UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO
CÂMPUS CORNÉLIO PROCÓPIO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

DAIANY APARECIDA DA SILVA

# CLASSIFICAÇÃO DE INCOMPATIBILIDADES CROSS-BROWSER DE LAYOUT: UM ESTUDO COMPARATIVO ENTRE DIFERENTES MODELOS

DEFESA DE DISSERTAÇÃO – MESTRADO

**CORNÉLIO PROCÓPIO**

**2020**

DAIANY APARECIDA DA SILVA

# CLASSIFICAÇÃO DE INCOMPATIBILIDADES CROSS-BROWSER DE LAYOUT: UM ESTUDO COMPARATIVO ENTRE DIFERENTES MODELOS

Defesa de dissertação de mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Tecnológica Federal do Paraná – UTFPR como requisito parcial para a obtenção do título de "Mestre Profissional em Informática".

Orientador: Professor Willian Massami Watanabe

**CORNÉLIO PROCÓPIO**

**2020**

**Dados Internacionais de Catalogação na Publicação**

S586 Silva, Daiany Aparecida da

Classificação de incompatibilidades cross-browser de layout: um estudo comparativo entre diferentes modelos / Daiany Aparecida da Silva. – 2020.
39 f. : il. color. ; 31 cm.

Orientador: Willian Massami Watanabe.
Dissertação (Mestrado) – Universidade Tecnológica Federal do Paraná. Programa de Pós-Graduação em Informática. Cornélio Procópio, 2020.
Bibliografia: p. 38-39.

1. Browsers (Programas de. computador). 2. Software – Compatibilidade. 3. Software - Testes. 4. Informática - Dissertações. I. Watanabe, Willian Massami, orient. II. Universidade Tecnológica Federal do Paraná. Programa de Pós-Graduação em Informática. III. Título.

CDD (22. ed.) 004

**Biblioteca da UTFPR - Câmpus Cornélio Procópio**
Bibliotecário/Documentalista responsável:
Romeu Righetti de Araujo – CRB-9/1676

# RESUMO

SILVA, Daiany. CLASSIFICAÇÃO DE INCOMPATIBILIDADES CROSS-BROWSER DE LAYOUT: UM ESTUDO COMPARATIVO ENTRE DIFERENTES MODELOS. 39 f. Defesa de Dissertação – Mestrado – Programa de Pós-graduação em Informática, Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2020.

A complexidade do desenvolvimento de aplicativos *Web* está maior a cada dia devido ao grande número de navegadores disponíveis no mercado. Como consequência o número de inconsistências entre as aplicações *Web* também aumentou. Quando o mesmo aplicativo *Web* é renderizado em diferentes navegadores, as inconsistências detectadas no *layout* ou no comportamento das páginas são conhecidas como *XBIs* (*Cross Browser Incompatibilities*). O impacto de *XBIs* em aplicativos *Web* varia de pequenas inconsistências a falhas no *layout* ou na funcionalidade de um aplicativo *Web*. Assim, os *XBIs* podem afetar diretamente a experiência do usuário final durante a navegação no aplicativo *Web*. Para garantir a qualidade dos aplicativos *Web*, testadores e desenvolvedores devem inspecionar manualmente os aplicativos em cada navegador específico, para que os *XBIs* sejam identificados e corrigidos antes da implantação do sistema. Atualmente, existem diversos modelos na literatura para a identificação e correção automática de *XBIs*. Esses modelos evoluíram com o objetivo de reduzir falsos positivos e negativos. Este trabalho compara alguns modelos, focando aqueles que utilizam a classificação de *XBIs* de *layout*, por meio de algoritmos de aprendizado de máquina. Ainda não há um trabalho na literatura que faça essa comparação, identificando suas principais vantagens e desvantagens. Este trabalho consiste em um experimento que compara os resultados dos modelos e apresenta métricas de eficácia, visando trazer informações importantes como contribuições para propor trabalhos futuros em relação à evolução dos modelos explorados. Nesse experimento o resultado é o valor obtido pela métrica da Medida F. Para essa métrica, os valores mais altos implicam maior eficiência na detecção de incompatibilidades entre os navegadores, e a configuração C5.0 10 iterações - Paes, Watanabe obteve o melhor resultado nesse experimento.

**Palavras-chave:** Incompatibilidade entre navegadores, Qualidade de software, Testes de software, Automação de testes, Engenharia de Software

## ABSTRACT

SILVA, Daiany. CLASSIFICATION OF CROSS-BROWSER LAYOUT INCOMPATIBILITIES: A COMPARATIVE STUDY BETWEEN DIFFERENT MODELS. 39 f. Defesa de Dissertação – Mestrado – Programa de Pós-graduação em Informática, Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2020.

The complexity of developing Web applications is increasing every day due to the large number of browsers available on the market. As a consequence, the number of inconsistencies between Web applications has also increased. When the same Web application is rendered in different browsers, the inconsistencies detected in the layout or behavior of the pages are known as (XBIs - Cross Browser Incompatibilities). The impact of XBIs on Web applications ranges from minor inconsistencies to critical flaws in the layout or functionality of a Web application. Thus, XBIs can directly affect the end user experience while browsing the Web (SILVA, 2005) application. To ensure the quality of web applications, testers and developers must manually inspect the applications in each specific browser, so that XBIs are identified and corrected before the system is deployed. Currently, there are several approaches in the literature for the identification and automatic correction of XBIs. These approaches have evolved with the aim of reducing false positives and negatives. This work proposes to compare some of the approaches, focusing on those that use the classification of XBIs of layout, through machine learning algorithms. There is still no work in the literature that makes this comparison, identifying its main advantages and disadvantages. This paper consists of an experiment that compares the results of the approaches and presents metrics that allow to affirm their effectiveness, aiming to bring important information as contributions to propose future works in relation to the evolution of the explored approaches. The result of the experiment is the F-Score metric. For this metric, the higher values imply greater efficiency in detecting incompatibilities between browsers, and the C5.0 10 iterations - Paes, Watanabe configuration obtained the best result in the experiment.



**Keywords:** Cross-browser incompatibility, Quality of software, Software testing, Test automation, Software engineering

## LISTA DE FIGURAS

## LISTA DE TABELAS

**SUMÁRIO**

# 1 INTRODUÇÃO

A alta competitividade na área de desenvolvimento de *software* tem pressionado as empresas a buscarem soluções para aumentar a qualidade e a agilidade das entregas. É crescente a busca pela redução dos custos e o aumento da qualidade em relação ao processo de desenvolvimento do *software*.

Os recentes avanços tecnológicos aumentaram a complexidade do desenvolvimento das aplicações *Web*, e como agravante, atualmente existe um grande número de navegadores *Web* distintos, como: *Internet Explorer, Mozilla Firefox, Opera, Google Chrome,* entre outros (CHOUDHARY et al., 2010).

A flexibilidade da escolha de navegadores aumenta a portabilidade das aplicações *Web* em relação aos dispositivos que irão renderizá-las, entretanto, também pode resultar em inconsistências na aparência ou nas funcionalidades. Essas inconsistências, que podem variar de pequenas diferenças no *layout* a falhas funcionais, são conhecidas na literatura como *XBIs* (*Cross Browser Incompatibilities*) (CHOUDHARY et al., 2012), conforme o exemplo apresentado na Figura 1.1. Na imagem pode-se observar inconsistências no elementos da tela quando a aplicação é renderizada em diferentes navegadores.

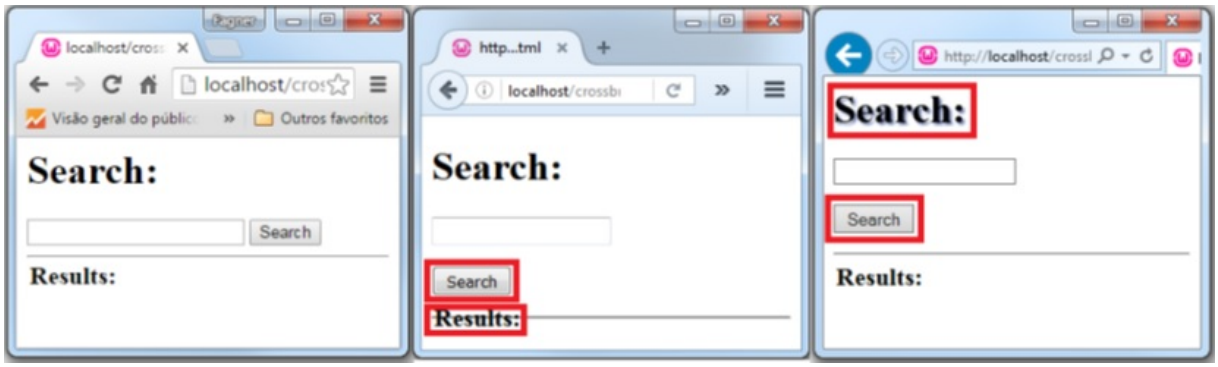


**Figura 1.1**: Exemplo de XBIs. (PAES; WATANABE, 2018).

De acordo com Choudhary et al. (2013), se os *XBIs* não forem identificados antes da entrega da aplicação, eles podem prejudicar a experiência dos usuários. Alguns *XBIs* podem impedir que os usuários acessem funcionalidades, tornando o aplicativo inútil. Por isso há uma grande preocupação das empresas em identificar e corrigir *XBIs* de maneira rápida e barata, a fim de evitar danos a sua imagem. No geral, testes em aplicativos *Web* requerem esforços manuais, o que torna o processo lento e custoso. Nesse cenário, as empresas buscam formas de automatizar o processo para agilizá-lo e barateá-lo (DALLMEIER et al., 2012).

Atualmente há um esforço no desenvolvimento de modelos que façam a identificação e

até a correção automática de *XBIs*. O estado da arte apresenta diferentes modelos para a identificação de *XBIs*, como por exemplo: *WebDiff* (CHOUDHARY et al., 2010); *CrossT* (MESBAH; PRASAD, 2011); *CrossCheck* (CHOUDHARY et al., 2012); *WebMate* (DALLMEIER et al., 2012); *X-Pert* (CHOUDHARY et al., 2013); *Browserbite* (SEMENENKO et al., 2013) *X-Check* (HE et al., 2016) e Paes, Watanabe (PAES; WATANABE, 2018).

A automatização de detecção dos *XBIs* reduz significativamente a quantidade necessária de esforço manual, entretanto a identificação automática de *XBIs*, muitas vezes, pode apresentar um excesso de sensibilidade e produz uma quantidade excessiva de falsos positivos e falso negativos (SEMENENKO et al., 2013).

Esse problema se deve ao fato de que um *XBI*, em oposição a uma simples diferença da aplicação, é até certo ponto subjetivo. Assim, definir limiares específicos para classificar uma diferença como um *XBI* ou não, está longe de ser uma tarefa fácil (SEMENENKO et al., 2013). Falsos positivos podem ser considerados pequenas diferenças nos elementos ou estrutura da página que são imperceptíveis aos olhos humanos, portanto não devem ser considerados *XBIs* mas ainda sim, são identificados pelos modelos. Enquanto falsos negativos são inconsistências perceptíveis aos olhos humanos que os modelos não conseguem detectar (CHOUDHARY, 2011).

Trabalhos como (CHOUDHARY et al., 2012), (SEMENENKO et al., 2013) e (PAES; WATANABE, 2018), apresentam modelos para classificar os *XBIs* e facilitar a identificação dos mesmos utilizando algoritmos de aprendizado de máquina. Esses modelos utilizam características que são extraídas das páginas *Web*, a fim de modelar um classificador que tenha capacidade de identificar *XBIs* de maneira eficaz.

Esse trabalho apresenta uma comparação de resultados obtidos entre os modelos *CrossCheck* (CHOUDHARY et al., 2012), *Browserbite* (SEMENENKO et al., 2013) e Paes, Watanabe (PAES; WATANABE, 2018). No estudo foi avaliado apenas *XBIs* de *layout*. Foi realizada a classificação manual para a identificação de *XBIs* em diversas páginas *Web*. As informações obtidas com a classificação foram utilizadas para constituir um *dataset* e validar os modelos para o estudo, em seguida os resultados obtidos foram comparados. Os dados coletados passaram por uma análise com a intenção de fornecer informações que possam contribuir para a melhoria e evolução do processo de identificação automática de *XBIs*.

A estrutura do trabalho é apresentada da seguinte maneira, no Capítulo 2 é apresentado a Fundamentação Teórica, a Incompatibilidade Entre Navegadores (XBIs), Aprendizado de Máquina Aplicado na Identificação de *XBIs*, os Trabalhos Relacionados e as Características Utilizadas Pelos Modelos, no Capítulo 3 a Avaliação, Resultados e Discussões, Limitação

do Trabalho e as Considerações Finais, no Capítulo 4 as Conclusões e por fim as Referências Bibliográficas.

# 2 FUNDAMENTAÇÃO TEÓRICA

Devido à crescente popularidade dos aplicativos *Web* e ao número de navegadores nos quais esses aplicativos podem ser executados, os *XBIs* são uma preocupação para as organizações que desenvolvem *software* baseado na *Web* (CHOUDHARY et al., 2013). A literatura apresenta modelos de identificação automatizadas de *XBIs*, algum desses fazem uso de algoritmos de aprendizado de máquina como uma maneira de agilizar esse processo, além de barateá-lo.

## 2.1 INCOMPATIBILIDADES ENTRE NAVEGADORES (*XBIS*)

Atualmente o número de navegadores ofertados no mercado é grande, o que aumenta a portabilidade das aplicações, mas ao mesmo tempo, devido as diferentes tecnologias utilizadas no desenvolvimento, geram um maior número de incompatibilidades, quando a mesma aplicação é executada em diferentes navegadores.

Os navegadores *Web* são essencialmente desenvolvidos com base nas seguintes tecnologias (PAES; WATANABE, 2018):

- ***HTML (Hypertext Markup Language)***, que contém a estrutura de marcações de um texto;
- ***CSS (Cascading Style Sheets)*** utilizado para definir o estilo da página e;
- ***JS(JavaScript)*** que lida com o comportamento dinâmico e a interatividade de uma página *Web*.

Diferenças no motor de renderização do navegador em cada uma dessas tecnologias podem levar a um comportamento inconsistente na apresentação. Essas inconsistências foram referenciadas em diversos estudos com o termo Incompatibilidade Entre Navegadores ou *Cross-browser Incompatibilities* (*XBIs*). Os *XBIs* são inconsistências apresentadas ou na aparência ou na funcionalidade de uma página *Web*, quando executada em diferentes navegadores (CHOUDHARY et al., 2013).

Segundo Choudhary et al. (2013), os *XBIs* podem ser classificados em 3 tipos:

- ***XBIs* de Estrutura:** são inconsistências encontradas ou na estrutura *html* ou no *layout* da página. Algumas vezes as páginas não apresentam a mesma quantidade de elementos *html* ou apresentam inconsistências na forma como alguns componentes estão organizados na estrutura da página. Exemplo: a Figura 2.1 apresenta uma inconsistência de estrutura já

que a posição (Position left) do elemento, quando apresentado nos navegadores *Internet Explorer* e *Google Chrome* não são similares.

- ***XBIs* de Conteúdo:** são as inconsistências observadas no conteúdo interno de um elemento DOM. O conteúdo interno consiste na representação da imagem de um elemento DOM, são inconsistências encontradas nos elementos individualmente, por exemplo, na disposição dos textos ou formatação dos elementos. A Figura 2.2 apresenta problema de compatibilidade de conteúdo relatado no *Mozilla Firefox*, que renderiza o elemento de entrada de texto com cor diferente em comparação com o mesmo elemento renderizado no *Google Chrome*.

- ***XBIs* de Comportamento:** envolvem inconsistências no comportamento de componentes funcionais. Por exemplo, um botão que exerce uma ação dentro de um navegador e em outro navegador a ação não é executada.

Além disso, (PAES; WATANABE, 2018) classifica ainda os *XBIs* de acordo com as seguintes classes:

- ***XBI* Externo:** são inconsistências observadas no posicionamento (horizontal e vertical) e tamanho (altura e largura) dos elementos DOM. Exemplo: a Figura 2.5 apresenta o mesmo elemento com a posição e o tamanho diferente quando executados no *Chrome* e *Internet Explorer*.

- ***XBI* Interno:** são inconsistências observadas no conteúdo interno de um elemento DOM ou na estrutura interna da página. Por exemplo diferenças de formatação no texto em um determinado elemento DOM, ou diferenças na quantidade de elementos *html* da página. A Figura 2.3 apresenta incompatibilidades no conteúdo interno do componente *checkbox*, pois são apresentados com aparência diferente quando executados no *Chrome* e *Firefox*. A Figura 2.4 também apresenta incompatibilidade no conteúdo interno do componente já que o texto "Cadastre-se"se encontra se maneiras diferentes na tela quando executados no *Chrome* e *Internet Explorer*.

## 2.2 APRENDIZADO DE MÁQUINA APLICADO A IDENTIFICAÇÃO DE XBIS

O aprendizado de máquina é uma área da inteligência artificial, onde o objetivo é desenvolver técnicas computacionais que são capazes de adquirir automaticamente o conhecimento sobre como executar uma tarefa específica, sem ser explicitamente programada para fazê-la. Segundo (WEISS; KULIKOWSKI, 1991), o aprendizado de máquina é como um programa

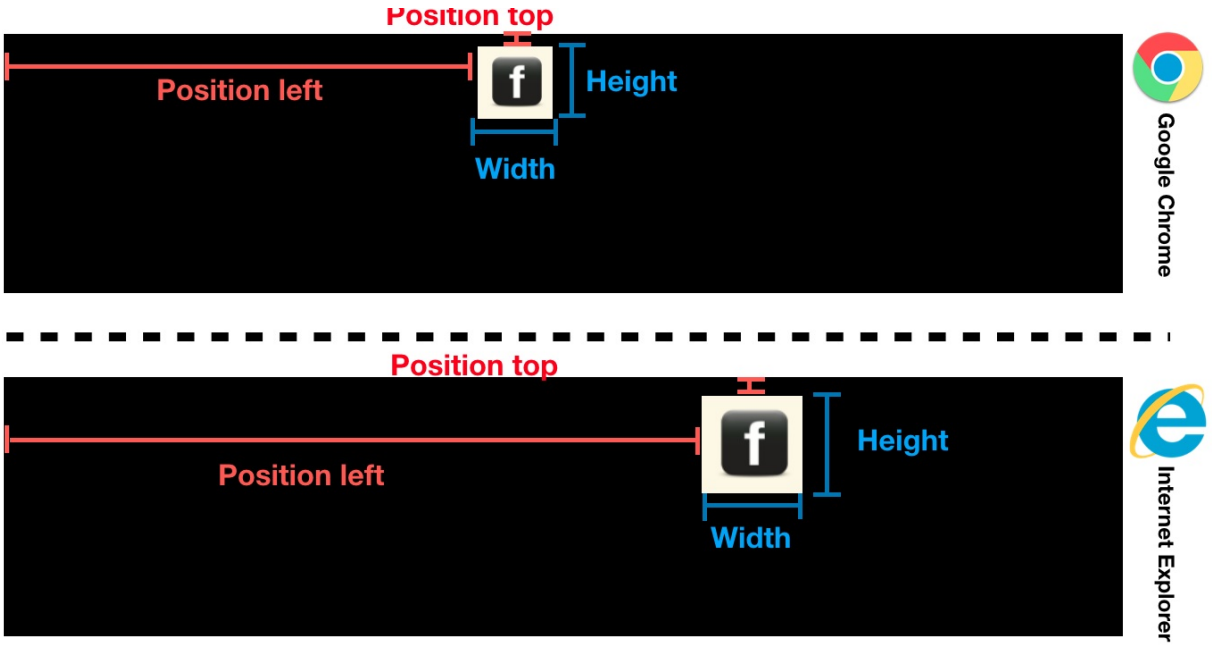


**Figura 2.1**: Incompatibilidade de Estrutura. (PAES; WATANABE, 2018)

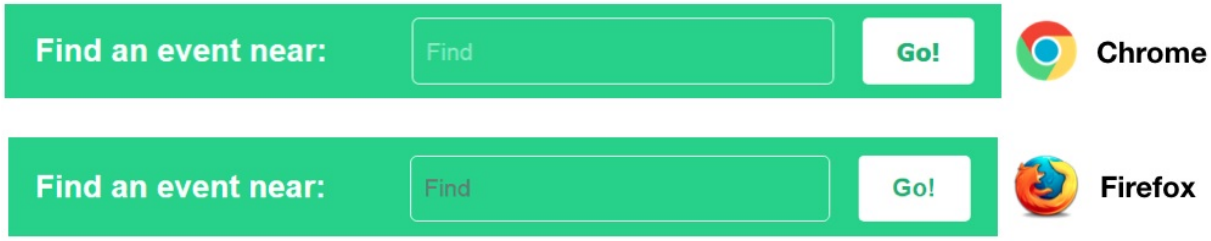


**Figura 2.2**: Incompatibilidade de Conteúdo (PAES; WATANABE, 2018).

**Figura 2.3**: Incompatibilidade de Conteúdo Interno(PAES; WATANABE, 2018).


**Figura 2.4**: Incompatibilidade de Conteúdo Interno (PAES; WATANABE, 2018).

de computador que toma decisões com base em experiências adquiridas e acumuladas através de soluções bem sucedidas de problemas anteriores. As técnicas de aprendizado de máquina são divididas em três principais categorias: aprendizado supervisionado, não supervisionado e por reforço. Os modelos explorados nesse trabalho utilizam o aprendizado supervisionado, onde conceitos são induzidos a partir de exemplos que são pré-classificados. Todos os modelos selecionados usam exemplos de códigos em HTML, CSS e JAVASCRIPT de aplicativos *Web* que continham *XBIs* e exemplos de códigos que não continham. Esses exemplos, são utilizados

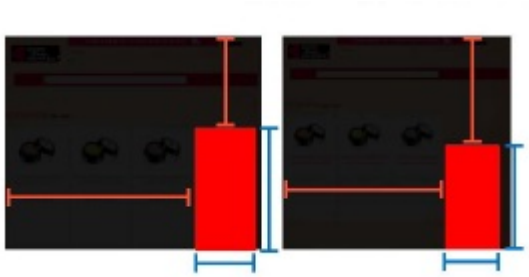

**Figura 2.5**: Incompatibilidade de Conteúdo Externo(PAES; WATANABE, 2018).

como dados de aprendizagem para as técnicas de aprendizado de máquina. Nesse trabalho foi utilizado o modelo de classificação dos algoritmos de árvore de decisão com a validação cruzada com o *K-fold*=10. As árvores de decisão são algoritmos de classificação de aprendizado de máquina supervisionados e uma das representações mais populares de classificadores (ROKACH; MAIMON, 2005). Sua popularidade está associada à capacidade de capturar preditores não lineares quase arbitrários com base nos dados de entrada (QUINLAN, 1986). São simples representações de um classificador utilizado por muitos sistemas de aprendizado de máquina tal como C4.5, (QUINLAN et al., 1996). A validação cruzada é uma técnica para avaliar a capacidade de generalização de um modelo, a partir de um conjunto de dados. Esta técnica é amplamente empregada em problemas onde o objetivo da modelagem é a predição. Busca-se então estimar o quão preciso é este modelo na prática, ou seja, o seu desempenho para um novo conjunto de dados (KOHAVI et al., 1995).

O conceito central das técnicas de validação cruzada é o particionamento do conjunto de dados em subconjuntos mutuamente exclusivos, e posteriormente, o uso de alguns destes subconjuntos para a estimação dos parâmetros do modelo (dados de treinamento), sendo os subconjuntos restantes (dados de validação ou de teste) empregados na validação do modelo (KOHAVI et al., 1995).

Os modelos selecionados utilizaram árvore de decisão para classificar cada elemento da estrutura DOM renderizado em diferentes navegadores que apresentaram *XBI* ou não.

## 2.3 TRABALHOS RELACIONADOS

Para detectar *XBIs* em aplicações *Web*, os desenvolvedores renderizam o aplicativo *Web* em diferentes navegadores e inspecionam manualmente o *layout* e as funcionalidades. Para facilitar esse processo e ao mesmo tempo barateá-lo, modelos comerciais como: *Microsoft*

*Visual Studio Test Professional* [1] e *Firebase Test Lab* [2], podem auxiliar na inspeção manual, apresentando uma comparação do aplicativo *Web* em dois navegadores distintos. No entanto, esses modelos ainda exigem que o desenvolvedor gaste uma quantidade considerável de esforços para identificar, entender e corrigir manualmente os problemas reais entre navegadores (MALLA, 2017).

Existem atualmente diferentes modelos para a identificação automática de *XBIs*, eles comparam os recursos coletados nos dois navegadores e, se forem diferentes, decidem se a diferença é um *XBI* ou não. Alguns modelos existentes fazem uso de algoritmos de aprendizado de máquina para classificar as inconsistências detectadas. O algoritmos mais utilizados por esses modelos são: redes neurais e árvore de decisões.

Há um grande esforço no desenvolvimento de modelos que façam a identificação e até a correção automática de *XBIs*. O estado da arte apresenta vários modelos para a identificação automática de *XBIs*, tais como: WebDiff (CHOUDHARY et al., 2010), CrossCheck (CHOUDHARY et al., 2012), X-Pert (CHOUDHARY et al., 2013) e o Browserbite (SEMENENKO et al., 2013). Entre os modelos para a identificação de *XBIS* são utilizadas técnicas de análise da estrutura DOM, comparações de captura de tela, análise de grafos de estados, análise de fragmentação de imagens e aprendizagem de máquina (PAES; WATANABE, 2018).

### 2.3.1 WEBDIFF

O *WebDiff* é um modelo que considera um navegador como “referência"e compara o comportamento da página *Web* em vários navegadores distintos. A técnica *WebDiff* é composta por cinco fases diferentes, conforme apresentado na Figura 2.6.

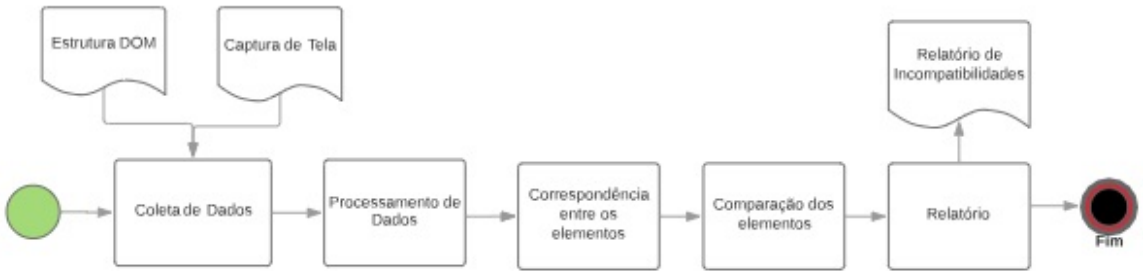


**Figura 2.6**: Processo de Identificação de *XBIs* pelo *WebDiff*.

Na primeira etapa (Coleta de dados) do *WebDiff*, a página *Web* a ser testada é carregada

[1] https://visualstudio.microsoft.com/pt-br/vs/test-professional/
[2] https://firebase.google.com/products/test-lab?hl=pt-br

em todos os navegadores selecionados para os testes, em seguida, extrai uma árvore DOM da página *Web* junto com suas capturas de tela, essas etapas são conduzidas para todos os navegadores (CHOUDHARY, 2011). As árvores DOM (Document Object Model) são estruturas hierárquicas organizadas em tempo de execução formadas pelos elementos HTML de uma aplicação *Web*, como $< body >$ e $< p >$ $(www.w3.org)$. O DOM [3] (*Document Object Model*) define uma estrutura lógica do documento HTML e XML que permite os programas e *scripts* acessem e atualizem conteúdo dinamicamente. O DOM representa os documentos HTML e XML como uma estrutura de dados em árvore e os navegadores frequentemente fornecem uma API (*Application Program Interface*) para ser usada pela aplicação *Web*.

Na segunda etapa (Processamento dos Dados), o modelo identifica os elementos variáveis, a página é executada novamente no navegador e outra extração da árvore DOM e da tela são realizadas. Os dados obtidos são comparados e os nós com diferenças são marcados como variáveis (CHOUDHARY, 2011).

Na terceira etapa (Correspondência entre os Elementos), o *WebDiff* identifica nós correspondentes da página executada no navegador “referência", com os nós da página executada em outros navegadores. O modelo faz isso percorrendo duas árvores DOM em paralelo para encontrar um par de nós com a melhor correspondência (CHOUDHARY, 2011).

Na quarta etapa (Comparação dos elementos) é realizada a comparação real das informações. Para cada par de nós correspondentes em duas árvores DOM, serão identificadas diferenças nas propriedades, caso existam. Essa etapa compara as seções das capturas de telas para os nós cuja as propriedades são correspondentes, a fim de identificar diferenças (CHOUDHARY, 2011). O elemento DOM não possui informações sobre como os elementos da página *Web* aparecem exatamente na tela. Nesta etapa, o *WebDiff* aproveita as informações estruturais calculadas na etapa anterior para (1) identificar os elementos correspondentes nas capturas de tela do navegador e (2) executar uma correspondência gráfica de tais elementos. O *WebDiff* verifica quatro classes de problemas: diferenças de posição, tamanho, visibilidade e aparência. Para realizar essa comparação visual, é utilizada uma técnica (EMD - Earth Mover’s Distance) baseada em histograma (RUBNER et al., 2000).

Na quinta e última etapa (Relatório) é gerado um relatório com a lista de diferenças identificadas na etapa anterior. Cada item na lista indica o tipo do problema, o local na tela onde foi identificado e o caminho do elemento DOM envolvido nesse problema (CHOUDHARY, 2011).

Os resultados do *WebDiff* relatam os diferentes tipos de problemas como: diferenças

[3]DOM: https://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/

posicionais, mudanças no tamanho e diferenças gerais de aparência. O modelo foi capaz de descobrir e relatar automaticamente um grande número *XBIs*. No geral, o *WebDiff* relatou 121 problemas de diferentes tipos e reportou 21 falsos positivos para nove aplicações *Web* consideradas no estudo. Uma investigação dos falsos positivos mostrou que muitos deles são devidos a pequenas diferenças em alguns elementos que contêm texto e imagens de fundo que são imperceptíveis para o olho humano, e a presença de alguns elementos variáveis que o *WebDiff* não conseguiu identificar (CHOUDHARY, 2011). Os autores concluíram que os resultados foram satisfatórios, já que o modelo foi precursor na área.

### 2.3.2 CROSSCHECK

O *CrossCheck* combina dois modelos complementares para detecção automática de *XBIs*: *WebDiff* e *CrossT* (RUBNER et al., 2000). O *CrossT* é um modelo que consiste em analisar automaticamente o aplicativo *Web* fornecido em diferentes ambientes de navegador, e capturar o comportamento como uma máquina de estado finito, comparando formalmente os modelos gerados para equivalência em pares e expondo quaisquer discrepâncias observadas Rubner et al. (2000). O *WeDiff* opera em páginas *Web* e se concentra em encontrar *XBIs* estruturais. Em contraste, o *CrossT* pode analisar aplicativos *Web* por inteiro, e se concentra em encontrar *XBIs* comportamentais. O *CrossCheck* utiliza o *Crawljax* para criar um grafo de estados do aplicativo *Web* em dois navegadores distintos. Em seguida, verifica se os dois grafos são isomórficos para detectar *XBIs* no nível comportamental (CHOUDHARY et al., 2012).

O modelo do *CrossCheck* é composta por três etapas, conforme apresentado na Figura 2.7.

Na primeira etapa é realizado um rastreio automático (Reatreamento e Captura dos Modelos), utilizando o *CrawlJax* (`http://crawljax.com`) (detecta estados dinâmicos de aplicativos *Web* baseados em *AJAX*). Ele captura e registra o comportamento observado na forma de dois modelos de navegação, M1 e M2, um para cada navegador (CHOUDHARY et al., 2012).

A segunda etapa (Compara os Modelos) compara os modelos M1 e M2 para verificar se são equivalentes e extrair um conjunto de diferenças. Essas diferenças representam possíveis manifestações de *XBIs* no comportamento do aplicativo *Web* e são usadas na fase final do algoritmo para identificar um conjunto de *XBIs*. A comparação é realizada em três etapas, representando análises em três diferentes níveis de abstração do modelo.

1) Comparação do Nível de Rastreamento: Esta etapa compara os grafos de estado

G1 e G2 (dos modelos M1 e M2). Para tanto, utiliza o algoritmo de isomorfismo proposto em (MESBAH; PRASAD, 2011), o qual encontra um mapeamento um a um entre os nós e as bordas de G1 e G2.

2) Comparação de Pares de Telas Correspondentes: Esta etapa repete os pares de telas correspondentes e, para cada par, compara os elementos DOM. O algoritmo de correspondência DOM segue em grande parte o proposto em (CHOUDHARY, 2011). A principal diferença entre os dois algoritmos é o cálculo da métrica de índice de correspondência - um número entre 0 e 1 que quantifica a similaridade entre dois nós DOM. Quanto maior o índice de correspondência, maior a probabilidade de os dois elementos DOM representarem o mesmo elemento no contexto de suas respectivas telas.

3) Comparação Visual de Pares de Elementos de Tela Combinados: Este é o último e mais importante passo na comparação do modelo. Essa etapa repete os pares de elementos DOM e, para cada par de nós DOM correspondentes, compara visualmente os elementos de tela. Se os dois nós encontrados são diferentes, eles são adicionados à lista de diferenças da tela. A comparação visual usa um classificador construído através de aprendizado de máquina.

Para realizar a classificação dos *XBIs* utiliza-se o algoritmo “Arvore de Decisão". Esse algoritmo é utilizado para comparar diferenças no tamanho, no deslocamento, na área, nos textos dos elementos DOM e também nas distâncias entre as imagens. Para calcular a distância entre as imagens, são utilizadas as técnicas de histograma e $\chi^2$ (CHOUDHARY et al., 2012).

A terceira etapa (Gera Relatório) analisa esse conjunto de diferenças do modelo e as agrupa em um conjunto de *XBIs*, que são então apresentadas ao usuário final (CHOUDHARY et al., 2012).

As diferenças entre navegadores relatadas por *CrossCheck*, relatam os seguintes tipo de problema: diferenças de nível comportamental, deslocamentos posicionais, diferenças de tamanho, problemas de conteúdo e problemas de aparência visual. Para comparar com *CrossCheck*, *CrossT* e *WebDiff* os autores (CHOUDHARY et al., 2012) executaram todos os três modelos nos mesmos aplicativos *Web*, confirmaram manualmente as diferenças relatadas pelas ferramentas e compararam os resultados. O *CrossCheck* superou o *CrossT* e o *WebDiff*, pois identificou um número maior de *XBIs* e um número menor de falsos positivos (CHOUDHARY et al., 2012).

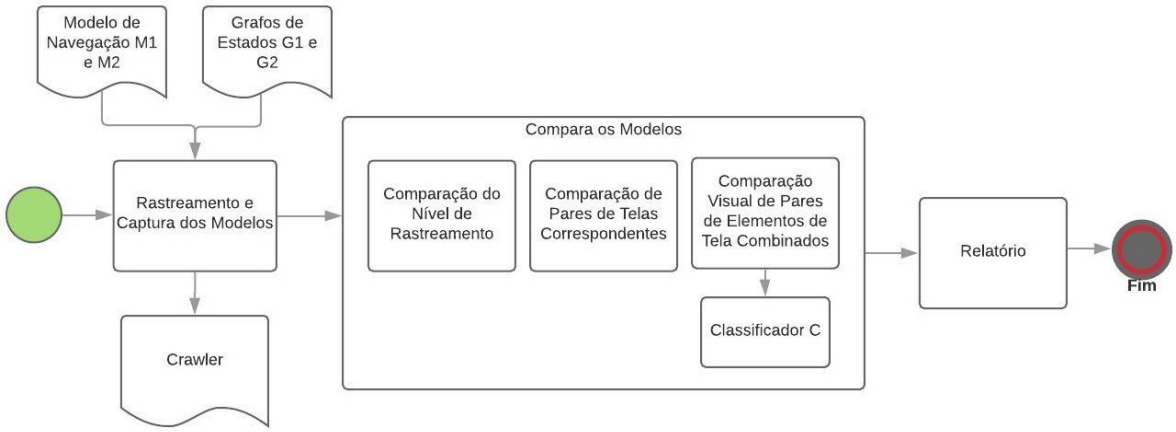


**Figura 2.7**: Processo de Identificação de *XBIs* pelo *CrossCheck*.

### 2.3.3 X-PERT

O *X-Pert* é um modelo de identificação de *XBIs* desenvolvida em *Java* que consiste em três módulos conforme a Figura 2.8.

O primeiro módulo (Rastreamento e Captura dos Modelos) é semelhante a etapa de captura do modelo *CrossCheck*. O segundo módulo (Compara os Modelos) é organizado com um conjunto de quatro algoritmos, cada um direcionado para detectar uma categoria específica de *XBIs*: comportamento, estrutura, conteúdo visual e conteúdo de texto (CHOUDHARY et al., 2013). Durante a detecção de *XBIs* de conteúdo visual o *X-Pert* captura as imagens da tela dos dois elementos correspondentes e compara seus histogramas de cores usando a distância $\chi^2$ similar ao *CrossCheck* (CHOUDHARY et al., 2013).

O *X-Pert* consiste em uma evolução do *CrossCheck*, e o modelo geralmente reporta apenas um *XBI* por erro real, ao contrário do *CrossCheck*, que normalmente produz vários relatórios de erros duplicados. Por exemplo, o movimento de um único elemento em uma página pode ter um efeito dominó nas posições de todos os elementos abaixo dela. O *CrossCheck* pode relatar todos esses elementos como *XBIs* diferentes, enquanto o *X-Pert* identifica apenas o elemento ofensivo. Essa melhoria simplifica bastante a tarefa de entender os *XBIs* para o desenvolvedor (CHOUDHARY et al., 2013).

Além disso, os algoritmos para detecção *XBI* de comportamento, conteúdo visual e conteúdo de texto são adaptados do *CrossCheck*, mas orquestrados de maneira diferente e mais eficaz no trabalho atual. O algoritmo para detectar *XBIs* de estrutura, que geralmente constituem a maior parte dos *XBIs*, é completamente novo e uma melhoria substancial em relação ao *CrossCheck* (CHOUDHARY et al., 2013).

Para detectar *XBIs* estruturais, o *X-Pert* executa as seguintes etapas: extrair grafos de alinhamentos e compara-los. Após extrair os grafos de alinhamento, o modelo verifica os dois grafos quanto à equivalência, qualquer diferença encontrada constitui um *XBI* estrutural (CHOUDHARY et al., 2013). Correspondência DOM: Para realizar a comparação de duas páginas, o algoritmo calcula uma lista de pares de elementos DOM correspondentes. Isso é feito com base em uma métrica de índice de correspondência. Esta métrica foi proposta pela primeira vez no *WebDiff* e mais desenvolvida no *CrossCheck* (CHOUDHARY et al., 2013). Ao contrário do *CrossCheck* que comparou todos os elementos DOM e gerou muitos falsos positivos, o *X-Pert* aplica a comparação visual apenas aos elementos DOM folha, onde é mais eficaz na detecção de *XBIs* de conteúdo visual (CHOUDHARY et al., 2013).

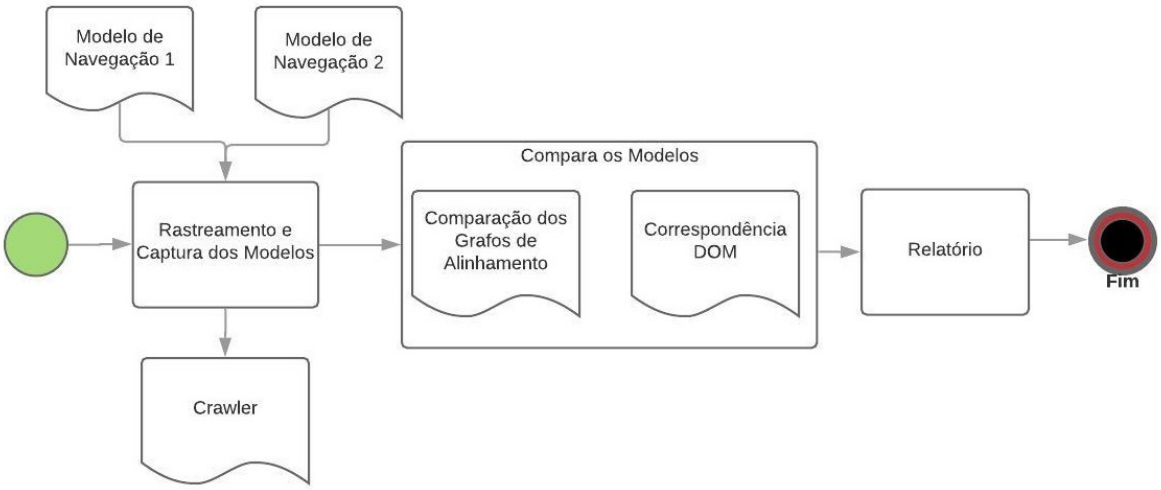


**Figura 2.8**: Processo de Identificação de *XBIs* pelo *X-Pert*.

O *X-Pert* demonstrou um alto índice de precisão na identificação de *XBIs*. Os problemas detectados incluíram todos os quatro tipos de *XBIs*, com uma prevalência de *XBIs* de estrutura, seguida por *XBIs* de comportamento e *XBIs* de conteúdo. O *X-Pert* é de fato efetivo em encontrar *XBIs* e superou os resultados obtidos pelo *CrossCheck* tanto em termos de identificação de *XBIs* quanto na redução do número de falsos positivos (CHOUDHARY et al., 2013).

### 2.3.4 BROWSERBITE

A arquitetura do *BrowserBite* é composta por três módulos principais: captura, comparação de capturas de tela e classificação, conforme apresentado na Figura 2.9. As páginas *Web* são abertas e renderizadas usando o *Selenium* [4]. O módulo de captura de tela é capaz de

[4]https://www.selenium.dev/

capturar a tela da página *Web* inteira, por meio de Máquinas Virtuais que são capazes de tirar *screenshots* de páginas *Web* em uma determinada configuração (SEMENENKO et al., 2013).

O módulo de comparação de capturas de tela se baseia nas técnicas de segmentação e comparação de imagens. Sua entrada é uma coleção de imagens, uma dessas imagens é designada pelo usuário como a imagem de referência, enquanto outras imagens são chamadas de imagens sob teste (IUT). A imagem de referência deve corresponder a uma renderização correta da página *Web*, assim, as imagens sob teste são comparadas com as imagens de referência (SEMENENKO et al., 2013).

Cada imagem é segmentada em regiões retangulares menores chamadas regiões de interesse (*ROI*), com base nas bordas e nas mudanças de cor na imagem. O conjunto de ROIs extraídos da imagem de referência é correlacionado com o conjunto de ROIs extraídas de cada uma das imagens IUT, ou seja, um ROI é mapeados até a identificação de um outro ROI correspondente. A ferramenta extrai dos ROIs correlacionados a um índice de similaridade. Os pares de ROIs são declarados compatíveis se sua similaridade estiver acima de um certo limite. Caso contrário, o *Browserbite* reporta o par como uma possível incompatibilidade por meio de relatórios (SEMENENKO et al., 2013).

O *BrowserBite* utiliza também um módulo de classificação que faz uso de algoritmos de aprendizado de máquina para aumentar a eficácia na identificação de *XBIs*. O objetivo do módulo é classificar possíveis incompatibilidades relatadas em duas categorias: positivos verdadeiros (a incompatibilidade que é percebida pelo usuário) e falsos positivos (a incompatibilidade que não é percebida pelo usuário) (SEMENENKO et al., 2013). As técnicas de aprendizado de máquina utilizadas para classificação são: árvores de decisão e redes neurais artificiais.

No trabalho (SEMENENKO et al., 2013) as conclusões apresentaram que as redes neurais superam os resultados das árvores de decisão. Os resultados sugerem que o *BrowserBite* aprimorado com um módulo de classificação baseado na rede neural alcança alta precisão em relação às técnicas de ponta. Estes resultados são uma melhoria significativa no que diz respeito ao estado da arte. O *BrowserBite* apresentou uma precisão maior do que a do *CrossCheck* (CHOUDHARY et al., 2012), e seu predecessor *WebDiff*. Esta última avaliação do *CrossCheck* e *WebDiff* é focada em falsos positivos (falsos negativos não são reportados).

### 2.3.5 PAES, WATANABE (2018)

No artigo (PAES; WATANABE, 2018) os autores propõem um modelo que apresenta a tarefa de detectar *XBIs* como um problema de classificação. O modelo proposto segmenta um

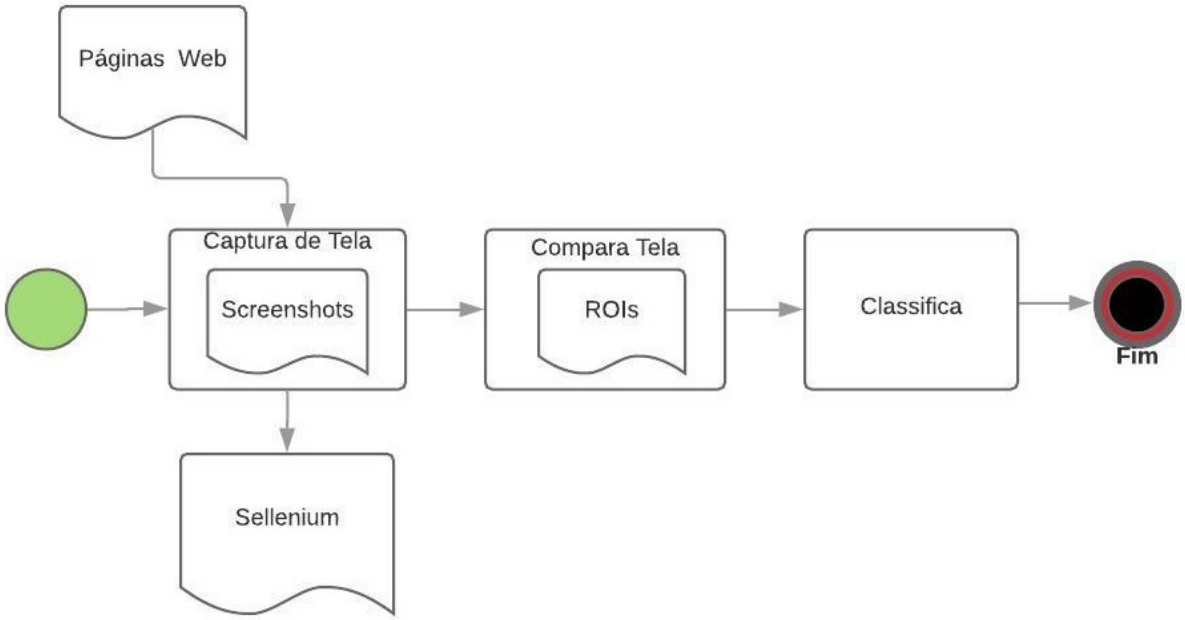


**Figura 2.9**: Processo de Identificação de *XBIs* pelo *BrowserBite*.

aplicativo *Web* de acordo com a sua estrutura *DOM*, assim como a etapa (Coleta de dados) do *WebDiff*. O modelo de avaliação utilizou algoritmo de aprendizado de máquina (Árvore de Decisão) que realizou testes sobre o classificador e coletou métricas da efetividade dos resultados.

As características que foram consideradas neste estudo são: posição, altura, largura, captura de tela dos elementos DOM e a janela de visão em que o aplicativo da *Web* foi renderizado. Essas características foram combinadas para compor o conjunto de recursos, comparando-as entre navegadores alvo e de referência (PAES; WATANABE, 2018). Os autores executaram uma etapa na qual removeram visualmente todos os elementos que não estavam sendo analisados. Todos os elementos internos ao elemento que estava sendo analisado foram retirados visualmente da página antes da execução da captura de tela. Essa etapa foi conduzida a fim de aumentar a precisão do modelo, ao compará-lo aos trabalhos relacionados.

A eficácia na detecção de *XBIs* do modelo (PAES; WATANABE, 2018), foi medida com as seguintes métricas: **Precisão:** representa a proporção de positivos previstos (*XBIs* identificados no modelo de classificação) que eram *XBIs* reais; ***Recall* ou Sensibilidade:** representa a proporção de todos os positivos reais (o conjunto completo de *XBIs*) que foram corretamente identificados pelo modelo; **Medida F:** é calculada como um valor médio entre a precisão e o *Recall*.

A implementação do algoritmo de árvore de decisão C5.0 [5] foi utilizada no modelo de

[5]https://cran.r-project.org/web/packages/C50/index.html

classificação. O modelo que apresentou os melhores resultados para a métrica de Precisão foi a C5.0 com 20 iterações. O classificador que apresentou os melhores resultados da métrica de *Recall* foi o C5.0 com 5 iterações. O modelo de classificação que apresentou os melhores resultados para a métrica de Medida F foi a C5.0 com 20 iterações. Em relação a estes resultados, o modelo do classificador que apresentou os melhores resultados globais foi a configuração C5.0 com 20 iterações.

## 2.4 CARACTERÍSTICAS UTILIZADAS PELOS MODELOS

Os modelos apresentados são soluções existentes para problemas no processo de identificação de *XBIs* como; o alto custo do processo manual e as inconsistências nas identificações de *XBIs*. Esses modelos executam um processo em comum de coleta de dados, análise de dados e relato dos erros, mas apresentam técnicas distintas. O foco desse trabalho é explorar os modelos baseados em aprendizado de máquina, com a intenção de propor um conjunto de características mais eficaz, que identifique um menor número de falsos positivos e negativos.

O *CrossCheck* (CHOUDHARY et al., 2012), o *BrowserBite* (SEMENENKO et al., 2013) e o modelo Paes, Watanabe(2018) (PAES; WATANABE, 2018), são os modelos mais atuais e que utilizam aprendizado de máquina para realizar a classificação dos *XBIs* afim de aumentar a efetividade dos resultados.

Apesar da ferramenta *X-Pert* (2013) apresentar resultados satisfatórios na identificação de *XBIs* e utilizar aprendizado de máquina para a classificação dos mesmos, optou-se pela utilização do *CrossCheck* no experimento devido a semelhança do módulo de identificação de *XBIs* de *layout*, o que simplificou o trabalho sem perdas nos resultados.

O *CrossCheck* usa aprendizado de máquina para construir um classificador que é usado para decidir se dois elementos de tela comparados são diferentes. No trabalho (CHOUDHARY et al., 2012) foi utilizado um classificador de árvore de decisão. O *CrossCheck* utilizou um conjunto de cinco características para modelar o classificador:

- ***Taxa de diferença de tamanho:*** essa característica foi projetada para detectar diferenças de tamanho entre os dois elementos em questão. A característica é calculada como uma proporção para normalizar e remover os efeitos de pequenas discrepâncias no tamanho, decorrentes de diferenças de espaço em branco ou de preenchimento entre os navegadores;

- ***Deslocamento:*** essa característica captura a distância euclidiana entre as posições dos ele-

mentos de tela correspondentes;

- ***Área:*** essa característica é calculada como a diferença entre (altura * largura) dos elementos correspondentes. Dessa forma, o *CrossCheck* pode ignorar diferenças de tamanho ou posição em elementos realmente pequenos, que são tipicamente o resultado do ruído na captura de dados, em vez da manifestação de um *XBI* real;

- ***Diferenças de texto nos elementos DOM folha:*** essa é uma característica de valor *booleano* que detecta diferenças no conteúdo de texto dos elementos da tela. Esse recurso é avaliado como *true* se, e somente se, os elementos que estão sendo comparados forem elementos folha de suas respectivas árvores DOM, se esses contiverem texto e o texto nos dois nós for diferente. Caso contrário, esse recurso recebe o valor *false*;

- $\chi^2$ ***Distância da imagem:*** essa característica é o árbitro final da igualdade na comparação dos elementos da tela. Para este recurso, as imagens dos respectivos elementos da tela são comparadas calculando seus histogramas de imagem e a distância $\chi^2$ entre eles.

O *BrowserBite* utiliza as seguintes 17 características para construir o modelo de classificação:

- ***10 histogram bins:*** (h0, h1,... h9). Esses 10 inteiros codificam o histograma da imagem do ROI. 10 *bins* representam a distribuição da intensidade do pixel em toda a imagem do ROI;

- ***Correlação entre o ROI na imagem referência e o ROI no IUT:*** Este é um número entre zero e um. Está próximo de zero em caso de correlação muito baixa entre ROIs. É zero no caso de uma imagem ausente ou adicional;

- ***Posição horizontal e vertical do ROI:*** (coordenadas X e Y) da imagem da linha de base;

- ***Tamanho horizontal e vertical do ROI:*** (largura e altura) da imagem da linha de base;

- ***Índice de configuração:*** um identificador numérico da combinação navegador-plataforma do IUT. *Browserbite* suporta 14 combinações de plataformas de navegadores, portanto, este é um inteiro entre 1 e 14;

- ***Densidade de incompatibilidade:*** MD = E / T, onde E é o número de ROIs nas imagens de testes (IUT) que não correspondem a um ROI na imagem de referência e T é o número total de ROIs. Ao construir os modelos de aprendizado de máquina, o MD é uma característica de cada ROI.

No artigo (PAES; WATANABE, 2018) o algoritmo de árvore de decisão foi configurado para construir os seguintes modelos para os dados: uma árvore de decisão não podada, uma árvore de decisão *winnowed* e árvore de decisão *winnowed*, com 5, 10 e 20 iterações de reforço. A implementação do algoritmo de árvore de decisão C5.0 foi utilizada. Os *scripts* que executaram o experimento e todas as análises estatísticas relatadas foram implementados na linguagem R.

O problema de classificação elaborado no artigo (PAES; WATANABE, 2018) é modelado com as seguintes características:

- ***Diferenças de posição:*** calcula a diferença entre a posição superior esquerda e direita do mesmo elemento DOM capturado no navegador de referência e um navegador alvo. O cálculo leva em consideração a posição do elemento em relação a janela de visão dos navegadores e em relação aos elementos pai e filhos. Valores de diferenças de posição próximas de zero indicam que a posição entre o navegador de referência e o navegador alvo não representam um *XBI* externo. Enquanto que, valores mais altos nesses recursos indicam diferença na posição do elemento DOM no navegador alvo em comparação ao navegador referência. Portanto, possivelmente indicando um *XBI* externo;

- ***Visibilidade da janela de visualização:*** consiste em dois valores numéricos para representar se um elemento está contido no navegador referência e no navegador alvo na janela de visualização do aplicativo *Web*, de acordo com as margens esquerda e direita da página. Os valores numéricos podem ser usados para analisar se esses elementos são renderizados de maneira semelhante nos navegadores. Valores próximos a 0 indicam que os elementos estão posicionados de maneira semelhante em relação às margens da janela de exibição da página;

- ***Diferenças de tamanho:*** o cálculo recebe a diferença de altura do elemento DOM do navegador alvo e referência, essa diferença então é dividida pela altura do elemento DOM no navegador referência. O mesmo cálculo é feito com os valores da largura e da área dos elementos DOM. A diferença entre os navegadores alvo e de referência são calculadas. Valores de diferenças de tamanho próximos a zero indicam que os elementos DOM renderizados no navegador referência e alvo apresentam dimensões semelhantes, portanto, não representam um *XBI* externo. Valores mais altos nesses recursos, por outro lado, possivelmente indicam que o mesmo elemento DOM foi renderizado no navegador alvo e referência, possivelmente indicando um *XBI* externo;

- ***Comparação de tela:*** diferenças entre a captura de tela dos elementos DOM capturados no

navegador de referência e alvo. As diferenças de captura de tela foram calculadas de acordo com as métricas:

- Diferença de imagem: a distância média de cor para cada pixel das capturas de tela;

- $\chi^2$ distância dos histogramas de tela: métricas de comparação de tela foram usadas como recursos para identificar *XBIs* internos, associados a mudanças no texto, fundo, efeitos visuais, entre outras características de um elemento DOM.

- Distância de *pHash* das capturas de tela, que realizam comparação de imagens usando a transformação de cosseno para baixas frequências da imagem para gerar um valor de *hash* para as imagens e o algoritmo de distância de *Hamming* para calcular diferenças entre os valores de *hash*, da mesma forma que a comparação de imagens realizada em (SANCHEZ; AQUINO, 2015).

As métricas foram utilizadas de forma complementar, para aumentar a eficácia na detecção de *XBIs* internos.

- ***Complexidade do elemento DOM:*** consiste em três valores para alavancar a complexidade do elemento DOM que foi analisado pelo algoritmo. Os seguintes recursos também foram modelados:

  - número de elementos filhos no navegador de referência;
  - número de caracteres no texto do elemento no navegador de referência;
  - a área do elemento DOM maior, calculada pela seguinte fórmula:

  **Área = max (Wbase x Hbase, Wtarget x Htarget)**

  onde Área é o valor calculado usado como um recurso; *Wbase* e *Wtarget* são os valores de largura do elemento DOM conforme renderizados no navegador de referência e alvo; e *Hbase* e *Htarget* são os valores de altura do elemento DOM conforme renderizados no navegador de referência e alvo.

A Figura 2.10 destaca um resumo das características utilizadas por cada um dos modelos que utilizam o aprendizado de máquina, além de apresentar o diferencial de cada modelo e quais tipos de *XBIs* as características são capazes de identificar.

O *script* que realizou a extração das características está disponibilizado ao público e pode ser acessado no repositório [6].

[6] https://github.com/daianyyy18/Mestrado/

| Modelo | Diferencial | Característica | Tipo de XBI |
|---|---|---|---|
| CrossCheck | Aprendizado de Máquina | Taxa de diferença de tamanho | Estrutural |
| | | Deslocamento | Estrutural |
| | | Área | Estrutural |
| | | Diferenças de texto nos elementos DOM folha | Conteúdo |
| | | Distância da imagem | Estrutural |
| BrowserBite | Aprendizado de Máquina e Segmentação ROI | 10 *histogram bins* | Estrutural |
| | | Correlação entre o ROI na imagem referência e o ROI no IUT | Estrutural |
| | | Posição horizontal e vertical do ROI | Estrutural |
| | | Tamanho horizontal e vertical da ROI | Estrutural |
| | | Índice de configuração | - |
| | | Densidade de incompatibilidade | Estrutural |
| Paes, Watanabe | Aprendizado de Máquina e Estratégia Crawler | Diferenças de posição | Estrutural |
| | | Visibilidade da janela de visualização | Estrutural |
| | | Diferenças de tamanho | Estrutural |
| | | Comparação de tela | Estrutural |
| | | Complexidade do elemento DOM | Estrutural |

**Figura 2.10**: Características dos modelos de classificação de *XBIs*

# 3 AVALIAÇÃO

Para a realização desse trabalho [1] foi conduzido um experimento com objetivo de comparar os resultados obtidos pelos modelos *CrossCheck* (2012), *Browserbite* (2013) e Paes, Watanabe (2018), e de analisar qual deles conseguiu atingir os melhores resultados para as métricas de Precisão, *Recall* e Medida F. O experimento foi conduzido utilizando algoritmos de aprendizado de máquina (árvore de decisão), aplicando os modelos em um ambiente real (*Desktop*) e levantando dados para responder a seguinte questão de pesquisa:

***Q1: Qual modelo dentre os selecionados geram melhores resultados na identificação automatizada de XBIs?***

A partir da questão de pesquisa foram levantadas as seguintes hipóteses:

***$H_0$ - Não existem diferenças significativas entre os modelos de classificação automática de XBIs.***

***$H_1$ - As abordagens apresentam diferenças significativas entre os modelos de classificação automática de XBIs.***

**Variáveis Independentes:** foram identificadas como sendo as características extraídas das páginas *Web* para cada modelo de classificação automática de *XBIs* utilizados nesse experimento.

**Variáveis Dependentes:** foram identificadas como a classificação de cada elemento do *dataset* caracterizando um *XBI* ou não.

A metodologia do experimento dividiu-se em quatro atividades sequenciais:

**(1) Aquisição de dados:** executando um *Crawler* foi realizada uma varredura em 35 aplicativos *Web*, apresentados na Tabela 3.1, selecionados para os testes. Os aplicativos *Web* foram selecionados com base no artigo (PAES; WATANABE, 2018). Essa etapa coletou dados referentes aos elementos DOM das páginas em relação a posição, altura, largura, captura de tela dos elementos DOM e a janela de visão em que o aplicativo *Web* foi renderizado. Para conduzir a atividade de aquisição de dados, foi utilizado um aplicativo de *Crawler open source*. O aplicativo de *Crawler* [2] quando executado, abre automaticamente o aplicativo *Web* em diferentes

[1] https://github.com/daianyyy18/Mestrado/
[2] https://github.com/watinha/collector

navegadores e coleta as propriedades dos elementos DOM para cada navegador configurado. Para esse experimento, o *Crawler* foi configurado para utilizar os seguintes navegadores: *Google Chrome* e o *Internet Explorer*. Todos os mecanismos do navegador foram executados no sistema operacional *Windows 10 Professional*. Vale ressaltar que o *Crawler* utilizado nesse trabalho foi o mesmo utilizado no trabalho (PAES; WATANABE, 2018), portanto não houve o esforço para implementá-lo.

| **Sites Utilizados nos Testes** |
|---|
| http://wwatana.be/mobile/mobile/201602/04/1.html |
| http://wwatana.be/mobile/mobile/201602/01/1.html |
| http://wwatana.be/mobile/mobile/201602/09/1.html |
| http://wwatana.be/mobile/mobile/201602/13/1.html |
| http://wwatana.be/mobile/mobile/201602/15/1.html |
| http://wwatana.be/mobile/mobile/201602/16/1.html |
| http://wwatana.be/mobile/mobile/201602/17/1.html |
| http://wwatana.be/mobile/mobile/201701/01/1.html |
| http://wwatana.be/mobile/mobile/201701/18/1.html |
| http://wwatana.be/mobile/mobile/201702/01/index.html |
| http://wwatana.be/mobile/mobile/201702/15/index.html |
| http://wwatana.be/mobile/mobile/201801/1/ |
| http://wwatana.be/mobile/mobile/201801/11/ |
| http://wwatana.be/mobile/mobile/201801/12/ |
| http://wwatana.be/mobile/mobile/201801/2/ |
| http://wwatana.be/mobile/mobile/201801/3/ |
| http://wwatana.be/mobile/mobile/201801/4/ |
| http://wwatana.be/mobile/mobile/201801/5/ |
| http://wwatana.be/mobile/mobile/201801/6/ |
| http://wwatana.be/mobile/mobile/201801/9/ |
| http://wwatana.be/mobile/mobile/201802/3/ |
| http://wwatana.be/mobile/topsites/www.google.com/ |
| http://wwatana.be/mobile/topsites/www.blogspot.com/ |
| http://wwatana.be/mobile/topsites/www.linkedin.com/ |
| http://wwatana.be/mobile/topsites/www.wikipedia.org/ |
| http://wwatana.be/mobile/topsites/www.amazon.com/ |
| http://wwatana.be/mobile/topsites/www.live.com/ |
| http://wwatana.be/mobile/topsites/m.youtube.com/ |
| http://wwatana.be/mobile/topsites/m.facebook.com/ |
| http://wwatana.be/mobile/topsites/h5.m.taobao.com/ |
| http://wwatana.be/mobile/topsites/m.360.cn/ |
| http://wwatana.be/mobile/topsites/m.twitch.tv/ |
| http://wwatana.be/mobile/topsites/mail.ru/ |
| http://wwatana.be/mobile/topsites/m.ebay.com/ |
| http://wwatana.be/mobile/topsites/m.aliexpress.com/ |

**Tabela 3.1**: Lista de páginas *Web* utilizadas nos testes

**(2) Classificação manual:** consiste da análise de cada elemento DOM do par de navegadores (considerando o navegador referência e o navegador alvo). Como resultado dessa classificação, foi obtido um *dataset*, com uma linha para cada elemento DOM capturado, sendo cada linha classifica o elemento DOM como um *XBIs* ou não. O *dataset* apresentou 3.314 elementos, e foram classificados 900 elementos como *XBIs*.

**(3) Execução dos modelos:** utilizando os mesmos aplicativos *Web* selecionados e os mesmos navegadores, os modelos *CrossCheck*(2012), *Browserbite*(2013) e Paes, Watanabe (2018) foram aplicadas. No experimento, foi utilizado um *script* desenvolvido na linguagem R para a implementação da árvore de decisão C5.0 em conjunto com as características de cada modelo apresentadas na Tabela 3.1

**(4)Análise dos Dados:** No experimento, foram analisados os algoritmos de aprendizado de máquina na detecção de *XBIs*. Uma maneira comum de analisar os resultados dos experimentos envolvendo algoritmos de aprendizado de máquina é usar as métricas de Precisão, *Recall* e Medida F. Para a utilização das métricas, é necessário identificar a qual grupo de classificação um *XBI* pertence. São elas (POWERS, 2011):

- Verdadeiro Positivo (VP): significa uma classificação correta de um *XBI* real. Por exemplo, quando os elementos representam um *XBI* e o modelo os classifica como *XBI*;
- Verdadeiro Negativo (VN): significa uma classificação correta de um não *XBI*. Por exemplo, quando os elementos não representam um *XBI* e o modelo não os classifica como *XBI*;
- Falso Positivo (FP): significa uma classificação errada de um *XBI* que não existe. Por exemplo, quando os elementos não representam um *XBI* e o modelo os classifica como *XBI*;]
- Falso Negativo (FN): significa uma classificação errada de um *XBI* real. Por exemplo, quando os elementos representam um *XBI* e o modelo não os classifica como *XBI*.

A métrica de Precisão nos resultados do experimento forneceram informações sobre qual dos modelos conseguiu ser mais assertivo em identificar *XBIs* reais em relação aos conjunto *XBIs* previstos (*XBIs* identificados nos testes manuais).

***Precisão = VP/(VP+FP)***

O valor do *Recall* que representa a proporção de todos os positivos reais (o conjunto completo de *XBIs*) que foram corretamente identificados pelo modelo; nos traz a informação de

qual modelo identificou um maior número de *XBIs* corretamente, mesmo que esses não tenham sido identificados nos testes manuais.

***Recall = VP/ (VP-FN)***

A métrica da Medida F apresenta uma média harmônica entre Precisão e Recall, como essa medida é uma média ela apresenta um resultado mais assertivo do classificador em relação as outras métricas.

***Medida F = 2*((Precisão * Recall)/(Precisão + Recall))***

Para a realização desse trabalho, foi executado um experimento com a intenção de explorar as características utilizadas nos modelos *CrossCheck* (2012), *BrowserBite* (2013) e Paes, Watanabe (2018), e verificar qual o modelo obtém o melhor resultado para a métrica da Medida F. Esse experimento fornecerá informações importantes para trabalhos futuros que tenham como objetivo a evolução das técnicas de identificação automática de *XBIs*.

O experimento é voltado à identificação de *XBIs* de *layout* e a ferramenta *CrossCheck* executa essa identificação na etapa "Comparação Visual de Pares de Elementos de Tela Combinados".

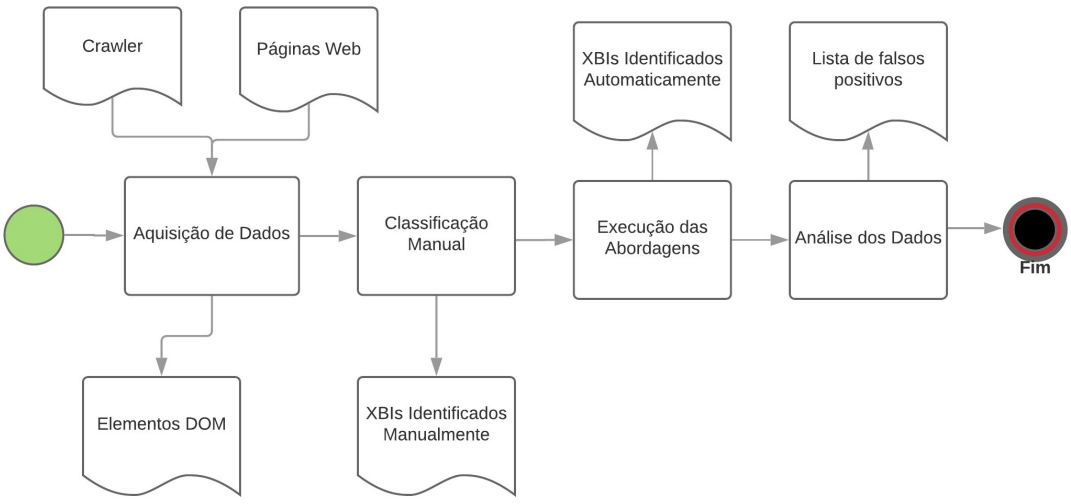


**Figura 3.1**: Metodologia do experimento

## 3.1 RESULTADOS E DISCUSSÕES

A Média e o Desvio Padrão (DP) das métricas: Precisão, *Recall* e Medida F observados no procedimento de validação cruzada de 10-fold são apresentados na Figura 3.2. E também podem ser analisados nos gráficos de caixas nas Figuras 3.3, 3.4 e 3.5.

| | Precisão | | Recall | | Medida F | |
|---|---|---|---|---|---|---|
| | **Média** | **DP** | **Média** | **DP** | **Média** | **DP** |
| **C5.0 - Paes, Watanabe** | 0.71049 | 0.18252 | 0.537882 | 0.15959 | 0.59598 | 0.14933 |
| **C5.0 - CrossCheck** | 0.76916 | 0.24472 | 0.447602 | 0.25002 | 0.52282 | 0.24050 |
| **C5.0 - Browserbite** | 0.63815 | 0.26071 | 0.521336 | 0.25897 | 0.51232 | 0.21829 |
| **C5.0 10 iterações - Paes, Watanabe** | 0.78332 | 0.16939 | 0.526993 | 0.17463 | 0.61104 | 0.15031 |
| **C5.0 10 iterações - CrossCheck** | 0.74105 | 0.24043 | 0.395996 | 0.23569 | 0.48599 | 0.21479 |
| **C5.0 10 iterações - Browserbite** | 0.75061 | 0.22706 | 0.411838 | 0.25202 | 0.46790 | 0.24215 |
| **C5.0 20 iterações - Paes, Watanabe** | 0.74790 | 0.15460 | 0.53445 | 0.17443 | 0.60764 | 0.14500 |
| **C5.0 20 iterações - CrossCheck** | 0.79246 | 0.24336 | 0.39632 | 0.24907 | 0.48710 | 0.24151 |
| **C5.0 20 iterações - Browserbite** | 0.71572 | 0.25424 | 0.37038 | 0.23285 | 0.41812 | 0.20923 |

**Figura 3.2**: Resultados obtidos pelas métricas de eficácia.

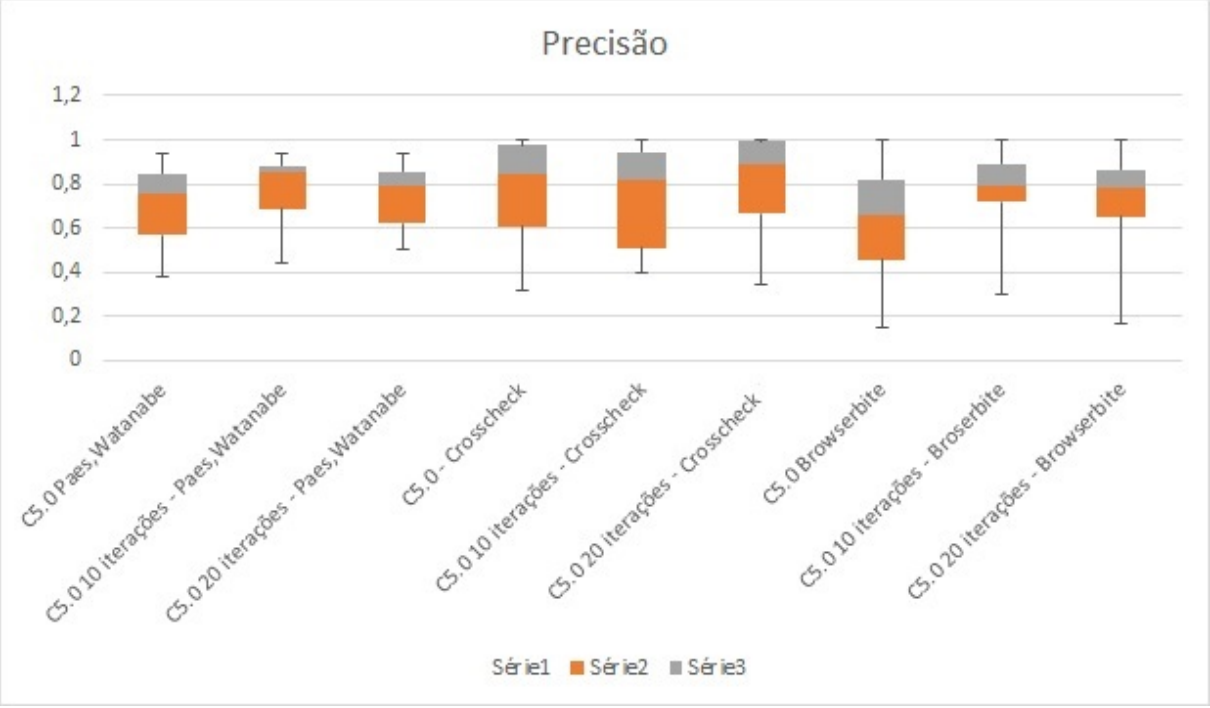


**Figura 3.3**: Resultados obtidos pelas métricas de Precisão.

Para realizar os testes estatísticos dos resultados foi utilizado a Análise de Variância (Anova). Para os resultados de Precisão, *Recall* e Medida F da configuração de classificação C5.0 para cada validação cruzada com 10-fold.

Os resultados médios da Medida F na validação cruzada com 10-fold para o modelo

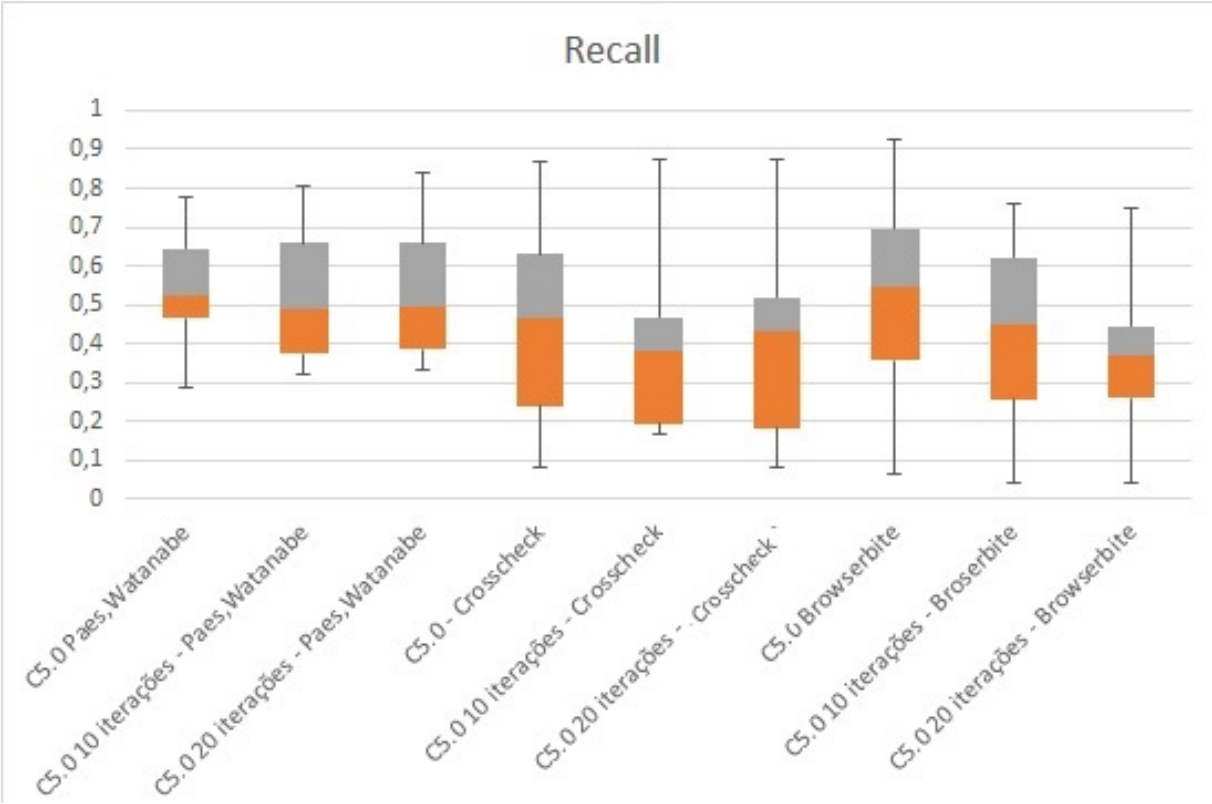


**Figura 3.4**: Resultados obtidos pelas métricas de Recall.

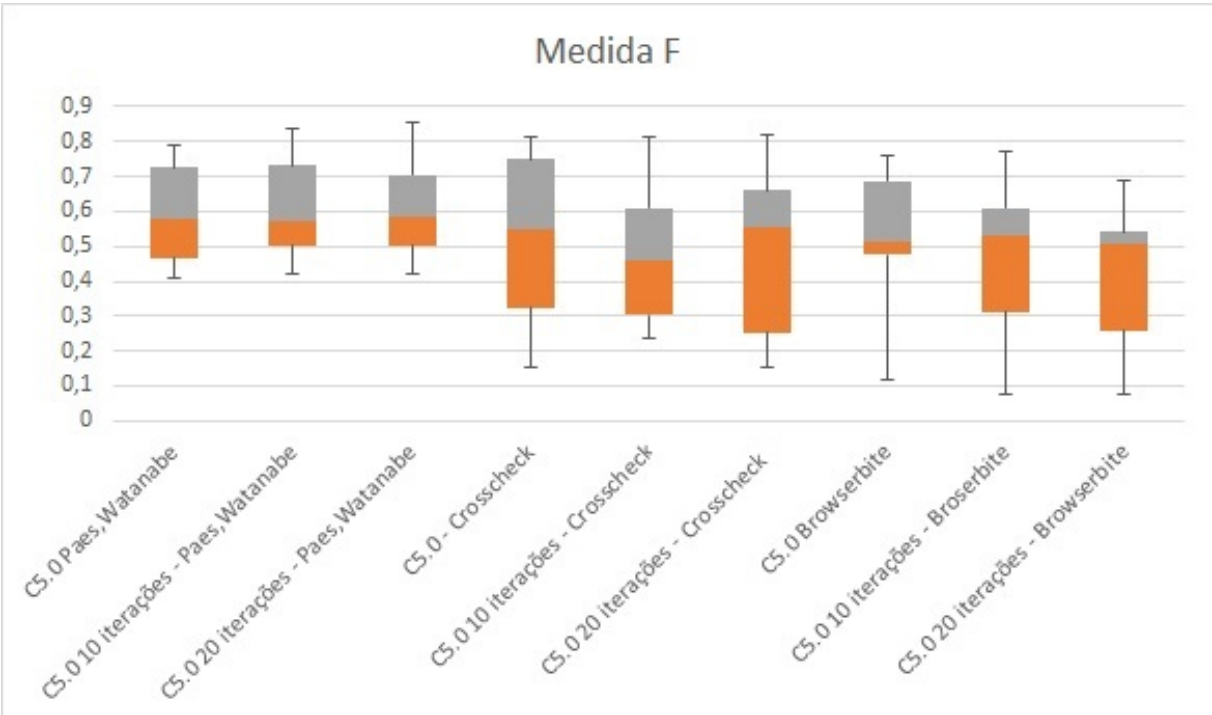


**Figura 3.5**: Resultados obtidos pelas métricas de Medida F.

Paes, Watanabe variaram de 0.59598 (C5.0 - Paes, Watanabe) a 0.61104 (C5.0 10 iterações - Paes, Watanabe).

Os resultados médios da Medida F para o modelo *CrossCheck* variaram de 0.48599 (C5.0 10 iterações - *CrossCheck*) a 0.52282 (C5.0 - *CrossCheck*).

Os resultados médios da Medida F para o modelo *Broserbite* variaram de 0.41812 (C5.0 20 iterações - *Browserbite*) a 0.51232 (C5.0 - *Browserbite*).

Em relação à métrica da medida F, valores mais altos implicam maior eficácia na detecção de incompatibilidades entre navegadores, e a configuração C5.0 10 iterações - Paes, Watanabe obteve melhor resultado nesse experimento.

Embora o modelo Paes, Watanabe(2018) tenha apresentado a maior média na Medida F, os valores Anova não apontaram diferenças significantes, com F= 1.102 e o P-value=0.371.

Os resultados médios de Precisão do modelo Paes, Watanabe(2018) na validação cruzada com 10-fold variaram de 0.71049 no pior cenário de configuração (C5.0 - Paes, Watanabe) a 0.78332 no melhor cenário de configuração (C5.0 10 iterações - Paes, Watanabe).

Os resultados médios de Precisão do modelo de classificação do *CrossCheck* variaram de 0.79246 (C5.0 20 iterações - *CrossCheck*) a 0.74105 (C5.0 10 iterações - *CrossCheck*).

Os resultados médios de Precisão do modelo de classificação do *Browserbite* variaram de 0.75061 (C5.0 10 iterações - *Browserbite*) a 0.63815 (C5.0 - *Browserbite*).

Em relação a métrica de Precisão, valores mais altos implicam um número menor de identificação de falso-positivos, e a configuração C5.0 20 iterações - *CrossCheck* obteve o melhor resultado nesse experimento.

Embora o modelo *CrossCheck* tenha apresentado a maior média na Precisão, os valores Anova não apontaram diferenças significantes, com F= 0.44 e o P-value=0.894.

O modelo Paes, Watanabe(2018) em relação a métrica *Recall* obteve os resultados médios na validação cruzada com 10-fold, variando de 0.526993 no pior cenário de configuração (C5.0 10 iterações - Paes, Watanabe) a 0.537882 no melhor cenário de configuração (C5.0 - Paes, Watanabe).

O modelo *CrossCheck*, por outro lado, obteve os resultados médios de *Recall* de 0.395996 (C5.0 10 iterações - *CrossCheck*) a 0.447602 (C5.0 - *CrossCheck*).

Enquanto o modelo *Browserbite* obteve os resultados médios de *Recall* de 0.37038 (C5.0 20 iterações - *Browserbite*) a 0.521336 (C5.0 - *Browserbite*).

Em relação à métrica *Recall*, valores mais altos implicam um número menor de falsos negativos, e a configuração C5.0 - Paes, Watanabe obteve o melhor resultado. Significando que

esse modelo na configuração desse experimento, conseguiu identificar mais *XBIs* reais do que as outros modelos.

Embora o modelo Paes, Watanabe(2018) tenha apresentado a maior média no *Recall*, os valores Anova não apontaram diferenças significantes, com F=0.96 e o P-value=0.473.

A comparação dos resultados das métricas traz uma resposta a Q1. A eficácia da classificação foi maior para as métricas (Recall e Medida F) ao usar o modelo de classificação Paes, Watanabe (2018), nesse estudo. Enquanto que para a métrica Precisão foi maior no modelo *CrossCheck*.

Em relação a Q1, as medidas de Recall e Medida F relatadas pelo modelo de classificação Paes, Watanabe representam evidências de que esse modelo pode detectar incompatibilidades com maior eficácia, na configuração desse experimento. Essa maior eficácia deve se principalmente à menor taxa de falso negativos identificados pelo modelo. No entanto, os resultados da Precisão ainda podem ser aprimorados, contribuindo ainda mais para reduzir o custo da detecção manual dos *XBIs*. Tornando assim a hipótese $H_1$ verdadeira.

Já para os testes estatísticos Anova, embora o modelo Paes, Watanabe obteve os valores de *Recall* e Medida F superiores aos outros modelos, e o *CrossCheck* obteve a Precisão com um melhor resultado, não foi observada significância estatística. São necessários porém, mais estudos para verificar se de fato existe uma diferença estatística entre os modelos de classificação, ou não.

## 3.2 LIMITAÇÕES DO TRABALHO

Estudos do *Browserbite* (SEMENENKO et al., 2013), relatam melhores resultados com o uso de redes neurais. Como limitações desse trabalho o experimento foi realizado apenas com algoritmos de árvore de decisão. Trabalhos futuros podem estender esse estudo para diferentes modelos de classificação, objetivando uma análise mais rica em detalhes e mais precisa em seus resultados.

Outra limitação identificada foi a não utilização de um conjunto de aplicativos *Web* que representassem tipos diversificados de aplicativos existentes, apesar da utilização de uma amostra representativa.

Algumas das aplicações foram desenvolvidas em cursos de Desenvolvimento *Web*, por desenvolvedores iniciantes, onde, esses não apresentavam conhecimento ou experiência anterior em problemas de *XBIs*. Esse fato pode ter impactado o número e a variedade de *XBIs*

observados no experimento. Além disso, os 35 aplicativos *Web* replicaram o *layout* dos sites brasileiros de alto tráfego, comércio eletrônico, portais de notícias, páginas de empresa, entre outros exemplos.

Trabalhos futuros devem abordar a extensão do experimento, com mais navegadores e aplicativos *Web*, para verificar a capacidade de generalização dos resultados.

## 3.3 CONSIDERAÇÕES FINAIS

Esse capítulo apresentou um experimento com o objetivo de comparar os resultados obtidos por modelos de identificação automática de *XBIs*, que utilizam algoritmos de aprendizado de máquina. Sendo essas: *Crosscheck* (CHOUDHARY et al., 2012), *Browserbite* (SEMENENKO et al., 2013) e Paes, Watanabe (PAES; WATANABE, 2018).

A avaliação foi conduzida por um profissional da área de qualidade de software e um engenheiro de software. Durante o estudo foi desenvolvido um *dataset* com todas as informações dos elementos DOM dos aplicativos *Web* citados na Tabela 3.1. Esse *dataset* foi desenvolvido com a utilização de um *Crawler*, e também continha as informações de quais elementos consistiam em um *XBI* ou não. Esse *dataset* foi utilizado para o treinamento de algoritmo de aprendizado de máquina de redes neurais, o C5.0 utilizando o procedimento de validação cruzada de 10-fold. Na execução do algoritmo foram utilizadas as características utilizadas por cada modelo selecionado. A partir da execução do algoritmo foram coletadas as métricas de Precisão, *Recall* e Medida-F. O resultados das métricas auxiliaram na análise de qual abordagem obteve um melhor desempenho na classificação de *XBIs* com um menor número de falsos positivos e negativos.

Os resultados obtidos para as métricas de *Recall* e Medida F foram maiores para o modelo Paes, Watanabe, enquanto que para a métrica de Precisão o *Crosscheck* obteve melhor resultado na configuração realizada nesse experimento. Já os testes estatísticos não apresentaram diferenças significantes para os resultados obtidos pelos modelos.

O experimento executado nesse trabalho teve como base o trabalho (PAES; WATANABE, 2018), mas utilizou características diferentes, conforme apresentado na 2.10, e tendo como principal objetivo a comparação dos modelos selecionados.

Diferentemente do trabalho Paes, Watanabe, nesse experimento a coleta dos dados para a formação do *dataset*, não executou o passo de remoção dos elementos visuais previamente coletados. Esse passo não foi realizado devido ao alto custo operacional desse tipo de coleta. Existem aplicativos *Web* com muito elementos, e esse tipo de coleta demanda a captura de

*screenshot* individual para cada elemento que é visualmente oculto, conforme reportado no estudo (PAES; WATANABE, 2018). A não execução desse passo pode ter afetado a capacidade de detecção de *XBIs* do modelo Paes, Watanabe, no entanto permitiu uma comparação direta com os outros trabalhos relacionados.

Trabalhos futuros podem abranger um experimento com a realização de testes com um maior número de navegadores, aplicações *Web* e variabilidade de modelos de classificação utilizando outros algoritmos de aprendizado de máquina.

Esse trabalho apresenta informações relevantes para trabalhos futuros que visem evoluir técnicas de identificação automática de *XBIs*, com o objetivo de melhoria do processo de testes auxiliando as empresas a aumentarem a qualidade de seus produtos ao mesmo tempo que reduzem os custos. Além disso também contribui com q disponibilização do *dataset*, que pode ser utilizado por outros pesquisadores em outros experimentos sobre o assunto.

# 4 CONCLUSÕES

Devido ao aumento da complexidade do desenvolvimento *Web*, e o aumento do número de navegadores *Web* disponíveis, o número de inconsistências nas aplicações aumentaram (CHOUDHARY et al., 2010). Um dos tipos de inconsistências encontradas são os *XBIs*, que são as inconsistências encontradas na mesma aplicação quando executada em diferentes navegadores. Esse trabalho comparou modelos de identificação de *XBIs* que utilizam algoritmos de aprendizado de máquina. A utilização dos algoritmos de aprendizado de máquina evoluiu modelos anteriores afim de reduzir o número de falsos positivos e negativos. O objetivo desse trabalho foi identificar qual modelo entre o *CrossCheck* (CHOUDHARY et al., 2012), o *BrowserBite* (SEMENENKO et al., 2013) e Paes, Watanabe (PAES; WATANABE, 2018), conseguiu um melhor resultado na identificação de positivos reais. Para isso foi executado um estudo experimental que comparou os resultados obtidos pelos modelos de classificação automática de *XBIs*. Os resultados obtidos foram analisados de acordo com as métricas de Precisão, *Recall* e Medida F. Nesse experimento, foram utilizadas as páginas de acordo com a Tabela 3.1, as características selecionadas de acordo com a Figura 2.10, o algoritmo C5.0 com validação cruzada 10-folders. Os resultados apontaram que, nesse experimento, a abordagem Paes, Watanabe teve um melhor resultado em relação a métrica *Recall* e Medida F, enquanto o *Crosschek* teve uma melhor Precisão. Esse trabalho contribuiu com informações importantes em relação ao modelos existentes e aos seus resultados, além de contribuir com um dataset que poderá ser utilizados em trabalhos futuros. Trabalhos futuros devem abordar outros tipos algoritmos de aprendizado de máquina e um número maior de diferentes aplicativos *Web*, afim de reavaliar os resultados obtidos nesse experimento.

# REFERÊNCIAS


CHOUDHARY, S. R. Detecting cross-browser issues in web applications. In: ACM. **Proceedings of the 33rd International Conference on Software Engineering**. 2011. p. 1146–1148.

CHOUDHARY, S. R.; PRASAD, M. R.; ORSO, A. Crosscheck: Combining crawling and differencing to better detect cross-browser incompatibilities in web applications. In: IEEE. **Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on**. 2012. p. 171–180.

CHOUDHARY, S. R.; PRASAD, M. R.; ORSO, A. X-pert: accurate identification of cross-browser issues in web applications. In: IEEE. **Software Engineering (ICSE), 2013 35th International Conference on**. 2013. p. 702–711.

CHOUDHARY, S. R.; VERSEE, H.; ORSO, A. Webdiff: Automated identification of cross-browser issues in web applications. In: IEEE. **2010 IEEE International Conference on Software Maintenance**. 2010. p. 1–10.

DALLMEIER, V. et al. Webmate: a tool for testing web 2.0 applications. In: ACM. **Proceedings of the Workshop on JavaScript Tools**. 2012. p. 11–15.

HE, M. et al. X-check: a novel cross-browser testing service based on record/replay. In: IEEE. **2016 IEEE International Conference on Web Services (ICWS)**. 2016. p. 123–130.

KOHAVI, R. et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: MONTREAL, CANADA. **Ijcai**. 1995. v. 14, n. 2, p. 1137–1145.

MALLA, P. K. **Cross-browser web application testing tool**. : Google Patents, dez. 5 2017. US Patent 9,836,385.

MESBAH, A.; PRASAD, M. R. Automated cross-browser compatibility testing. In: **Proceedings of the 33rd International Conference on Software Engineering**. New York, NY, USA: ACM, 2011. (ICSE '11), p. 561–570. ISBN 978-1-4503-0445-0. Disponível em: <http://doi.acm.org/10.1145/1985793.1985870>.

PAES, F. C.; WATANABE, W. M. Layout cross-browser incompatibility detection using machine learning and dom segmentation. In: ACM. **Proceedings of the 33rd Annual ACM Symposium on Applied Computing**. 2018. p. 2159–2166.

POWERS, D. M. W. Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation. **Journal of Machine Learning Technologies**, v. 2, n. 1, p. 37–63, 2011.

QUINLAN, J. R. Induction of decision trees. **Machine learning**, Springer, v. 1, n. 1, p. 81–106, 1986.

QUINLAN, J. R. et al. Bagging, boosting, and c4. 5. In: **AAAI/IAAI, Vol. 1**. 1996. p. 725–730.

ROKACH, L.; MAIMON, O. Decision trees. In: **Data mining and knowledge discovery handbook**. : Springer, 2005. p. 165–192.

RUBNER, Y.; TOMASI, C.; GUIBAS, L. J. The earth mover's distance as a metric for image retrieval. **International journal of computer vision**, Springer, v. 40, n. 2, p. 99–121, 2000.

SANCHEZ, L.; AQUINO, P. T. Automatic deformations detection in internet interfaces: Addii. In: SPRINGER. **International Conference on Human-Computer Interaction**. 2015. p. 43–53.

SEMENENKO, N.; DUMAS, M.; SAAR, T. Browserbite: Accurate cross-browser testing via machine learning over image features. In: IEEE. **Software Maintenance (ICSM), 2013 29th IEEE International Conference on**. 2013. p. 528–531.

SILVA, L. Uma aplicação de árvores de decisão, redes neurais e knn para a identificação de modelos arma não sazonais e sazonais. **Rio de Janeiro. 145p. Tese de Doutorado-Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro**, 2005.

WEISS, S. M.; KULIKOWSKI, C. A. **Computer systems that learn: classification and prediction methods from statistics, neural nets, machine learning, and expert systems**. : Morgan Kaufmann Publishers Inc., 1991.