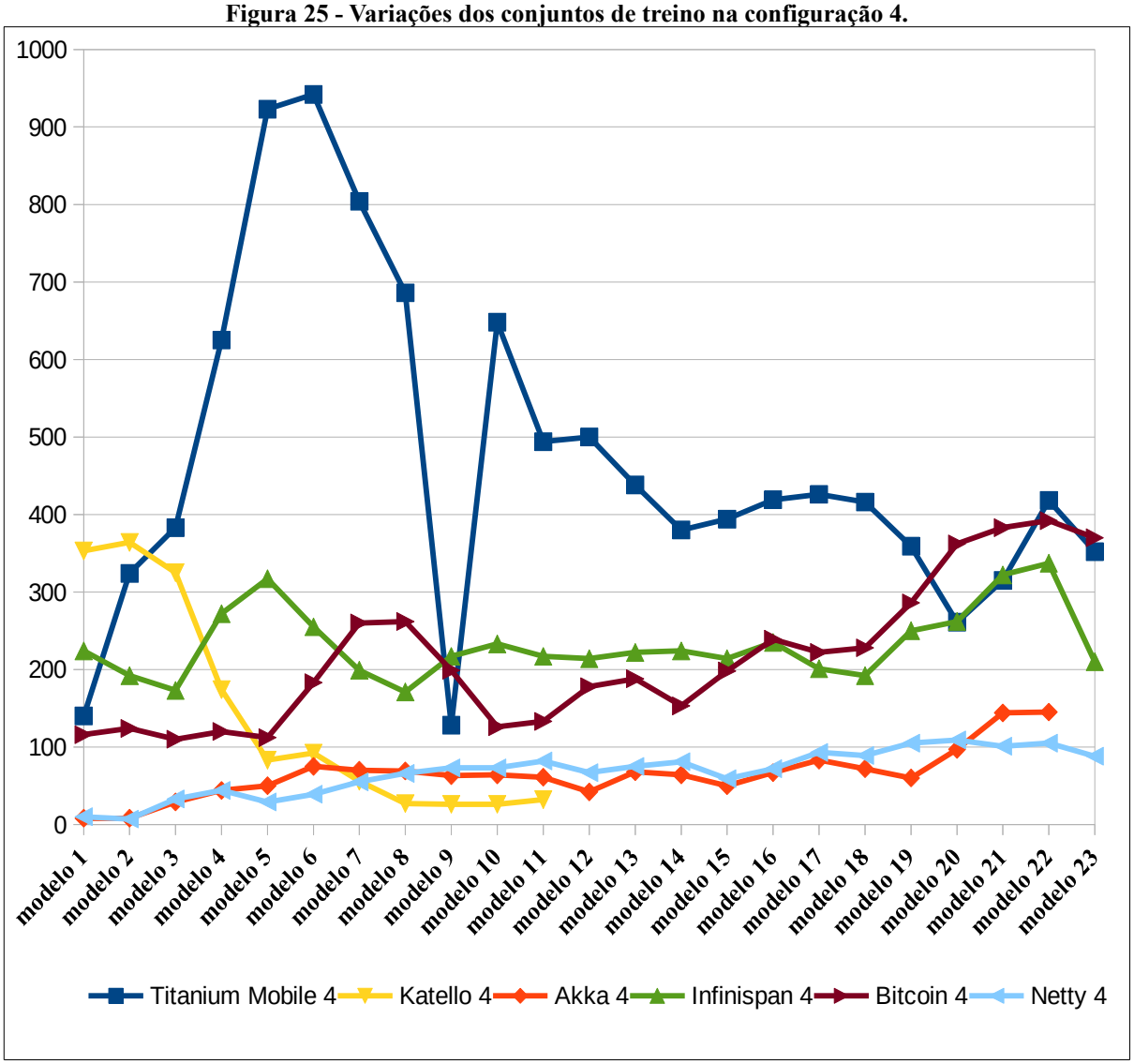
Fonte: Elaboração própria.

Percebe-se que a configuração 4 gera uma alta dispersão nos conjuntos *t* para 5 das 6 bases de dados avaliadas, tendo apenas *infinispan* gerado um CV de baixa dispersão. Na configuração 5 percebe-se u

ma melhoria, as bases avaliadas obtiveram uma redução média de aproximadamente 8,25% no CV, sendo que *titanium_mobile*, *akka* e *bitcoin* entraram na zona de media dispersão, tendo somente *netty* e *katello* continuado com altas dispersões.

Uma forma de compreender melhor a dispersão dentro das bases é analisar um gráfico de linhas, demonstrando a variação de uma forma “visual”, acompanhando o tamanho dos conjuntos em cada base de dados. A Figura 25 demonstra um exemplo desse gráfico, aplicando aos modelos preditivos do conjunto *k* da configuração 4, o eixo dos X possui a numeração dos modelos e o eixo dos Y possui o número de instâncias.

Figura 25 - Variações dos conjuntos de treino na configuração 4.



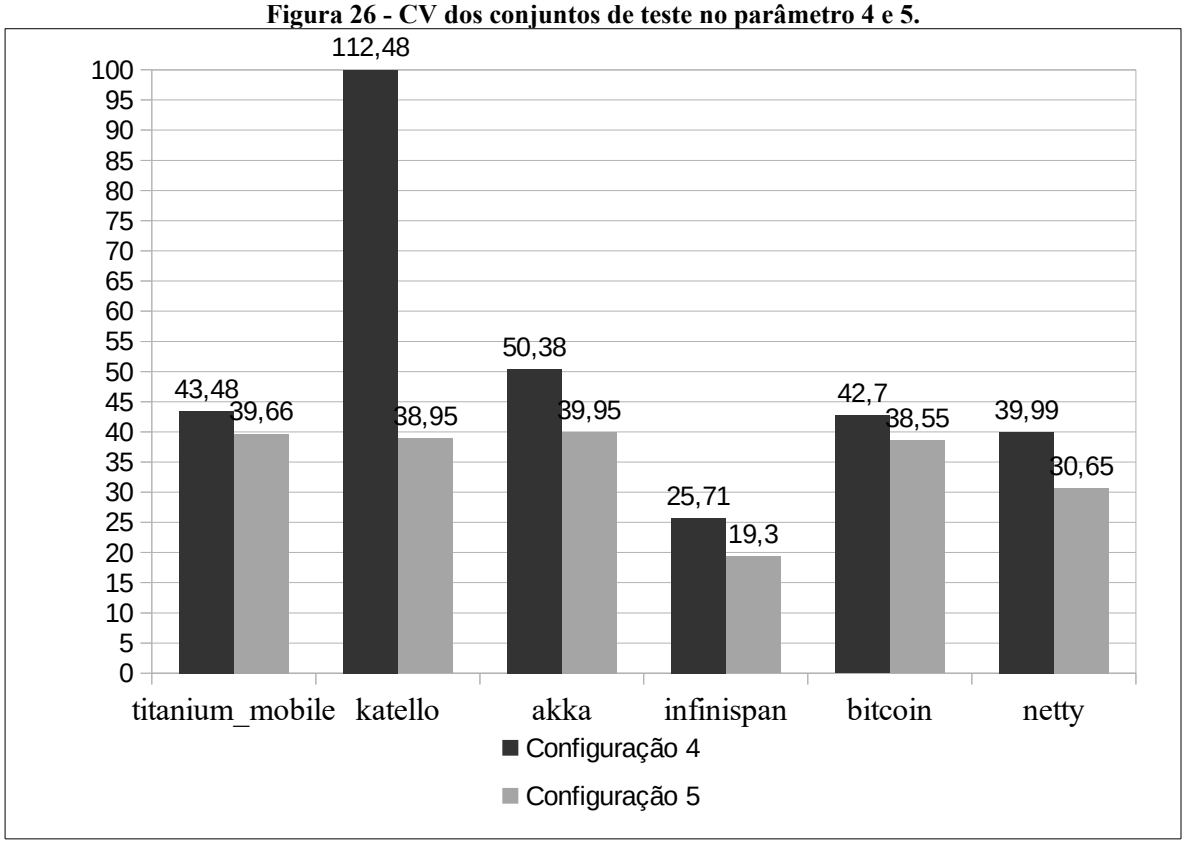
Fonte: Elaboração própria.

O gráfico da Figura 24 foi produzido utilizando os valores obtidos no sumário do método, permitindo assim, observar com mais detalhes a variação dos modelos. Percebe-se que *titanium_mobile* possui uma característica de rápido crescimento inicial seguido de queda, obtendo um aumento rápido e constante no número de integrações durante os 6 modelos inicias, levando em conta que o modelo inicial de treinamento representa semanticamente 4 meses e que cada modelo subsequente avança 2 meses dentro da base, é possível deduzir que o crescimento ocorre no primeiro ano do projeto (modelo 1 a 6), sofrendo uma pequena queda nos 4 meses seguintes (modelo 7 e 8), seguindo para uma queda brusca ao longo de 2 meses no modelo 9 e se recuperando nos dois meses seguintes (modelo 10), mantendo um ritmo de queda seguidos de momentos de estabilidade no decorrer da base.

Cada base de dados temporais possui uma característica diferente na distribuição das instâncias, analisar e entender suas características, torna possível descobrir uma melhor estratégia para a configuração das janelas e assim, diminuir a dispersão dos modelos preditivos gerados. *Katello* por exemplo, apresenta o pior CV dentre as bases avaliadas, observando os seus modelos nota-se na maioria uma tendência de queda contínua, onde primeiro modelo possui mais de 200 instâncias e a cada novo modelo esse número diminui, chegando a cerca de pouco mais que 20 instâncias no modelo final, tornando difícil configurar janelas que gerem uma boa dispersão, já que a base apresenta somente dois intervalos de estabilidade, um nos modelos 1 e 2 e outros durante os modelos 8 a 11, entretanto, existe um número de instâncias muito diferente entre cada um dos intervalos e a tendência de queda nos demais modelos impedem que a base consiga uma menor variação.

Para bases que apresentam essa característica, com poucos ou nenhum momento de estabilidade, dependendo dos objetivos do usuário responsável pela mineração, seria recomendável o uso dos métodos por número absoluto de instâncias ou porcentagem. A Figura 26 apresenta o gráfico do CV obtido com os conjuntos de teste.

Figura 26 - CV dos conjuntos de teste no parâmetro 4 e 5.



Fonte: Elaboração própria.

Os conjuntos de teste *Akka*, *titanium_mobile*, *bitcoin* e *netty*, obtiveram melhores CV

na configuração 4 em relação aos conjuntos de treino na mesma configuração, com *katello* e *infinispan* seguindo o caminho inverso, obtendo um CV mais elevado que seus conjuntos de treino na mesma configuração.

Com a aplicação da configuração 5, melhores dispersões foram obtidas, com exceção de *infinispan*, todas as bases de dados adentraram a zona de media dispersão, mesmo *katello* que possui uma tendência ruim na dispersão dos modelos obteve uma redução de 281,55% no seu CV e *infinispan* retornou a zona de baixa dispersão.

Com a aplicação do experimento nota-se uma tendência nos conjuntos *t* e *k,* indicando a obtenção de melhores dispersões nos modelos preditivos das bases avaliadas, a medida que as janelas tornam-se maiores, algo que o usuário responsável pela mineração poderia explorar mais a fundo, gerando novas configurações em busca do parâmetro ideal a ser utilizado. O experimento também demonstrou a dependência do método quanto à configuração das janelas *t* e *k*.

4.4 IMPACTO NA ACURÁCIA

Este experimento busca identificar dentre 5 algoritmos, aquele que produz as melhores acurácias, utilizando *closedBy* como atributo alvo e buscando identificar o desenvolvedor mais recomendado para fechar um *pull request*. Tanto o método por intervalo de datas como o método por número absoluto de instâncias foram utilizados neste experimento. Além disso, o comportamento das bases ao acumular e não acumular os conjuntos de treino foi observado. Para isso, foram analisadas as médias obtidas por algoritmo e o algoritmo que gerou as melhores acurácias individuais (observando a acurácia em cada base).

A configuração 4 foi selecionada para compor as janelas *Jt* e *Jk* do método por intervalo de datas, representando aproximadamente 4 meses para *Jt* e 2 meses para *Jk,* pois no experimento anterior não gerou conjuntos vazios. Quanto ao método por número absoluto de instâncias, foi selecionado 100 instâncias para a janela de treino e 50 instâncias para a janela de testes, mantendo os tamanhos utilizados nos exemplos do capítulo 3. Tais experimentos, serão descritos nas seções 4.4.1 e 4.4.2 a seguir.

4.4.1 Impacto na acurácia - método por intervalo de datas

A primeira etapa do experimento analisou os resultados obtidos com o método por intervalo de datas na configuração 4, dispondo as acurácias em tabelas, o cabeçalho apresenta o nome de cada base e a coluna na ponta esquerda apresenta o nome de cada algoritmo utilizado. A Tabela 2 demonstra os resultados obtidos ao acumular os conjuntos de treino.

Tabela 2 - Resultados obtidos com intervalo de datas acumulando o conjunto de treino.

Projetos	***randomForest***	**naiveBayes**	**SMO**	**decisionTable**	**IBK**
Titanium mobile	41,68%	28,14%	**57,12%**	34,12%	33,67%
Katello	**27,47%**	24,35%	23,80%	18,49%	20,32%
Akka	**49,45%**	42,68%	48,00%	46,91%	40,74%
Infinispan	34,24%	21,25%	35,42%	**42,14%**	24,05%
Bitcoin	**66,80%**	59,34%	63,50%	63,96%	55,80%
Netty	**65,74%**	58,08%	57,96%	63,59%	59,52%
Média	47,56%	38,97%	**47,63%**	44,86%	39,02%
Número de vitórias	**4**	0	1	1	0

Fonte: Elaboração própria.

Com a acumulação SMO obteve o melhor resultado geral, alcançando a melhor acurácia média dentre os algoritmos experimentados (47,63%), estando extremamente próximo ao *randomForest* que obteve 47,56%, uma diferença de apenas 0,07%, enquanto a pior média foi de 38,97%, obtida pelo algoritmo naiveBayes, estando também extremamente próximo a IBK que obteve 39,02%, uma diferença de apenas 0,05%.

Mesmo SMO tendo obtido o melhor resultado geral, nota-se que *randomForest* obteve as melhores acurácias individuais, alcançando 4 vitórias nas bases: *netty*, *bitcoin*, *akka* e *katello*. Um comportamento interessante é percebido ao observar a acurácia obtida em *katello* e *infinispan,* que geraram as piores acurácias até o momento, ao mesmo tempo que obtiveram a melhor dispersão (*infinispan)* e pior dispersão (*katello).* No entanto, nesse experimento não é possível afirmar que exista uma relação direta entre a acurácia e a variação dos modelos, essa investigação foge do escopo desta pesquisa e pode ser investigada por trabalhos futuros.

Em todo caso, a motivação para a busca por uma boa variação contínua sendo importante, com sua principal motivação estando relacionada ao princípio fundamental que

levou a criação da janela deslizante, **“buscar uma avaliação mais coerente com o mundo real para bases de dados temporais”**. A Tabela 3 demonstra os resultados obtidos ao não acumular os conjuntos *t*.

Tabela 3 - Resultados obtidos com intervalo de datas não acumulando o conjunto de treino.

Projetos	***randomForest***	**naiveBayes**	**SMO**	**decisionTable**	**IBK**
Titanium mobile	**39,74%**	30,25%	31,81%	30,76%	33,09%
Katello	**26,19%**	25,27%	23,07%	20,14%	19,04%
Akka	**50,06%**	44,61%	41,11%	44,25%	39,90%
Infinispan	31,62%	22,61%	27,60%	**41,58%**	24,20%
Bitcoin	**65,88%**	52,46%	55,80%	65,03%	60,00%
Netty	**64,67%**	62,39%	60,59%	57,36%	58,32%
Média	**46,36%**	39,60%	34,74%	43,19%	35,92%
Número de vitórias	**5**	0	0	1	0

Fonte: Elaboração própria.

Ao não acumular os conjuntos de treino, a média de acurácia reduz em 80% dos algoritmos, com redução de 1,20% no *randomForest*, 12,89% em SMO, 1,67% em *decisionTable* e 3,10% em IBK, e aumenta em 0,63% no algoritmo naiveBayes.

Dentre os algoritmos testados *randomForest* obteve a melhor acurácia geral, com uma acurácia média de 46,36%, enquanto a pior média foi obtida por SMO com 34,74%. Individualmente, *randomForest* também obteve as melhores acurácias em 5 bases: *netty*, *bitcoin*, *akka, katello* e *titanium_Mobile*, 1 a mais em relação ao experimento anterior.

Percebe-se então, que, pelo menos, no contexto das bases testadas, com a configuração 4 acumular os conjuntos de treino não foi capaz de melhorar tanto o resultado final na maioria das bases, com apenas SMO obtendo um aumento relevante na sua média geral (12,89%), e ao mesmo tempo, não gerou reduções significativas na única base onde a acurácia média caiu (*naiveBayes*), obtendo uma redução extremamente baixa, além disso, *randomForest* foi o melhor algoritmo para o método por intervalo de datas, alcançando a melhor acurácia no maior número de bases, e no único caso onde perdeu na média geral, a diferença foi muito pequena, apenas 0,07%.

4.4.2 Impacto na acurácia - método por número absoluto de instâncias

Na segunda etapa do experimento, o método por número absoluto de instâncias foi utilizado e as acurácias foram obtidas com cada algoritmo ao acumular e não acumular as janelas de treino. A Tabela 4 demonstra o resultado obtido ao acumular os conjuntos de treino.

Tabela 4 - Resultados obtidos com número absoluto de instâncias acumulando o conjunto de treino.

Projetos	*randomForest*	naiveBayes	SMO	decisionTable	IBK
Titanium mobile	**46,96%**	33,28%	44,93%	43,69%	39,30%
Katello	**27,57%**	23,59%	23,47%	24,42%	22,52%
Akka	**48,66%**	40,11%	46,33%	44,22%	38,55%
Infinispan	36,66%	21,00%	39,63%	**45,53%**	25,06%
Bitcoin	**70,96%**	61,14%	67,42%	66,53%	58,78%
Netty	**70,00%**	57,87%	57,12%	63,75%	61,87%
Média	**50,14%**	39,50%	39,00%	48,02%	37,26%
Número de vitórias	**5**	0	0	1	0

Fonte: Elaboração própria.

Ao usar acumulação, *randomForest* obteve o melhor resultado geral, alcançando uma acurácia média de 50,14%, enquanto *naiveBayes* obteve a pior acurácia média, com 39,50%. Individualmente *randomForest* também gerou as melhores acurácias em 5 bases: *netty*, *bitcoin*, *akka, katello* e *titanium_Mobile*.

Nota-se uma tendência em *infinispan*, onde *decisionTable* sempre gera as melhores acurácias, enquanto as demais bases geram melhores acurácias com o uso de *randomForest*, com exceção de *titanium_Mobile* no primeiro experimento com acumulação. A Tabela 5 demonstra os resultados obtidos sem acumulação.

Tabela 5 - Resultados obtidos com número absoluto de instâncias não acumulando o conjunto de treino.

Projetos	*randomForest*	naiveBayes	SMO	decisionTable	IBK
Titanium mobile	39,10%	33,71%	33,94%	**39,32%**	33,81%
Katello	**26,94%**	21,68%	20,21%	24,00%	21,68%
Akka	**48,77%**	40,88%	42,55%	40,55%	41,55%
Infinispan	32,40%	25,46%	27,56%	**34,43%**	24,66%
Bitcoin	**70,39%**	62,37%	60,75%	63,62%	59,62%
Netty	**71,12%**	62,37%	60,75%	63,62%	59,62%
Média	**41,60%**	41,08%	40,96%	37,70%	36,54%
Número de vitórias	**4**	0	0	2	0

Fonte: Elaboração própria.

Sem acumular os conjuntos de treino, a acurácia média cai em 60% das bases de dados testadas, sendo 8,53% em *randomForest*, 10,32% em *decisionTable* e 0,72% em IBK, enquanto sobe nas demais bases, em 1,58% no *naiveBayes* e 1,97% em SMO. Observando o resultado para ambos os métodos, nota-se uma tendência de redução na acurácia para os algoritmos decisionTable, randomForest e IBK, quando não são acumulados os conjuntos de treino, além de uma tendência de crescimento na acurácia para naiveBayes.

Novamente, dentre os algoritmos testados no método por número absoluto de instâncias, *randomForest* se destacou obtendo uma acurácia média de 41,60%, enquanto a pior média foi obtida por IBK com 36,54%. Individualmente, *randomForest* também se destacou, mas, diferente da acumulação, obteve a melhor acurácia em apenas 4 das 6 bases, sendo elas *netty*, *bitcoin*, *akka* e *katello*, enquanto *decisionTable* obteve as melhores acurácias nas 2 bases restantes, *infinispan* e *titanium_mobile*.

Com o fim dos experimentos, nota-se que o melhor algoritmo foi *randomForest*, gerando as melhores acurácias gerais e individuais em quase todos os casos, tanto para instâncias como para datas, com exceção do método por intervalo de datas ao acumular, onde obteve uma média um pouco inferior a SMO, desta forma *randomForest* é o algoritmo indicado para uso na extensão, em bases de dados que apresentem um contexto próximo aos das bases escolhidas para esta pesquisa. Entretanto, é preciso destacar que *infinispan* apresentou um comportamento que foge do padrão das demais, onde *decidisonTable* gerou as melhores acurácias em todos os experimentos realizados.

4.5 CONSIDERAÇÕES SOBRE O CAPÍTULO

Neste capítulo foram detalhados os experimentos realizados com as variações implementadas na extensão TTSV. Para isso foram demonstradas as bases de dados utilizadas e seus principais atributos. Em seguida, os experimentos foram realizados e seus resultados foram demonstrados com o uso de gráficos e tabelas, por fim algumas análises foram feitas quanto aos resultados obtidos.

O próximo capítulo finaliza essa pesquisa, apresentando as considerações finais e recomendações para futuros trabalhos.

5 CONSIDERAÇÕES FINAIS E RECOMENDAÇÕES

Esta pesquisa expandiu o método de avaliação criado por Lima Júnior (2017) aplicado a extensão criada por Lima (2017) para a ferramenta WEKA, adicionando novos métodos a mesma e trazendo maior flexibilidade na sua utilização. O documento de requisitos sobre as modificações realizadas na ferramenta, está disponível no Apêndice A desta pesquisa.

Este capítulo apresenta as considerações finais sobre a pesquisa na seção 5.1 e recomendações para pesquisas futuras na seção 5.2.

5.1 CONSIDERAÇÕES FINAIS

Baseado nos trabalhos de Lima Júnior (2017) e Lima (2017) esta pesquisa propôs desenvolver e analisar variações para a extensão *Training Test Sliding Validation*, que pudessem torná-la útil para um número ainda maior de usuários, com isso um estudo bibliográfico referente a conceitos e técnicas de mineração de dados foi realizado e ao longo desta pesquisa, foram desenvolvidos dois novos métodos para a extensão.

Em termos gerais, poucos problemas foram encontrados durante o desenvolvimento desta pesquisa, dentre eles, vale citar a dificuldade inicial para compreender o código fonte da extensão TTSV original, o que exigiu algum tempo a mais de análise, além disso outra dificuldade foi o longo tempo de pesquisa dedicado ao desenvolvimento da fundamentação

teórica desta pesquisa, algo que foi bastante trabalhoso.

Por fim, esta pesquisa conseguiu cumprir com seu objetivo geral, desenvolvendo com sucesso ambas as variações propostas, criando novas formas de realizar a mineração em bases de dados temporais, tornando mais diversificada as possibilidades de uso da extensão TTSV.

Com as modificações realizadas na extensão, foi possível realizar experimentos preditivos no contexto de *pull request*, abordando cenários diferentes ao gerar conjuntos com base em datas e em número absoluto de instâncias, permitindo assim demonstrar a funcionalidade dos métodos desenvolvidos, sua utilidade e características. Com isso, esta pesquisa deixa como legado **a evolução de um método adequado na avaliação de algoritmos de classificação em bases de dados temporais**, evolução esta que pode e deve ser explorada em futuras pesquisas, com o intuito de evoluir cada vez mais o método da janela deslizante.

5.2 RECOMENDAÇÕES

Com base na pesquisa realizada e nos resultados obtidos é possível sugerir as seguintes recomendações para pesquisas futuras:

a. Implementar a extensão em um sistema web, permitindo realizar avaliações com a extensão.
b. Realizar experimentos com outras bases de dados temporais para verificar a utilidade dos métodos de avaliação dos algoritmos de classificação.
c. Adição de funcionalidades na interface do método por intervalo de datas, permitindo observar as variações modelos através de gráficos e simulações com tamanhos de janelas.
d. Um trabalho de investigação que busque entender e comprovar se existe ou não uma relação entre variação dos modelos preditivos e acurácia gerada, através do experimento com diversas bases de dados.
e. Evoluir os 3 métodos da extensão, permitindo identificar quais os melhores modelos gerados (modelos que tenham impacto positivo na acurácia) e permita

selecioná-los para utilização posterior, ignorando modelos que tenham impacto negativo na acurácia.

REFERÊNCIAS

ANDREONI, Martin. **An intrusion detection and prevention architecture for software defined networking**. Paris, França: Pierre and Marie Curie University, 2014.

ARAÚJO, Marco Antônio Pereira; SPÍNOLA, Eduardo Oliveira. **Engenharia de software magazine, análise de pontos de função.** 2ª Ed. Devmedia, 2009.

ASKLUND, Ulf; DAHLQVIST, Annita Persson; CRNKOVIC, Ivica; HEDIN, Allan; LARSSON, Magnus; RANBY, Johan; SVENSSON, Daniel; **Product Data Management and Software Configuration Management –** Similarities and Differences, The Association of Swedish Engineering Industries. 2001.

BERGMEIR Christoph; HYNDMAN Rob; KOO Bonsoo. **A Note on the Validity of Cross-Validation for Evaluating Autoregressive Time Series Prediction.** Melbourne, Australia: Faculty of Information Technology, 2017.

BERRY, Michael; LINOFF, Gordon. **Data miming techniques:** for marketing, sales and customer support**.** ISBN: 978-0-471-17980-1, USA: Wiley Computer Publishing, 1997.

CAMILO, Cassio Oliveira; SILVA, João Carlos. **Mineração de Dados:** Conceitos, Tarefas, Métodos e Ferramentas. Goiás, Brasil: Instituto de Informática Universidade Federal de Goiás, 2009.

CAVALCANTI, Yguaratã Cerqueira; NETO, Paulo Anselmo da Mota Silveira; MACHADO, Ivan do Carmo; VALE, Tassio Ferreira; ALMEIDA, Eduardo Santana; MEIRA, Silvio Romero de Lemos. **Challenges and Opportunities for Software Change Request Repositories:** A Systematic Mapping Study. Journal of Software: Evolution and Process. 2014.

CHACON, Scott; STRAUB, Ben. **Pro Git.** 2ª Ed. ISBN: 1484200772, Califórnia, USA: Apress, 2014.

CIOS, Krzysztof; SWINIARSKI Roman W; PEDRYCZ Wiltold; KURGAN, Lukasz A. **Data mining:** A Knowledge Discovery Approach. Oklahoma, EUA: 2007.

CLÉSIO, Flavio. **Métricas de avaliação de modelos de classificação/predição.** Disponível em: <https://mineracaodedados.wordpress.com/tag/matriz-de-confusao/>. Acesso em 15 de março de 2018.

CONRADI, Reidar; WESTFECHTEL, Bernhard. **Version models for software configuration management**. Trondheim, Noruega: ACM Computing Surveys (CSUR), v. 30, n. 2, 1998.

CUNHA, Guilherme Carvalho. **Reconhecimento de emoçõoes atraves de imagens em tempo real com o uso de ASM e SVM**. Dissertação (Mestrado), Rio de Janeiro, Brasil: PUC, 2013.

DIAS, André Felipe. **Conceitos Básicos de Controle de Versão de Software — Centralizado e Distribuído.** Disponível em: <https://blog.pronus.io/posts/conceitos-basicos-de-controle-de-versao-de-software-centralizado-e-distribuido/>. Acesso em: 3 de setembro de 2017.

ESTUBLIER, Jacky. **Software Configuration Canagement:** A Roadmap. In: Proceedings of the Conference on The Future of Software Engineering. Limerick, Irlanda: ACM, 2000.

FAYYAD, Usama; PIATETSKY-SHAPIRO, Gregory; SMYTH, Padhraic. **From Data Mining to Knowledge Discovery in Databases.** California, USA: American Association for Artificial Intelligence Menlo Park**,** 1996.

FILHO, Wilson de Paula. **Engenharia de software: fundamentos, métodos e padrões**. 2ª Ed. Rio de Janeiro, Brasil: LTC – Livros Técnicos e Científicos. 2003.

FUKUNAGA, K.; NARENDRA, P. M. **A branch and bound algorithm for computing k-nearest neighbors**. IEEE Transactions on Computers, 1975.

GAMA, João. **Functional Trees**. Porto, Portugal: LIACC, FEP--University of Porto, 2004.

GOUSIOS, G; SPINELLIS, D. GHTorrent: **Github's Data From a Firehose. In:** Proceedings on the 9th IEEE Working Conference on Mining Software Repositories (MSR). Suiça, Europa: IEEE, 2012.

HALL, Mark; WITTEN, Ian; FRANK, Eibe. **The WEKA Data Mining Software:** An Update. Nova zelandia, Oceania: Department of Computer Science, University of Waikato, 2009.

HAN, Jiawei; KAMBER, Micheline; PEI Jian. **DataMining:** Concepts and Techniques. 3ª Ed. ISBN 978-0-12-381479-1. Elsevier, 2011.

HAND, David; MANNILA, Heikki; SMYTH, Padhraic. **Principles of data mining**. ISBN: 026208290x. Londres, Inglaterra; Massachusetts Institute of Technology, The Mit Press, 2001.

IEEE, Std 610.12 - **IEEE Standard Glossary of Software Engineering Terminolog**. Institute of Electrical and Electronics Engineers, 1990.

IEEE, Std 828 - **IEEE Standard for Software Configuration Management Plans**. ISBN: 978-0-7381-4765-9, Institute of Electrical and Electronics Engineers, 2005.

LIMA JÚNIOR, Manoel Limeira. **Previsão de integradores e tempo de vida de *pull requests***. Tese (Doutorado). Rio de Janeiro, Brasil: Universidade Federal Fluminense, 2017.

LESSMANN, Stefan; BAESENS, Bart; MUES, Christopher; PIETSCH, Swantje. **Benchmarking Classification Models for Software Defect Prediction:** A Proposed Framework and Novel Findings. Nova Jersey, EUA: IEEE Transactions on Software Engineering, 2008.

LIMA, Max Wilian. **Uma extensão da ferramenta weka para avaliação de tarefas preditivas**. Acre, Brasil: Universidade Federal do Acre, 2017.

MURTA, Leonardo Gresta Paulino. **Gerência de configuração no desenvolvimento baseado em componentes.** Tese (Doutorado). Rio de Janeiro, Brasil: Universidade Federal do Rio de Janeiro. 2006.

NAGEL, Wilian A. **Subversion version control:** Using the Subversion version control system in development projects**.** Michigan, EUA: Prentice Hall Professional Technical Reference, 2005.

OLIVEIRA, Alciano Gustavo Genovez. **Aplicação da Lógica Nebulosa em um Classificador para Identificação de Perfis por Aspectos Cognitivos.** São paulo, Brasil: UNESP, 2016.

PACHECO, Andre. **K vizinhos mais próximos –** KNN. Disponível em: <http://www.computacaointeligente.com.br/algoritmos/knn-k-vizinhos-mais-proximos/>. Acesso em 10 de agosto de 2018.

PINHO, Alexandre Ferreira; MONTEVECHI, José Arnaldo Barra; MARINS, Fernando Augusto Silva; MIRANDA, Rafael de Carvalho. **Algoritmos genéticos**: Fundamentos e aplicações. ISBN 978-85-64619-10-4. 2013.

PRESSMAN, Roger. **Engenharia de software**. 5ª Ed. ISBN: 85-86804-25-8, Nova Iorque, EUA: McGraw Hill. 2002.

PRESSMAN, Roger. **Software Engineering:** A Practitioners Approach. 7ª Ed. ISBN: 0071267824, McGraw Hill. 2009.

REZENDE, D; ABREU, A. **Tecnologia da informação aplicada a sistemas de informação empresariais**. 3ª Ed. São Paulo, Brasil: Atlas, 2003.

SILVA, Daniel Augusto Nunes. **Wekapar** – uma extensão da ferramenta WEKA para auxiliar o pós-processamento de regras de associação. Acre, Brasil: Universidade Federal do Acre, 2016.

WAZLAWICK, Raul. **Metodologia de pesquisa para ciência da computação.** 2ª Ed. Brasil: Elsevier, 2014.

APÊNDICES

APÊNDICE A – DOCUMENTO DE REQUISITOS

Documento de Requisitos de Software

VARIAÇÕES PARA A EXTENSÃO TTSV DA FERRAMENTA WEKA

2.1

Desenvolvedor/Analista

Bruno Maia da Costa

Rio Branco – AC

2017

Histórico de Alterações

Data	Versão	Descrição	Autor
20/07/2018	2.1	Pequena alteração, RF4	Bruno Maia da Costa
15/07/2018	2.0	Modificações menores no nas seções 1 a 4 do documento de requisitos	Bruno Maia da Costa
03/07/2018	1.3	Modificações gerais no Documento de Requisitos	Bruno Maia da Costa
22/12/2017	1.2	Alteração nos requisitos funcionais	Bruno Maia da Costa
08/07/2017	1.1	Alteração nos requisitos não funcionais	Bruno Maia da Costa
17/06/2017	1.0	Criação do Documento de Requisitos	Bruno Maia da Costa

1. Análise do Problema

A janela deslizante é um método sensível ao tamanho dos conjuntos de treino e teste e que depende diretamente da qualidade dos dados usados para o treinamento, dependendo de tais dados, ocorrem variações nas medidas de avaliação o que pode melhorar ou piorar os resultados obtidos pelo método de avaliação. A divisão original em porcentagens utilizada para criar os conjuntos de treino e teste, limita as variações de distribuição dos dados de treinamento, já que o método sempre dividirá uma porcentagem dos dados, independente do número de dados que compõe essa porcentagem, ou de sua distribuição ao longo do tempo dentro da base de dados.

Essa limitação pode não atender as necessidades de todos os usuários da ferramenta, sendo necessário fornecer variações ao método para flexibilizar a sua utilização, tornando o método útil para um maior número de usuários.

2. Necessidades Básicas do Cliente

Desenvolver novas variações para a extensão TTSV, permitindo realizar as avaliações com janelas formadas por um número x de instâncias e janelas formadas pelas instâncias existentes em um intervalo entre datas.

3. Estudo de Viabilidade

Nesta seção serão apresentadas as justificativas que tornaram viáveis o desenvolvimento da solução proposta neste documento.

3.1. Viabilidade Técnica

A ferramenta WEKA dá todo o suporte necessário para o desenvolvimento de extensões, assim como uma clara documentação e exemplos de implementação. A extensão original já foi desenvolvida e apenas será alterada, sendo que a linguagem java foi utilizada no seu desenvolvimento, linguagem da qual o analista deste documento possui familiaridade, desta forma se justifica a viabilidade técnica para o desenvolvimento apresentado neste documento.

3.2. Viabilidade Econômica

As ferramentas utilizadas para o desenvolvimento do software com são livres, portanto é economicamente viável realizar o projeto. Uma vez que este projeto trata-se de um trabalho de conclusão de curso, não haverá custo financeiro associado.

3.3. Viabilidade Legal

A solução proposta neste documento de requisitos não possui nenhuma restrição legal, pois não infringe nenhuma lei existente, seja ela municipal, estadual ou federal.

4. Missão do Software

Expandir a extensão original, possibilitando atender um maior número de usuários.

5. Benefícios Gerais

ID	Benefício
B1	Maior abrangência de usuários.
B2	Maires opções de avaliação.

6. Restrições

ID	Restrição	Descrição
R1	Compatibilidade	A extensão não é compatível com versões da ferramenta WEKA que não dão suporte para uso de pacotes de novas funcionalidades.

7. Atores

ID	Atores	Descrição
A1	Minerador de dados	Usuário que realiza a mineração de dados com o uso da ferramenta WEKA

8. Requisitos Funcionais

ID	Funcionalidade	Necessidades	Prioridade
RF1	Implementar interface variação por número absoluto de instâncias	Implementar uma interface para a variação que permita realizar a avaliação com janelas compostas por números.	Essencial
RF2	Implementar interface variação por intervalo de datas	Implementar uma interface para a variação que permita realizar a avaliação com janelas compostas instâncias entre uma faixa de datas.	Essencial
RF3	Campo para número total de instâncias	Implementar um campo que mostre o número total de instâncias da base de dados selecionada.	Baixo
RF4	Campo para número previsto de modelos	Implementar um campo que mostre o número total de modelos que serão gerados com base no tamanho das janelas no método por número absoluto de instâncias.	Média
RF5	Modificar os métodos de recebimento de dados	Deverá ser adaptado o método original para recebimento e montagem dos conjuntos de treino e teste, de forma a atender os novos métodos implementados.	Média
RF6	Identificação do atributo temporal	Permitir identificar o atributo temporal no método por intervalo de datas.	Alta

9. Requisitos Não-Funcionais

ID	Requisitos	Categoria
NRF1	A interface gráfica deve acompanhar o padrão da extensão original.	Usabilidade
NRF2	É preciso desenvolver as variações na linguagem java.	Implementação

10. Requisitos de Hardware

No geral o desempenho da ferramenta WEKA está diretamente relacionada a memória RAM disponível, a extensão segue a mesma lógica, quanto maior o número de instâncias, mais memória é utilizada, além disso, alguns algoritmos são executados mais rapidamente em processadores mais potentes.

Como o WEKA não possui requisitos mínimos documentados, os requisitos a seguir são baseados na experiência de uso do autor desta pesquisa.

10.1. Configuração Mínima

- Processador dual core;
- 2GB de memória RAM;
- 10GB de espaço em disco;

10.2. Configuração Recomendada

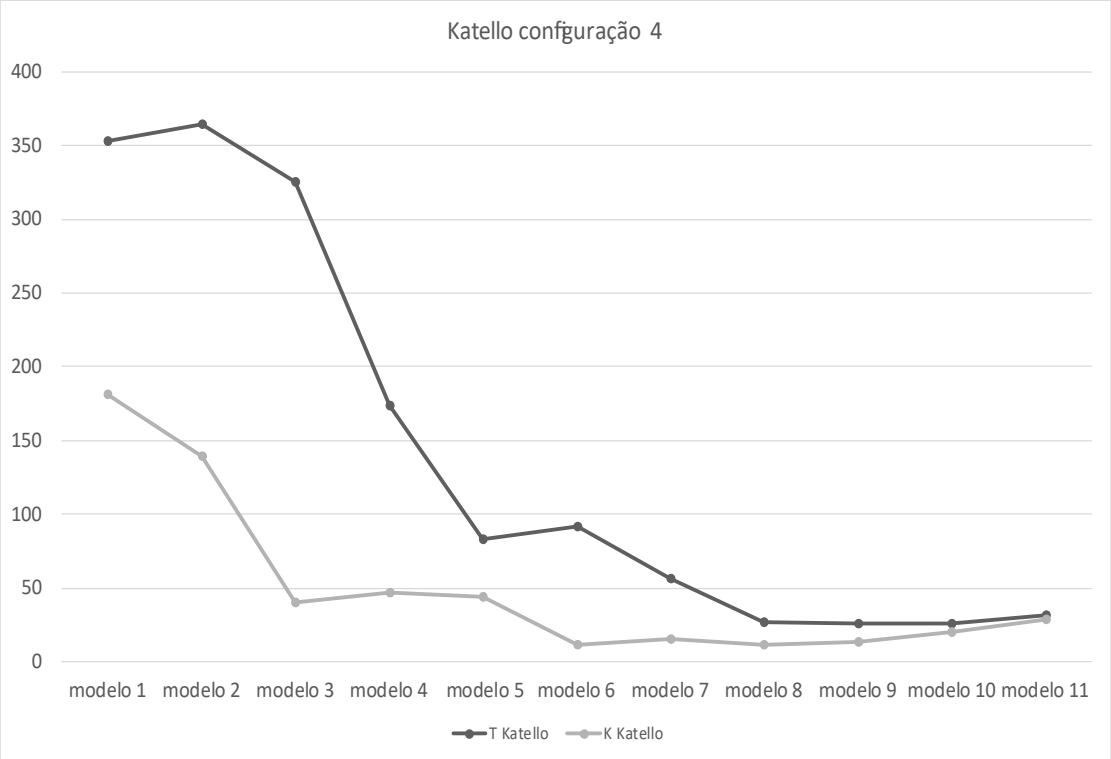
- Processador dual core 2,6ghz +;
- 8GB de memória RAM (com pelo menos 6gb disponibilizado exclusivamente para a ferramenta WEKA);
- 20GB de espaço em disco;

11. Ferramentas de Desenvolvimento e Licença de Uso

a. Eclipce oxygen 3 (Eclipce Public License);
b. WEKA 3.8.2 (General Public License);
c. Java Se Development Kit 8 (Oracle Binary Code License);

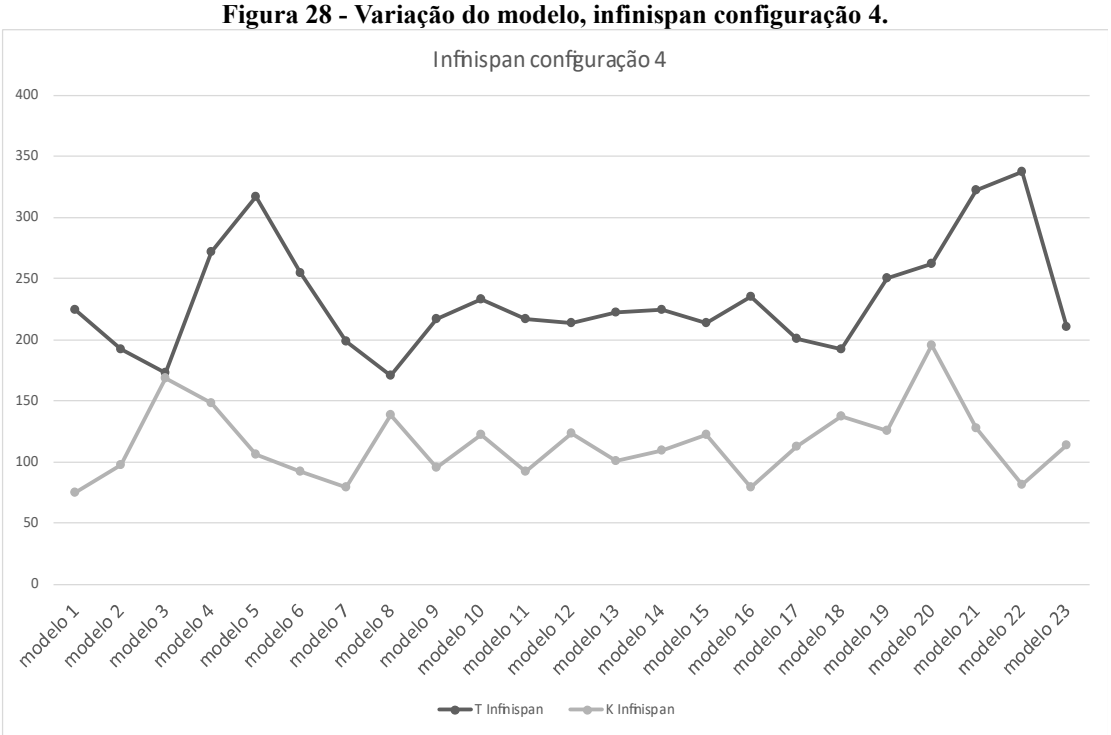
APÊNDICE B – GRÁFICOS DOS MODELOS

Figura 27 - Variação do modelo, Katello configuração 4.



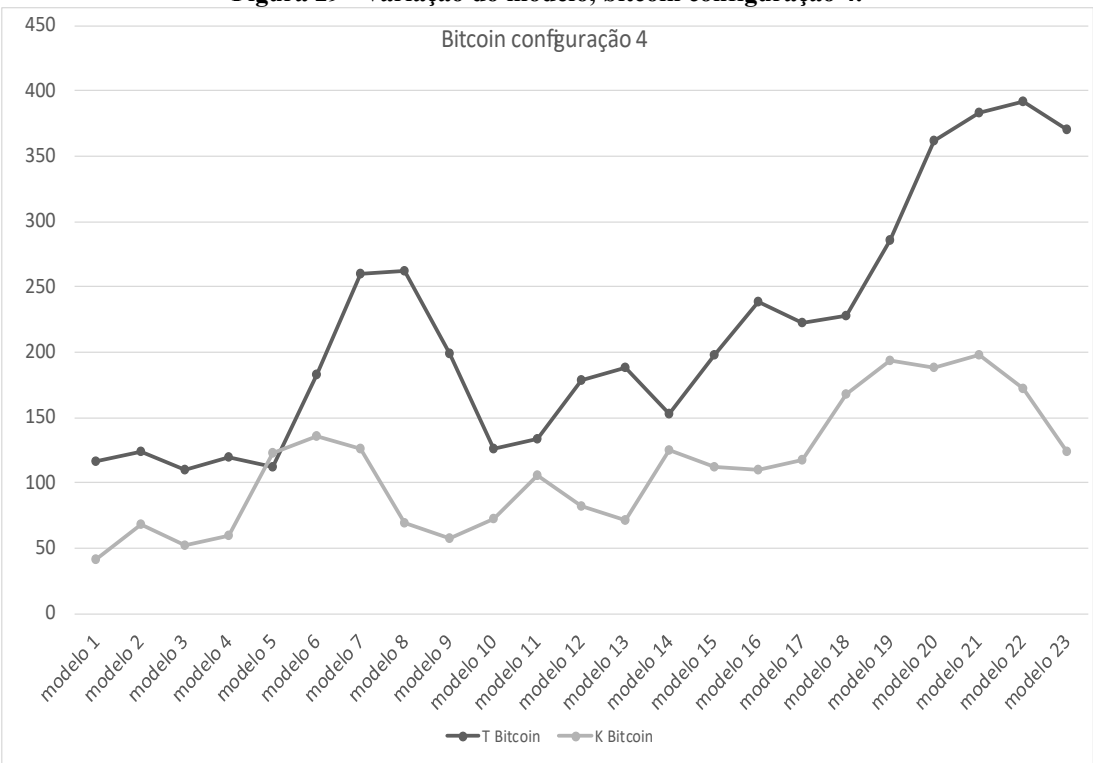
A Figura 27 demonstra as variações por modelo da base *katello* na configuração 4, onde é possível perceber que nenhum modelo ruim foi gerado, além disso a base gera um número menor de modelos em relação as demais, isso ocorre pelo fato da primeira instância da base possuir data de 2012, enquanto as demais bases possuem datas de 2011.

Figura 28 - Variação do modelo, infinispan configuração 4.



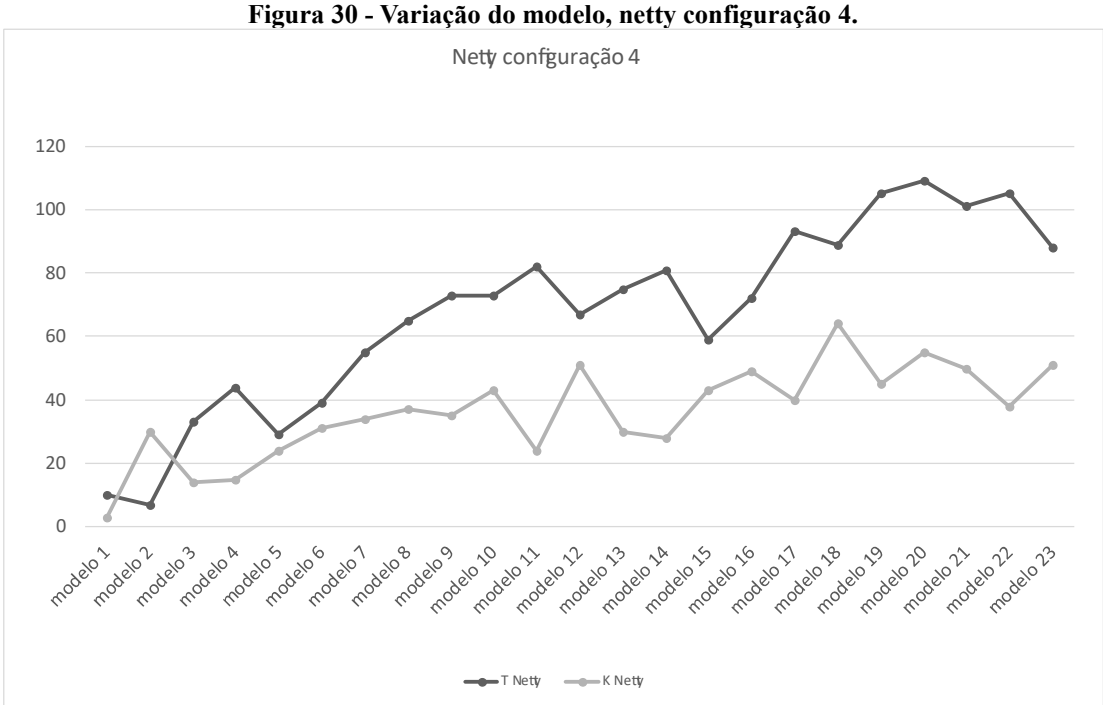
A Figura 28 demonstra as variações por modelo da base *infinispan* na configuração 4, onde é possível perceber que novamente nenhum modelo ruim foi gerado.

Figura 29 - Variação do modelo, bitcoin configuração 4.



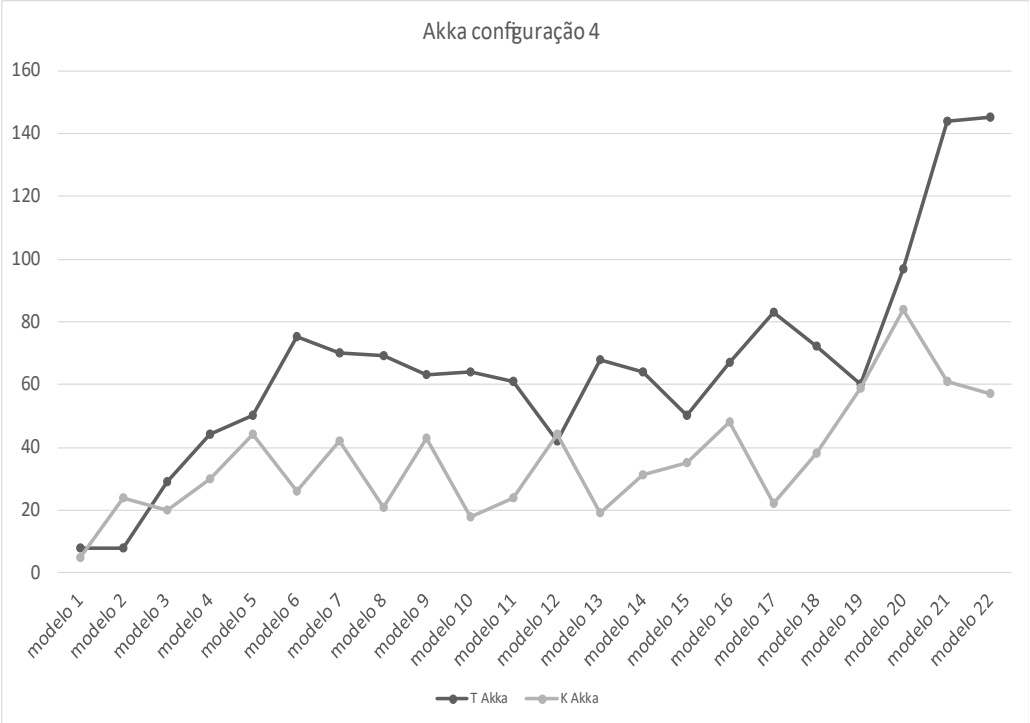
A Figura 29 demonstra as variações por modelo da base *bitcoin* na configuração 4, onde é possível perceber a geração de 1 modelo ruim (modelo 5), que possui uma quantidade um pouco inferior de instâncias em relação ao modelo de teste.

Figura 30 - Variação do modelo, netty configuração 4.



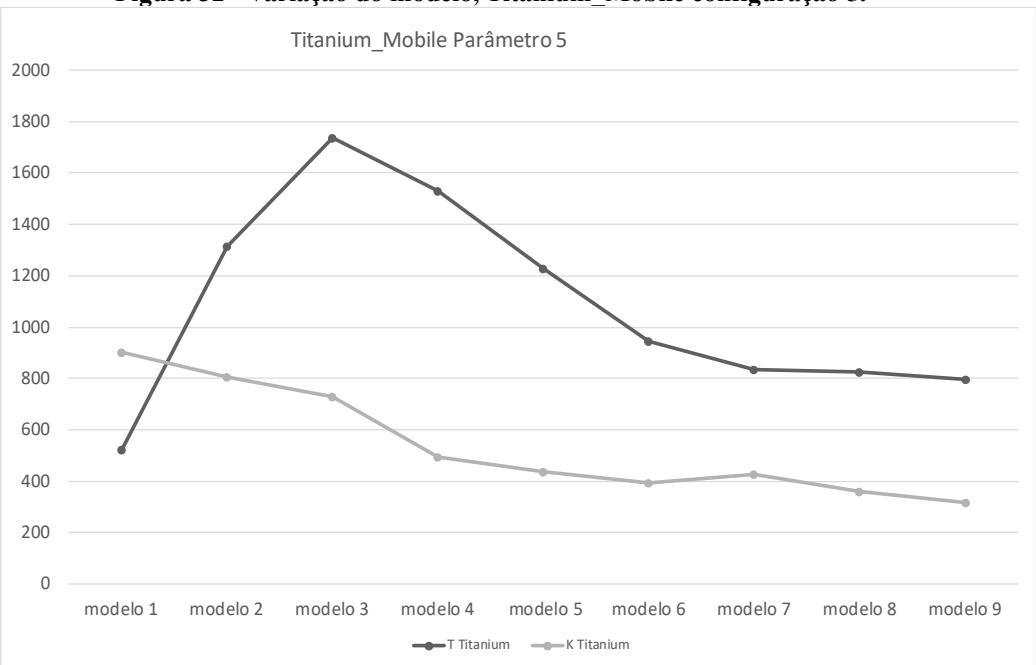
A Figura 30 demonstra as variações por modelo da base *netty* na configuração 4, onde é possível perceber novamente a geração de 1 modelo ruim (modelo 2).

Figura 31 - Variação do modelo, akka configuração 4.



A Figura 31 demonstra as variações por modelo da base *akka* na configuração 4, onde é possível perceber dessa vez a geração de 2 modelo ruim (modelo 2 e modelo 12).

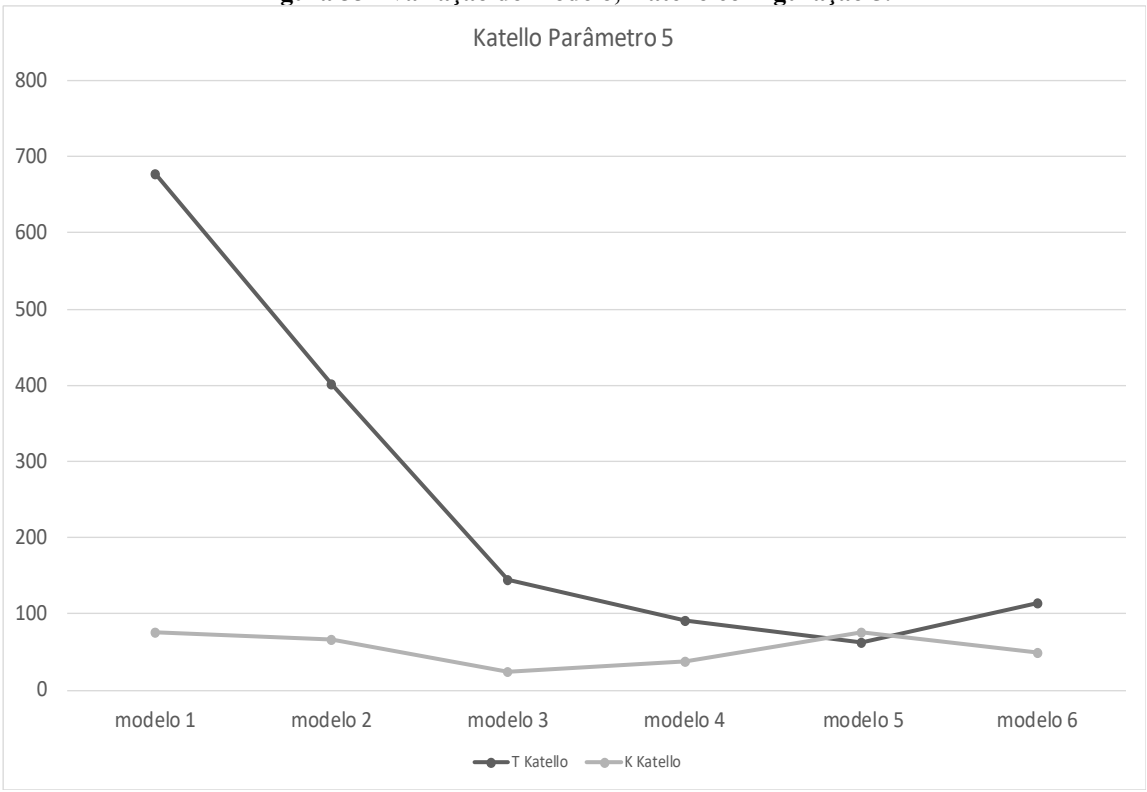
Figura 32 - Variação do modelo, Titanium_Mobile configuração 5.



A Figura 32 começa a demonstrar as variações obtidas com a configuração 5, a

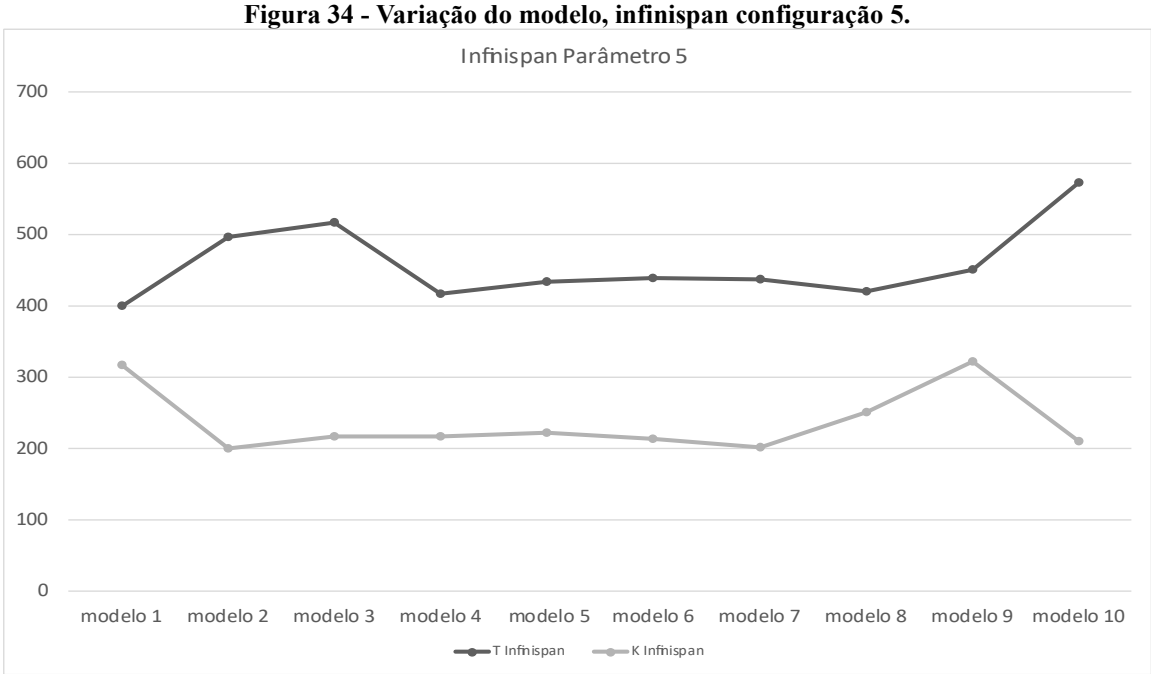
primeira base é *titanium_mobile*, percebe-se como o aumento das janelas reduz o número de modelos gerados e diferente da configuração 4 onde a base gerou 3 modelos ruins, na configuração 5 somente 1 modelo ruim foi gerado (modelo 1).

Figura 33 - Variação do modelo, Katello configuração 5.



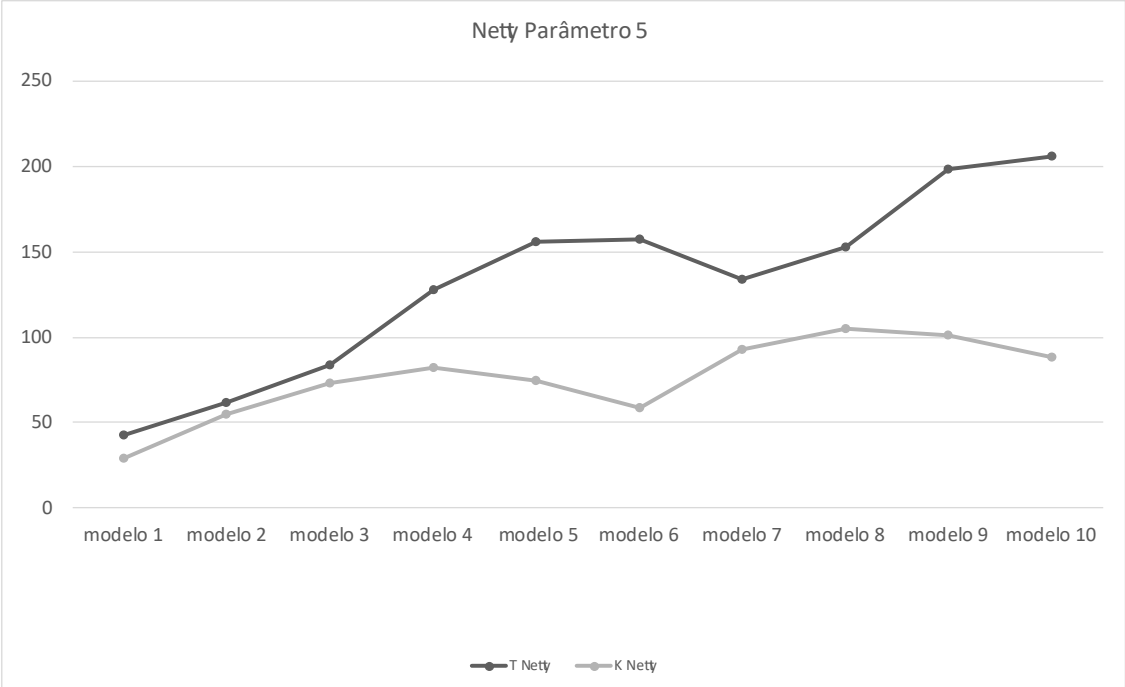
A Figura 33 demonstra as variações por modelo da base *katello* na configuração 5, onde diferente da configuração 4, 1 modelo ruim foi gerado na configuração 5 (modelo 5).

Figura 34 - Variação do modelo, infinispan configuração 5.



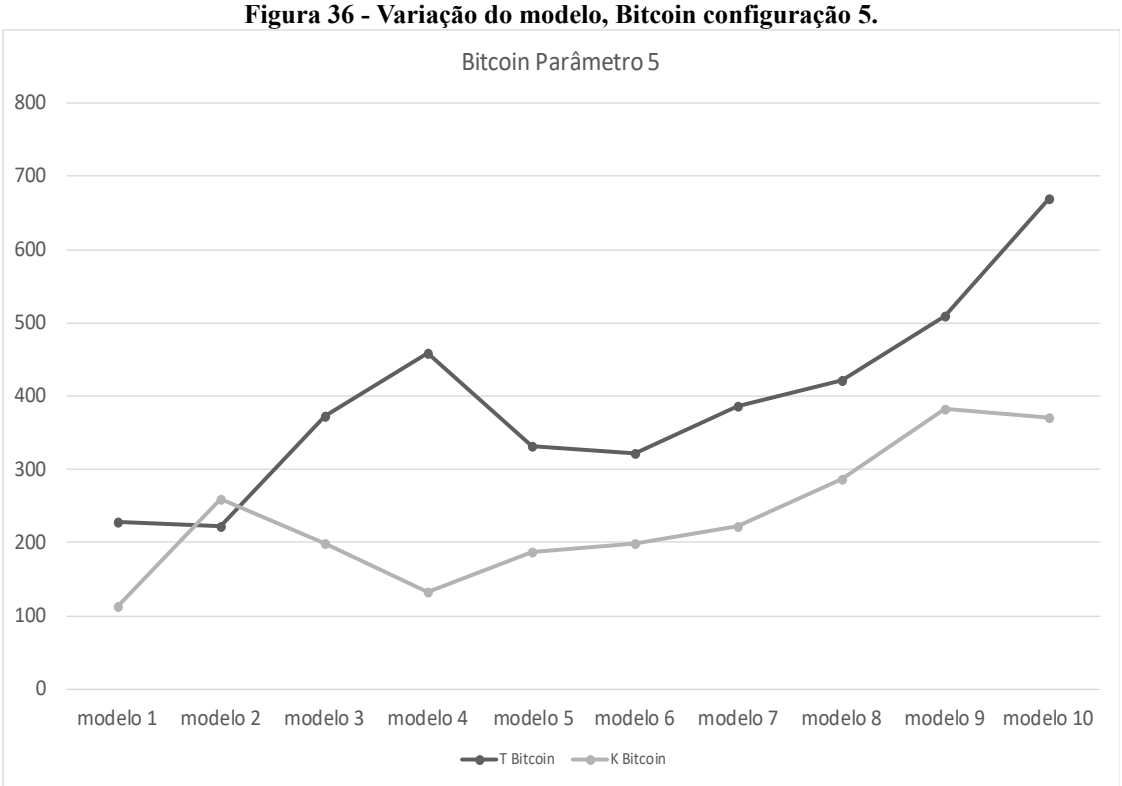
A Figura 34 demonstra as variações por modelo da base *infinispan* na configuração 5, onde novamente nenhum modelo ruim foi gerado.

Figura 35 - Variação do modelo, netty configuração 5.



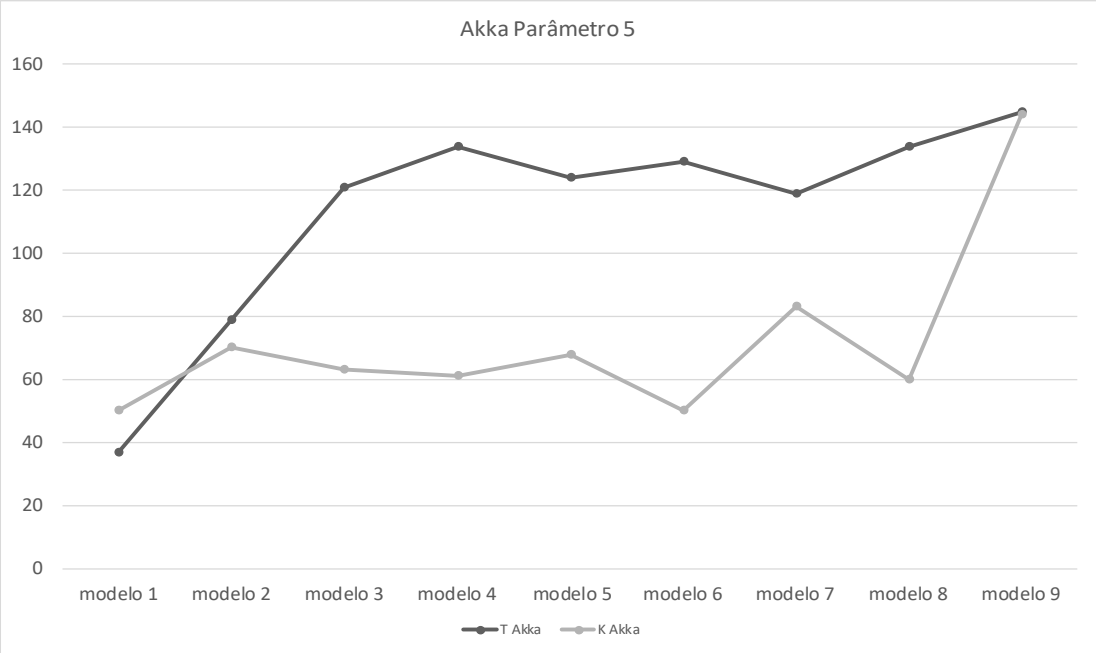
A Figura 35 demonstra as variações por modelo da base *infinispan* na configuração 5, onde novamente nenhum modelo ruim foi gerado.

Figura 36 - Variação do modelo, Bitcoin configuração 5.



Na Figura 36 temos as variações por modelo da base *bitcoin* na configuração 5, pode-se perceber a geração de um modelo ruim (modelo 2).

Figura 37 - Variação do modelo, akka configuração 5.



Na Figura 37 temos as variações por modelo da base *akka* na configuração 5, pode-se perceber a geração de um modelo ruim (modelo 1).