INVESTIGATING MODEL EXPLANATION OF BUG REPORT ASSIGNMENT RECOMMENDERS

FARJANA YEASMIN OMEE
BSc in Computer Science & Engineering, Shahjalal University of Science and Technology, Sylhet, Bangladesh, 2011

A thesis submitted
in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

INVESTIGATING MODEL EXPLANATION OF BUG REPORT ASSIGNMENT RECOMMENDERS

FARJANA YEASMIN OMEE

Date of Defence: November 22, 2022

Dr. John Anvik Thesis Supervisor	Associate Professor	Ph.D.
Dr. Wendy Osborn Thesis Examination Committee Member	Associate Professor	Ph.D.
Dr. Yllias Chali Thesis Examination Committee Member	Professor	Ph.D.
Dr. John Sheriff Chair, Thesis Examination Committee	Assistant Professor	Ph.D.

Dedication

This dissertation is dedicated to my family and close friends, especially my parents, who helped me achieve my academic goals.

Abstract

Software projects receive a lot of bug reports, and each bug report needs to be triaged. An objective of the bug report triaging process is to find an appropriate developer who can fix the reported bug. As this process can be time-consuming and requires a lot of effort, researchers have implemented recommender systems using a variety of algorithms to automate this process. Although using these recommender systems has a number of benefits, there are still many obstacles to overcome. A key obstacle is that commonly used algorithms are black-box, making it difficult for practitioners to comprehend how the models make decisions. Lack of explainability results in a lack of trust and transparency in the recommendations.

This work investigates approaches that lead to visually explainable bug report assignment recommender systems. First, we developed and compared six different recommender systems using three distinct machine learning algorithms: Random Forest (RF), MLP Classifier and Bidirectional Neural Networks (BNN) and two different feature extraction techniques: TF-IDF and Word2Vec. Second, we examine the use of WordNet to improve recommender accuracy. Third, we explore the explanation of a bug report assignment recommender using the feature-based local model LIME. Finally, we assess the use of a positive-negative horizontal bar chart, feature table, and word cloud to explain the recommender systems visually.

Our analytical analysis indicates that the optimum approach for developing a bug report assignment recommender system uses TF-IDF with RF and visually explains the recommendation with a word cloud and LIME as a local model.

Acknowledgments

First and foremost, I want to express my gratitude to Almighty Allah for providing me with the opportunity, knowledge, and strength to start and successfully complete this research.

I want to convey my profound gratitude to Dr. John Anvik, my research supervisor, for giving me the opportunity to do research and for his helpful advice during this process. It could not have been completed without his leadership, support, and commitment. Working and learning under his direction was a great privilege and honour. I am extremely grateful for what he has offered me. I also want to thank him for his friendship, understanding, and excellent sense of humour.

I also like to thank the members of my thesis committee, Dr. Wendy Osborn, Dr. Yllias Chali, and Dr. John Sheriff, for their thoughtful comments, advice, and encouragement.

In addition, I want to convey my thanks to all of the Sibyl lab members who helped me, directly or indirectly, to finish the research work.

Finally, I would like to express my sincere gratitude for my family's love and support. They are the reason I am the person I am today.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Identifying an appropriate developer who can fix a reported bug is one of the tasks of a bug-triaging process. A classification problem that uses the bug title and description as input and maps it to one of the available developers can be used to build an automatic bug-triaging technique. Researchers have proposed different recommender systems using different machine learning algorithms to improve accuracy, such as Neural Network [1], Random Forest [2], Support Vector Machines (SVM)[2] [3] [4] and Naive Bayes [2] [3] [4] [5] [6]. Most of these recommender systems used TF [3] [6] [7], TF-IDF [4] [5] [8], and Word2Vec [1] for feature extraction.

Despite recent developments in this area, decision-making based on Artificial Intelligence (AI)/Machine Learning (ML) systems requires explanations and must adhere to privacy rules. According to Article 22 of the General Data Protection Regulation (GDPR) [9] of the European Union, each algorithmic decision that impacts an individual or group must be justified. Recent research raises questions regarding the lack of software analytics in software engineering that can be explained [10]. According to the practitioners, analytical models in software engineering must be explainable [10]. When creating AI/ML-based systems, Miller [11] also contends that human factors should be considered. Consequently, it is imperative to use Explainable Software Analytics, a collection of AI/ML techniques that generate precise predictions while also being able to explain such predictions.

Researchers usually produce global explanations, which are explanations that condense the conclusions of predictions made by black-box AI/ML learning algorithms. Model-specific explanation techniques of machine learning techniques can produce such global explanations (e.g., a variable importance analysis for random forests and an ANOVA analysis for logistic regression). These methods for model interpretation have been used in earlier studies to comprehend the connection between investigated variables and results.

Such broad justifications, however, cannot support an individual model's prediction based on experimental or fictitious evidence. Serious errors could hamper the use of software analytics in industrial operations of decision and policy-making caused by a lack of explanation of the predictions made by such analytical models [10].

For determining each metric's contribution to the prediction of an instance according to a trained model, model-agnostic techniques such as LIME [12], and PyExplainer[13] have recently been proposed to explain the prediction of black-box AI/ML algorithms [14]. However, to the best of our knowledge, no formal introduction or empirical evaluation of such strategies for the bug report assignment recommender system has occurred.

To address this challenge, this work focuses on implementing an explainable bug report assignment recommender system which can recommend a developer and then create an explanation of the recommendation for a bug report. For this investigation, we first implement and compare three recommender systems using different algorithms: Random Forests and two types of neural networks, specifically Multilayer Perceptrons (MLP) and Bidirectional Neural Networks (BNN). Then we try to improve the accuracy of the recommender systems by using WordNet as a filter. At first, all these recommender systems used Word2Vec for feature selection. However, when we tried to create a feature-based local model using LIME, we discovered that using Word2Vec for the recommender system makes it challenging to explain the results. Therefore, we investigated the use of TF-IDF instead of Word2Vec for feature selection and explained the results using LIME. We also tried to implement a rule-based local model using PyExplainer. However, we could not do that since our feature set's dimensionality was incompatible with it. Finally, we compared LIME, which uses a positive-negative horizontal bar chart and feature table to explain the result, with the use of a word cloud for visualization of the explanation.

1.1 Research Questions

The following are the research questions for this work:

- RQ#1: Which classifier gives us better accuracy: Random Forest (RF), Multi-layer Perceptron (MLP) or Bidirectional Neural Network (BNN)?

- RQ#2: Which feature extraction approach gives us better accuracy: TF-IDF or Word2Vec?

- RQ#3: Does using WordNet improve the recommender accuracy?

- RQ#4: Which local model explanation approach, feature-based or rule-based, provides better explanations?

- RQ#5: Which visualization method appears to best support explanations of assignment recommendation: word cloud, positive-negative horizontal bar chart or feature table?

1.2 Contributions

This thesis makes the following contributions:

- We investigate six bug report assignment recommender systems utilizing two different feature extraction approaches, TF-IDF and Word2Vec, along with three global models, RF, MLP, and BNN.

- Examining whether or not WordNet improves recommender accuracy when used as a filter in a bug report assignment recommender system.

- Studying the use of a feature-based local model system (LIME) to explain the global models created using RF, MLP and BNN.

- Assessing the use of a word cloud, a positive-negative horizontal bar chart, and a feature table to visually explain the results of the local model.
- An analytical evaluation of three global models (RF, MLP, BNN) to determine which creates a more accurate bug report assignment recommender.
- A comparison of the accuracy and level of explanation provided by the two feature extraction techniques, TF-IDF and Word2Vec.

1.3 Outline

The rest of this thesis proceeds as follows. Chapter Two provides background information on bug reports, bug report triaging workflow, different recommender creation algorithms, feature extraction and visualization techniques. Chapter Three presents previous research on the bug report assignment recommender systems, the use of WordNet to improve recommender accuracy, explaining global models using different local models and various visualization techniques. In Chapter Four, we provide details about the two approaches we used for creating bug report assignment recommender systems and the visualization of recommendation explanation. Chapter Five presents the results of our evaluations to answer our research questions. Finally, we discuss our findings in Chapter Six before making our concluding remarks and discussing future directions of study in Chapter Seven.

Chapter 2

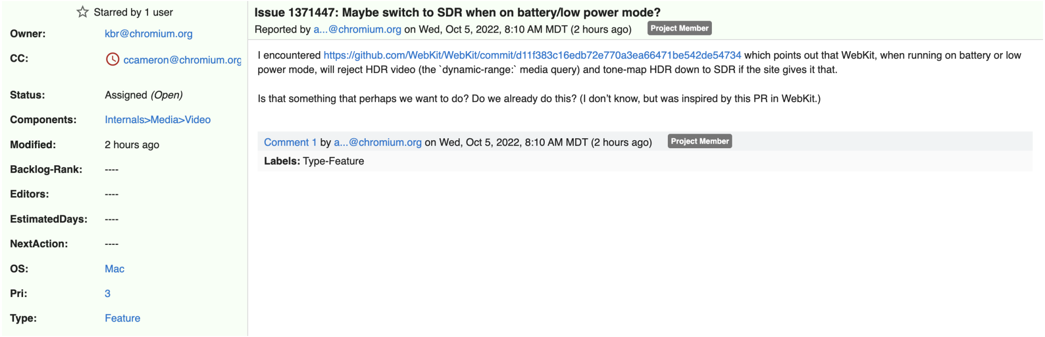
Background

This section discusses bug reports, bug report triaging workflow, different recommender creation algorithms, feature selection, explanation and visualization techniques, specifically Neural Network, Random Forest, Word2Vec, LIME and WordCloud.

2.1 Bug Reports

A bug is a mistake, fault, or failure that leads to an inaccurate or unexpected outcome in software development. Software bugs basically refer to situations when the program is not functioning as intended. A bug report, sometimes referred to as a change request or issue report, is a document that details the behaviour caused by a software defect. An effective way for users and developers to communicate is through a bug report. A bug report's identifier, a description of the issue (typically in the form of a title), instructions for reproducing the issue, report creation time, and its current status are among the various pieces of information a bug report contains. A bug report may include attachments or links to other related reports. Figure 2.1 shows an example of a bug report from Google Chrome[1] which has various information such as title, description, reported by and when, owner, comments and status.

[1] https://bugs.chromium.org/p/chromium/issues/detail?id=1371447



☆ Starred by 1 user

Owner:	kbr@chromium.org
CC:	ccameron@chromium.org
Status:	Assigned *(Open)*
Components:	Internals>Media>Video
Modified:	2 hours ago
Backlog-Rank:	----
Editors:	----
EstimatedDays:	----
NextAction:	----
OS:	Mac
Pri:	3
Type:	Feature

Issue 1371447: Maybe switch to SDR when on battery/low power mode?

Reported by a...@chromium.org on Wed, Oct 5, 2022, 8:10 AM MDT (2 hours ago) Project Member

I encountered https://github.com/WebKit/WebKit/commit/d11f383c16edb72e770a3ea66471be542de54734 which points out that WebKit, when running on battery or low power mode, will reject HDR video (the `dynamic-range:` media query) and tone-map HDR down to SDR if the site gives it that.

Is that something that perhaps we want to do? Do we already do this? (I don't know, but was inspired by this PR in WebKit.)

Comment 1 by a...@chromium.org on Wed, Oct 5, 2022, 8:10 AM MDT (2 hours ago) Project Member

Labels: Type-Feature

Figure 2.1: An example of a bug report

2.2 Bug Report Triaging Workflow

The process of screening, prioritizing, and assigning a developer to a bug report is called bug report triage. An important goal of a bug triaging procedure is to find an appropriate developer for a given bug report who may be able to solve the bug. Each project has a different approach to handling bug triaging. In general, triaging a bug report includes ensuring it contains enough details for developers, such as the steps to reproduce. Before exerting significant effort, the triager should ensure the bug has not already been reported. Figure 2.2 shows an example bug triaging procedure.

2.3 Machine Learning Algorithm

As we use Random Forest and neural networks in our work, we describe these particular algorithms and explain the two types of neural networks we use.

2.3.1 Random Forests

A decision tree is a technique that forms a tree-like structure. It has three components: decision nodes, leaf nodes, and the root node. This algorithm divides a training dataset into branches, segregating it into other branches until a leaf node is attained.

A Random Forest is a machine learning technique that can be used to solve regression and classification problems [15]. It utilizes an ensemble learning method that combines many classifiers to provide solutions to complex problems. A random forest algorithm

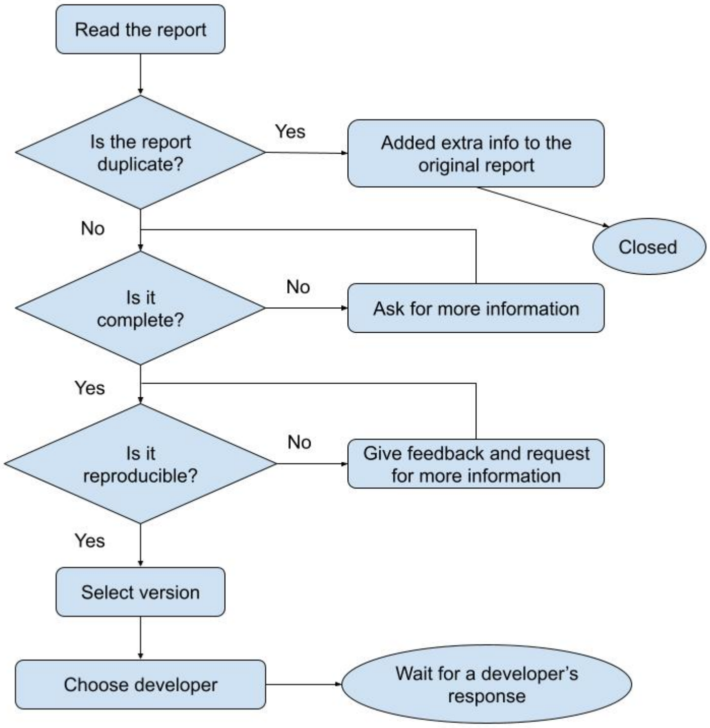


Figure 2.2: An example of a bug report triage workflow

contains many decision trees. The ‘forest’ is trained through bagging or bootstrap aggregating. Bagging or bootstrap aggregation is a powerful ensemble method that Leo Breiman proposed in 1994 to prevent overfitting [16]. It combines the predictions of several base learners to create a more accurate output [17]. This algorithm generates the outcome based on the predictions of the decision trees. For classification problems, the random forest output is the class chosen by the most trees. For regression tasks, the mean or average prediction of the individual trees is returned.

2.3.2 Artificial Neural Network

Artificial Neural Networks (ANN) are relatively simple electronic networks of neurons based on the brain’s neural structure. An ANN process records one at a time and learns by comparing its classification with the actual classification of the instance. The errors resulting from an instance’s classification are fed back into the network and used to modify the results for further iterations.

A neuron in an artificial neural network contains:

1. A set of input values (x) and associated weights (w).

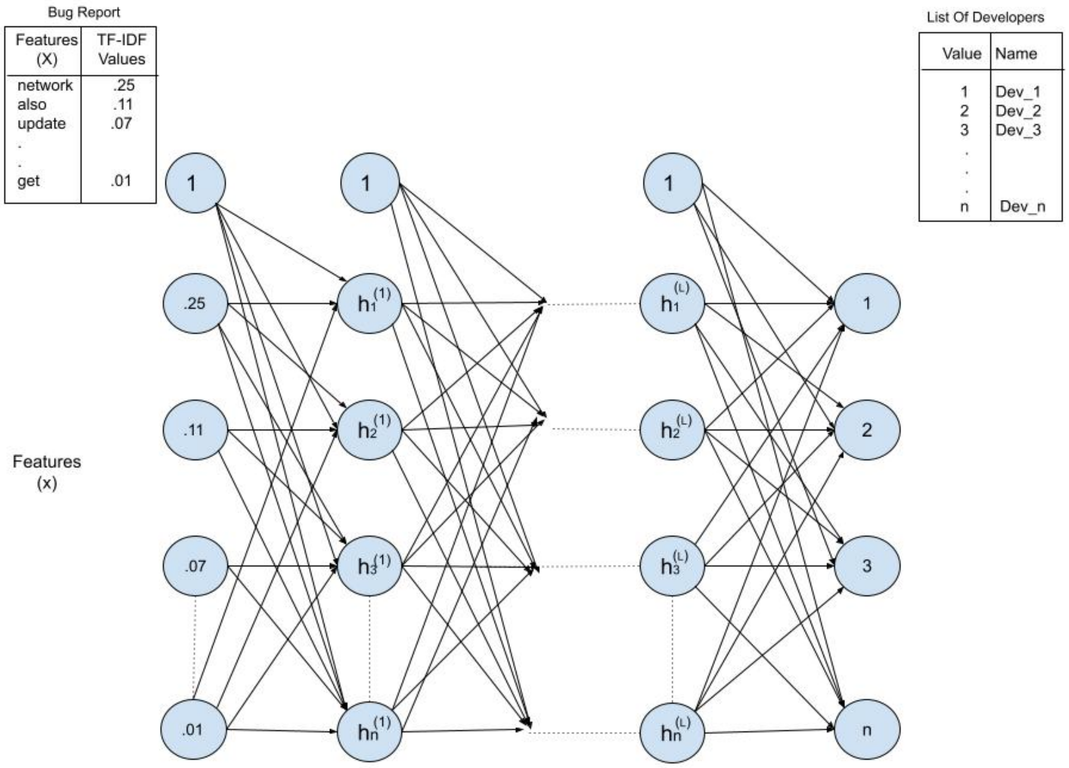
2. A function (g) that sums the weights and maps the results to an output (y).

The neurons in the network are organized into three layers: input, hidden and output. The input layer is composed not of total neurons but consists of the record's values that are transmitted to the next layer of neurons, which is the hidden layer. In the hidden layer, artificial neurons take a set of weighted inputs and produce an output through an activation function. A traditional artificial neural network has one or two hidden layers to process data. The final layer, with one node for each class, is the output layer. A single pass through the network assigns values to the output nodes, and the record is assigned to the class node with the highest value. [18]

2.3.3 Deep Neural Network

A Deep Neural Network (DNN) is an extension of the traditional artificial neural network (ANN). ANN uses one or two hidden layers to process data, whereas DNN uses several hidden layers between the input and output layers. DNN combines the computations performed by several layers.

Figure 2.3 shows an example of a DNN for a bug report assignment recommender system. The input layer in Figure 2.3 is the layer on the left, and it takes as inputs the TF-IDF values for the features'Bug Report':.25 (network),.11 (also),.07 (update) through to .01 (get) from the table. Then, multiple hidden layers use an activation function to create an output from a set of weighted inputs from the earlier levels. The rightmost layer, known as the output layer, takes the values from the last hidden layer and converts them into output values that correspond to the appropriate developer names from the table called 'List of Developers'.



Bug Report

Features (X)	TF-IDF Values
network	.25
also	.11
update	.07
.	
.	
get	.01

List Of Developers

Value	Name
1	Dev_1
2	Dev_2
3	Dev_3
.	
.	
.	
n	Dev_n

Figure 2.3: Example of a DNN for a Bug Report Assignment Recommender Systems

2.3.4 Multi-layer Perceptron

Multi-layer Perceptron is a type of DNN that learns the following function by training on a dataset.

$$f(.) : R^m \rightarrow R^o$$

Here,

$$m \rightarrow \text{number of dimensions for input}$$

$$o \rightarrow \text{number of dimensions for output}$$

MLP can learn a non-linear function estimator for either classification or regression from a feature set $X = x_1, x_2, ..., x_m$ and a target y. Figure 2.4 shows a one-hidden layer MLP with scalar output [19].

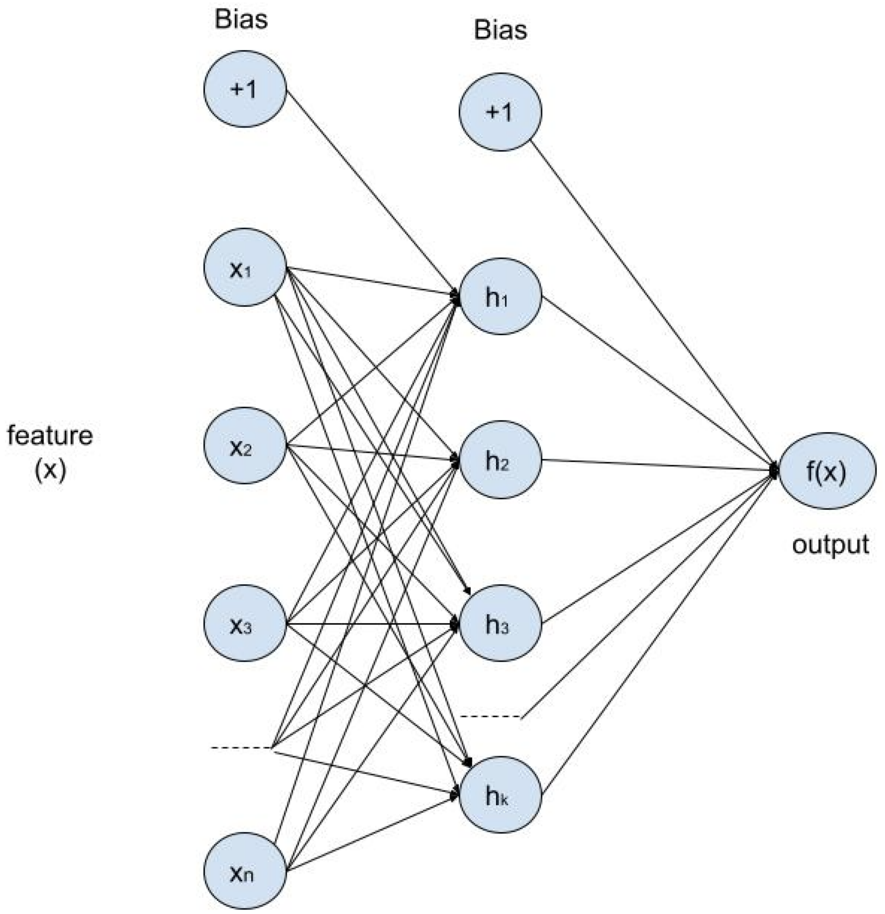


Figure 2.4: One hidden layer MLP

The leftmost layer is the input layer, $x_i | x_1, x_2, ..., x_n$, which represents the input features. At first, each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation $w_1x_1 + w_2x_2 + ... + w_nx_n$, followed by a non-linear activation function $g(.) : R \rightarrow R-$ like the hyperbolic tan function. Then the output layer receives the values from the last hidden layer and transforms them into output values [19].

2.3.5 Bidirectional LSTM

The LSTM (Long short-term memory)[2] based models are an extension of Recurrent Neural Networks (RNNs), which address the vanishing gradient problem. The LSTM models extend the RNNs' memory to enable them to keep and learn long-term dependencies of inputs. This memory extension can remember information over a more extended period, thus enabling reading, writing, and deleting information from their memories. The LSTM memory is called a 'gated' cell, where the word gate is inspired by the ability to preserve or ignore the memory information. An LSTM model captures important input features and preserves this information over a long time. The decision to delete or preserve the information is based on the weight values assigned during the training process. Hence, an LSTM model learns what information is worth preserving or removing [20].

The bidirectional LSTMs are an LSTM model extension in which two LSTMs are applied to the input data. In the first round, an LSTM is applied on the input sequence (i.e., forward layer). In the second round, the reverse form of the input sequence is fed into the LSTM model (i.e., backward layer). Applying the LSTM twice enhances the model's accuracy by improving the learning of long-term dependencies [20]. A previous study proved that the bidirectional networks are significantly better than the standard ones in many fields [21]. The structure of a Bidirectional LSTM layer with a forward and a backward LSTM layer is illustrated in Figure 2.5.

As the forward LSTM layer output sequence $\vec{h}$, is obtained typically as the unidirectional one, the backward LSTM layer output sequence $\overset{\leftarrow}{h}$, is calculated using the reversed inputs from time $t-1$ to $t-2$. These output sequences are then fed to σ function to combine into an output vector y_t [22]. Similar to the LSTM layer, a vector can represent the final output of a Bidirectional LSTM layer, $Y_t = [y_{t-n}, ..., y_{t-1}]$.

[2] A type of recurrent neural network capable of learning long-term data dependencies.

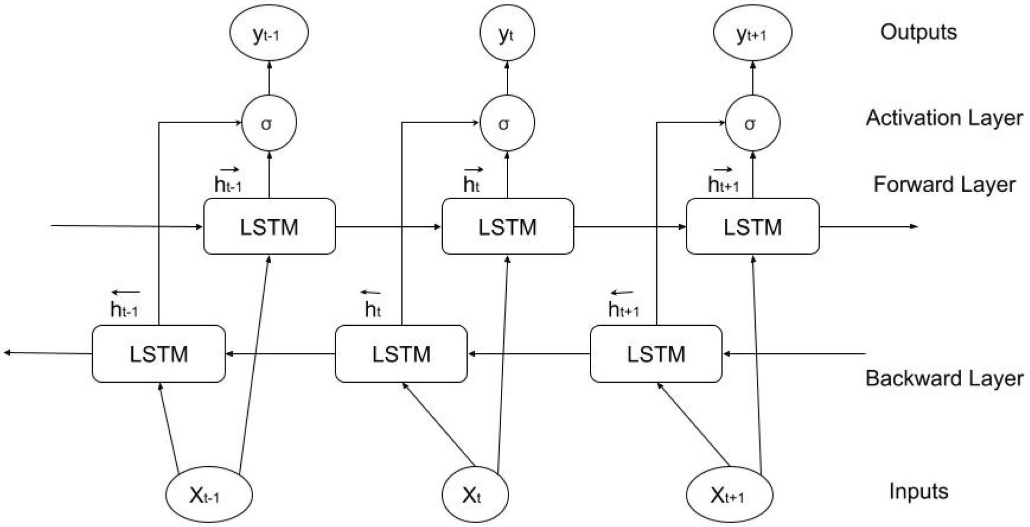


Figure 2.5: A Bidirectional LSTM

2.4 Feature Selection Algorithms

We describe TF-IDF and Word2Vec, as we utilize them for feature extraction in our work. We also describe WordNets, as we use it for data filtering to answer one of our research questions.

2.4.1 TF-IDF (Term Frequency-Inverse Document Frequency)

The acronym TF-IDF stands for the Term Frequency-Inverse Document Frequency of records. It can be described as determining how pertinent a word is to a corpus or series of texts. The meaning increases according to the number of times a word appears in the text, which is offset by the corpus's word frequency (dataset) [23].

This term weighting technique gives each word in a document a weight depending on the frequency of its terms *(tf)* and the reciprocal frequency of the document *(tf) (idf)*. The words with higher weight scores are thought to be more important.

The *tf-idf* weight is computed as

$$tf - idf(t,d) = tf(t,d) * idf(t)$$

where, *tf* is the normalized term frequency, and *idf* is the inverse document frequency.

The term frequency is the frequency of the word *t* in document *d*. We can simply use the raw count as the term frequency.

The raw count of the frequency:

$$tf(t,d) = count(t,d)$$

However, we typically use log_{10} to flatten the raw frequency. The reasoning for this is that a word does not become 100 times more likely to be significant to the meaning of the document by appearing 100 times in it. We also add 1 to the count since we cannot calculate the *log* of 0 [24].

The logarithmically scaled frequency:

$$tf(t,d) = log_{10}(count(t,d)+1)$$

The inverse document frequency (idf) is defined using the fraction

$$N/df(t)$$

where, *N* is the total number of documents in the collection, and *df(t)* is the number of documents that contain the term *t*. The higher this weight, the fewer documents in which a term appears. Terms that appear throughout all documents are given the lowest weight of 1 [24].

Finally, we calculate the term inverse frequency's logarithm. As a result, the weight of term *t* becomes:

$$idf(t) = log_{10}(N/df(t))$$

2.4.2 Word2Vec

The Word2Vec tool efficiently implements the continuous bag-of-words and skip-gram architectures for computing vector representations of words. It constructs a vocabulary from the training data and then learns its vector representation. The resulting word vector file is used as a feature in many natural language processing and machine learning applications [25]. The Word2Vec methods are fast, efficient to train, and readily available online with static code and pre-trained embeddings. Word2Vec embeddings are static, meaning that the method learns one fixed embedding for each word in the vocabulary. [24]

2.4.3 WordNet

WordNet is a lexical database designed under program control widely used in natural language processing. English nouns, adjectives, verbs, and adverbs are organized into synsets. A synset is a set of synonyms that refer to one concept. It provides an effective combination of traditional lexicographic information and modern computing. [26]

2.5 Explanation

A definition of explainability or interpretability is the extent to which a person can comprehend the justifications for a choice or behaviour, according to theories in philosophy, social science, and psychology [27]. By (1) making the entire decision-making process clear and understandable and (2) explicitly explaining each choice, AI/ML algorithms can become more explainable [28]. As a result, research has been conducted to examine how to explain the choices made by sophisticated black-box models and how to give explanations in a way that people will find acceptable.

Questions on the inference mechanism of AI/ML systems were divided by Lim et al. [29] into the following categories: What, Why, Why Not, What If, and How To. To illustrate, here are examples of each type of intelligibility question:

- What reasoning underlies the AI/ML models?

- Why is a particular class expected for a given instance?
- Why can't an instance of a different class be predicted?
- How can the system's prediction for an instance change from one class to another?
- If the values of an instance were to change, what would the system predict?

Prior research has frequently used white-box AI/ML techniques such as decision trees [30] and decision rules [31]. By closely examining the model components, we can immediately determine how much each metric contributed to the learned results due to the transparency of such white-box AI/ML methodologies, for instance, the pathways in a decision tree, the coefficients of each measure in a regression model, or the rules in a decision rule model. On the other hand, white-box AI/ML techniques are often less accurate than complex black-box AI/ML techniques and frequently generate generic explanations (e.g. one decision node may cover 100 instances) [14].

Therefore, there are two levels of explainability.

Global Explainability: Using interpretable machine learning techniques (e.g., decision trees, decision rules, or logistic regression techniques) or intrinsic model-specific techniques (e.g., ANOVA, variable importance) to ensure that the entire prediction and recommendation process is transparent and understandable. The goal of such intrinsic model-specific techniques is to provide global explainability. As a result, users can only understand how the model works globally (e.g., by inspecting a branch of decision trees). On the other hand, users frequently do not understand why the model makes that prediction.

Local Explainability: Using model-agnostic techniques (e.g., LIME [12], PyExplainer [13]) to explain software analytics model predictions (e.g., neural network, random forest). Such model-independent post-hoc techniques can explain each prediction (i.e., an instance to be explained). Users can then understand why the software analytics models make the predictions. [14].

2.5.1 Local Model

Understanding “Why did the model choose a particular outcome for a particular instance?” and “Why did the model choose a particular outcome for a group of Instances?” is the key. We treat a model as a black box and are not concerned with its underlying structure or the presumptions it makes for local interpretability. To understand prediction decisions for a single datapoint, we focus on that datapoint and investigate a local subregion in our feature space around it, attempting to understand model decisions for that point based on this local region. Local data distributions and feature spaces may behave very differently and provide more accurate explanations than global interpretations. An illustration of a local model interpretation is shown in Figure 2.6.

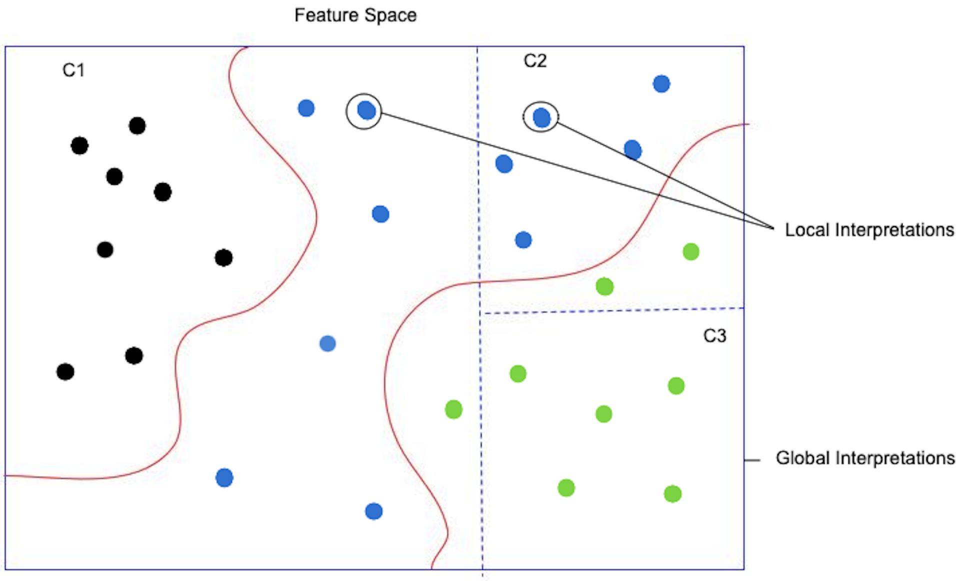


Figure 2.6: Local Model

2.5.2 RuleFit

The RuleFit algorithm [32] learns a sparse linear model with the original features and several new feature decision rules that capture interactions between the original features. The algorithm automatically generates features from decision trees. Each tree path is transformed into a decision rule by combining the split decisions into a rule. These trees are trained to predict the outcome of interest to ensure that the splits are meaningful for the prediction. Any algorithm, such as a random forest that generates many trees, can be used for RuleFit. Each tree is decomposed into decision rules used as additional features in a sparse linear regression model (Lasso). RuleFit, which helps identify linear terms and rules necessary for the predictions, can be used to measure the importance of a feature. With the help of the regression model's weights, feature importance is determined. For the original features, the significance metric can be aggregated. Additionally, partial dependence charts are introduced to display the average change in prediction caused by changing a feature. Any model can be utilized with the partial dependence plot, a model-agnostic technique. However, sometimes RuleFit creates many rules that get a non-zero weight in the Lasso model, and the interpretability degrades with an increasing number of features in the model. [33]

2.5.3 LIME

LIME (Local Interpretable Model-agnostic Explanations) [12] is a technique that successfully explains any classifier's predictions by learning an illustratable model locally around the prediction. It is used to interpret individual predictions of black-box machine learning models. Instead of training a global surrogate model, LIME focuses on training local surrogate models to explain individual predictions. LIME tests the predictions depending on variations of the data given into the ML model. It generates a new dataset consisting of perturbed samples and the corresponding predictions of the black-box model. This dataset trains an interpretable model weighted by the proximity of the sampled instances to the instance of interest. The learned model should be a good local estimation of the ML model predictions, but it does not have to be an excellent global estimation. This kind of accuracy is also called local fidelity. LIME works differently for different data types, such as; text data, tabular data, and image [33]. The output of LIME is a list of explanations that reflect every feature's contribution to the prediction of a data sample that provides local interpretability. It also determines which feature changes will impact the prediction most [34].

2.5.4 PyExplainer

PyExplainer [13] is a rule-based model-agnostic model. It utilizes a local rule-based regression model, which learns the associations between the synthetic instances' characteristics and the black-box model's predictions. Given a black box model and an instance to explain, it performs the following key steps to generate an explanation: [35]

- Generates synthetic neighbours around the instance to be explained using the crossover and mutation techniques.
- Obtains the predictions of the synthetic neighbours from the black-box model.
- Builds a local rule-based regression model.
- Generates an explanation from the local model for the instance to be explained.

2.6 Visualization

We describe the three visualization strategies examined in this work. First, we describe the word cloud visualization. Next, we describe the two visualizations used by LIME: a bar chart and a feature table.

2.6.1 Word Cloud

A word cloud visualizes text data, typically depicting free-form text's keyword metadata (tags). The tags usually are single words, and the importance of each tag is shown with font size. This format is helpful for quickly perceiving the most prominent terms. A bigger font size for a term is means the term has more significant weight or occurs more frequently [36]. Figure 2.7 shows an example of using a word cloud visualization.

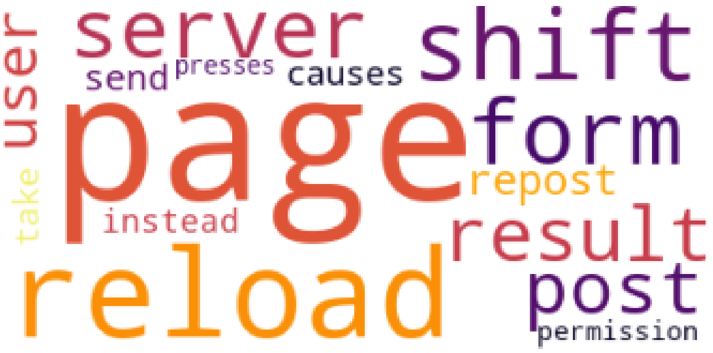


Figure 2.7: Example of word cloud visualization for a recommendation system

2.6.2 Bar Chart

A bar chart, often known as a bar graph, is a visual representation of categorical data that uses rectangular bars with heights or lengths that are proportionate to the values they represent. The bars can be plotted either vertically or horizontally. A vertical bar chart may also be referred to as a column chart. Figure 2.8 shows an example of two types of bar charts.

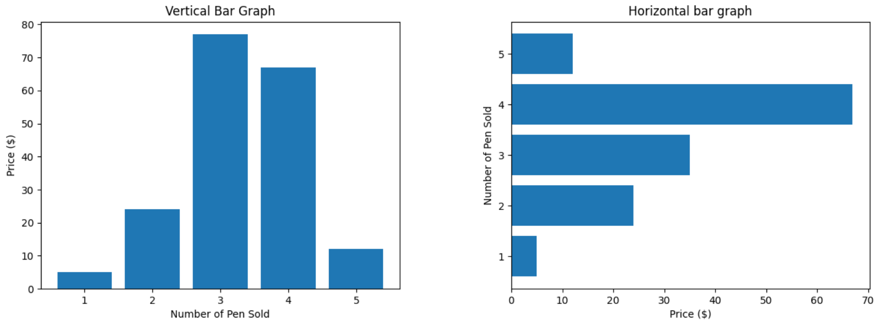


Figure 2.8: Example of bar charts

2.6.3 Positive-Negative Horizontal Bar Chart

To interpret and identify both positive and negative data values in a bar chart, this visualization splits a bar chart into two halves using the y-axis. The positive data is presented from the axis to the right, much like in a typical horizontal bar chart. On the other hand, the negative data is given on the left-hand side of the axis. The y-axis is zero at the beginning of the bars, with the highest value being on the far right and the lowest value on the left. The legend for the bars is presented on the sides opposite the bars across the y-axis. Figure 2.9 shows an example of a positive-negative horizontal bar chart.

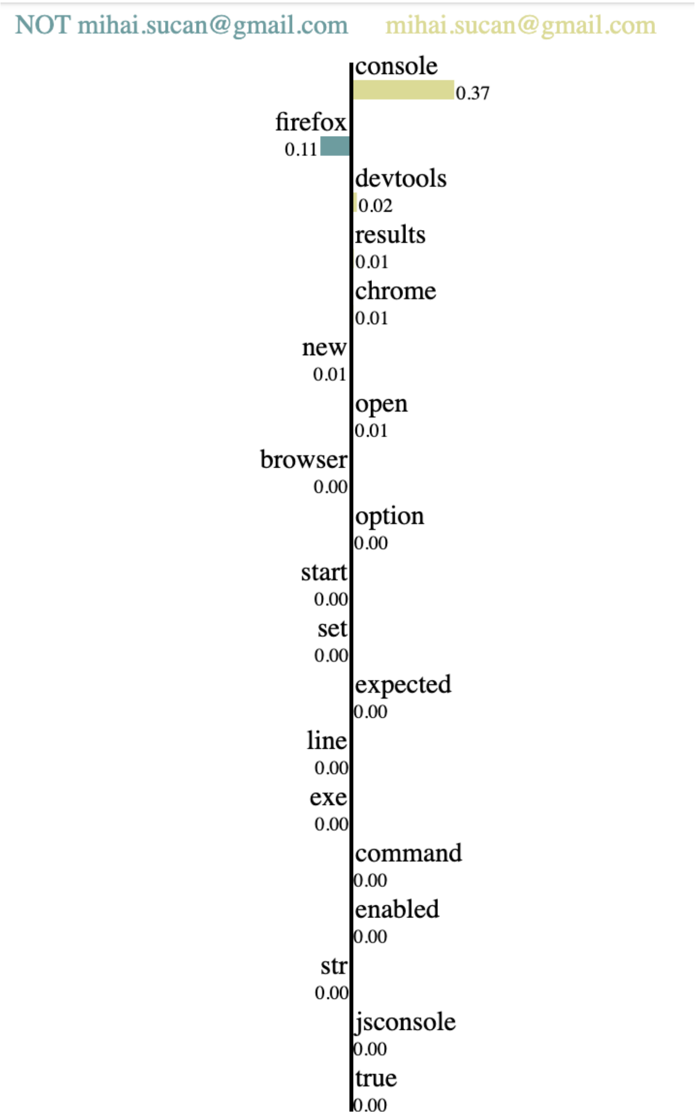
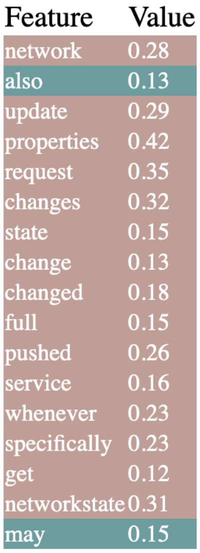


Figure 2.9: Example of a Positive-Negative Horizontal Bar Chart

2.6.4 Feature Table

A feature table is a table that shows a list of features together with the weights assigned to each one. A feature table in a recommender system shows each feature's impact on a particular choice. Figure 2.10 is an example of a feature table. Each row in Figure 2.10 represents a feature with its corresponding weight, and each colour represents which features impact which choice.



Feature	Value
network	0.28
also	0.13
update	0.29
properties	0.42
request	0.35
changes	0.32
state	0.15
change	0.13
changed	0.18
full	0.15
pushed	0.26
service	0.16
whenever	0.23
specifically	0.23
get	0.12
networkstate	0.31
may	0.15

Figure 2.10: Example of a Feature Table

2.7 Summary

This chapter presented the definition of a bug report, the bug report triage process, the details of various machine learning algorithms, feature selection, explanation, and visualization techniques, especially for global models like the Neural Network and Random Forest that we used in our implementation.

Chapter 3

Related Work

It is challenging for practitioners to comprehend how the models used for bug report assignment recommendations arrive at a choice because the majority of such research uses black-box models, such as neural networks and random forests. As these works focus more on accuracy than on explanation, the adoption of such recommender systems suffers significantly from a lack of explainability since it fosters a lack of confidence and transparency.

This section discusses the previous research related to the bug report assignment recommender system, the use of WordNet for improving accuracy, explaining global models using different local models and various visualization techniques.

3.1 Bug Report Assignment Recommender Creation Techniques

Table 3.1 presents a list of significant related works on bug triaging. The table is an extension of the one presented in Mani et al. [1], representing the work chronologically from 2010 to 2016. Our version extends this previous table to include works until 2021.

A majority of previous techniques used the summary/title and description [3] [4] [6] [37] because they are available at the time of ticket submission and do not change during a ticket's lifetime. Bhattacharya et al. [5] use additional attributes such as product, component, and the last developer activity to shortlist developers. Shokripour et al. [38] used code information for improved performance. Bastian et al. [39] identify developers' expertise using StackOverflow and keywords from bug descriptions.

From Table 3.1, we observe the use of many different feature models, such as term-

Table 3.1: An overview of the various machine learning-based bug triaging techniques that have been studied, outlining their characteristics and methods, as well as their experimental results.

Paper	Information used	Feature extracted	Approach	Dataset	Performance
Bhattacharya et al., 2010 [5]	title, description, keywords, product, component, last developer activity	TF-IDF + bagof- words	Naive Bayes + Tossing graph	Eclipse# 306,297 Mozilla# 549,962	Top#5 accuracy 77.43% Top#5 accuracy 77.87%
Tamrawi et al., 2011 [37]	title, description	terms	A fuzzy-set feature for each word	Eclipse# 69829	Top#5 accuracy 68.00%
Anvik et al., 2011 [3]	title, description	normalized TF	Naive Bayes, EM, SVM, C4.5, nearest neighbor, conjunctive rules	Eclipse# 7,233 Firefox# 7,596	Top#3 prec. 60%, recall 3% Top#3 prec. 51%, recall 24%
Xuan et al., 2012 [4]	title, description	TF-IDF, developer prioritization	Naive Bayes, SVM	Eclipse# 49,762 Mozilla# 30,609	Top#5 accuracy 53.10% Top#5 accuracy 56.98%
Shokripour et al. 2013 [38]	title, description, detailed source code info	weighted unigram noun terms	Bug location prediction + developer expertise	JDT-Debug# 85 Firefox# 80	Top#5 accuracy 89.41% Top#5 accuracy 59.76%
Wang et al., 2014 [7]	title, description	TF	Active developer cache	Eclipse# 17,937 Mozilla# 69,195	Top#5 accuracy 84.45% Top#5 accuracy 55.56%
Xuan et al., 2015 [6]	title, description	TF	feature selection with Naive Bayes	Eclipse# 50,000 Mozilla# 75,000	Top#5 accuracy 60.40% Top#5 accuracy 46.46%
Badashian et al., 2015 [39]	title, description, keyword, project language, tags from stackoverflow, github	Keywords from bug and tags	Social expertise with matched keywords	20 GitHub projects, 7144 bug reports	Top#5 accuracy 89.43%
Jonsson et al., 2016 [8]	title, description	TF-IDF	Stacked Generalization of a classifier ensemble	Industry# 35,266	Top#1 accuracy 89%
Mani et al., 2018 [1]	title, description	Word2Vec	Attention-based deep bidirectional recurrent neural network (DBRNN-A) + Softmax	Chromium# 383,104 Core# 314,388 Firefox# 162,307	Top#10 accuracy 47.0% Top#10 accuracy 43.3% Top#10 accuracy 55.8%
Tanaz S et al., 2021 [2]	description, component	Lemmatizer	Tossing Graph + Multinomial Naive Bayes Random Forest SVM	Eclipse# 2,868,000	Top#10 accuracy 62.3% Top#10 accuracy 66.7% Top#10 accuracy 65.6%

frequency (TF), normalized TF, TF-IDF, n-grams and Word2Vec. Various classifiers, such as Neural Network [1], Random Forest [2] , SVM [2] [3] [4] and Naive Bayes [2] [3] [4] [5] [6] were also used. Some compare the accuracy between different types of classifiers using the same dataset [1] [2]. However, to our knowledge, no previous work has compared the accuracy between Neural Network and Random Forest algorithms.

3.2 Use of WordNet in Recommendation Systems

Recommender systems accuracy has been shown to improve using WordNet for feature extraction. Haifa et al. used WordNet synsets to enrich a content-based recommender system [40]. Bernardo et al. used WordNet for improving user modelling in a web document recommender system [41]. However, the use of WordNet for bug report assignment recommendations has not been explored.

3.3 Explanation Techniques using Local Model

The use of local explanation allows practitioners to understand better the reasons behind the predictions of the ML models [42]. LIME [12] is one of the model-agnostic techniques that have been widely adopted to address various software engineering problems and other domains. For example, recent work [43] [44] uses LIME for explaining line-level defect predictions. Jiarpakdee et al. [42] found that LIME effectively explains the predictions of file-level defect prediction models.

However, LIME has the following limitations. First, LIME's neighbourhood generation process is suboptimal. Second, the approximation of the LIME local models to the predictions of the global model is suboptimal. Third, the explanations generated by LIME are not specific to an instance to be explained. To solve these limitations, Pornprasit et al. proposed a rule-based model-agnostic technique called PyExplainer [13]. We described LIME and PyExplainer in detail previously (see Sections 2.5.3 and 2.5.4).

3.4 Recommendation Explanation using Visualization

Explainability is essential for recommendation systems, and researchers are trying to implement different types of explainable techniques for recommender systems. Saeed et al. enhanced the explainability of the social recommendation system using 2D graphs, and word cloud visualizations [45].

Ninghao et al. implemented an explainable recommender system by resolving learning representations. The authors first analyze the IOM (Input data format, Output attribution, and Middle-level representations) elements that benefit model interpretability and then propose a more transparent representation learning module model. The proposed model considers all three elements above, and the data is formatted as a graph. According to different source types and factors, the model resolves information passing in graph convolution according to different source types and factors [46].

Bhyan et al. investigated visualization for explaining bug report assignment recommenders and found that developers prefer visual explanations. Seventy-five percent of participants in their user study stated that the visual explanations increased their understanding of the assignment recommendations and helped them trust the process [47].

Deep neural networks (DNNs) have become indispensable machine learning tools. As a black-box model, it is not easy to diagnose what aspects of the model's input drive the decisions of a DNN. In real-world domains, from law enforcement to healthcare, such diagnosis is essential to ensure DNN decisions are driven by aspects appropriate for its use. Therefore, developing the methods and studies enabling the explanation of a DNN's decisions has blossomed into an active, broad area of research. Gabrielle et al. wrote a field guide to explore the space of explainable deep learning aimed at those inexperienced in the field [48]. It introduces three simple dimensions defining the space of foundational methods that contribute to explainable deep learning, discusses the evaluations for model explanations, places explainability in the context of other related deep learning research areas, and finally elaborates on user-oriented explanation designing, and potential future directions on explainable deep learning [48].

Alvin et al. proposed neural-backed decision trees (NBDTs) that improved the accuracy and interpretability of modern neural networks by drawing unique insights from the hierarchy. They also confirm that humans trust NBDTs over saliency and illustrate how path entropy can be used to identify ambiguous labels [49].

3.5 Summary

Although recommenders for bug report assignments have been well studied, NN and RF have never been compared. Word2Vec with NN was used in the majority of research studies, however, TF-IDF with NN had never been investigated in the context of bug report assignment recommenders. WordNet has been utilized in a number of recommender systems in the past, but never in bug report assignment recommenders. Researchers have used the local models (LIME and PyExplainer) in several works to explain the global models (RF and NN), which have not yet been investigated for a bug report assignment recommender system. Although work on visually explaining bug report assignment recommender has been previously done, these techniques have not been applied to recommenders created using RF and NN.

Chapter 4

Approach

The methods for developing explainable bug report assignment recommender systems are presented in this section. First, we integrated three different global models: Random Forest, MLPClassifier, and Bidirectional Neural Network, with two different feature selection techniques: TF-IDF and Word2Vec, to create several recommender systems. Then we introduce the use of a local model, specifically LIME, to explain recommendations. Finally, we used three different visualization techniques to present the results visually. The source code is available on GitHub [3].

4.1 Data Source and Preparation

In our research, we used the data set gathered by Mani et al. [1]. In their study, Mani et al. used bug reports from three different projects. Table 4.1 lists the specifics of the datasets they gathered. Every bug report has fields like ‘title’, ‘description’, ‘owner’ or ‘assigned-to’, and ‘reported time’. The only fields we used in our research were ‘title’, ‘description’, and ‘owner’ or ‘assigned-to’. After collecting these bug reports with the required fields, all bug reports must have their features extracted and labelled. In every instance, the text content of the bug’s title and description are merged to extract features, and the terms ‘owner’ or ‘assigned-to’ are utilized to label the bug. Google Chromium considered the developer listed in the ‘owner’ field the ground truth assignment class for the specific bug. Mozilla’s developer in the ‘assigned-to’ field is considered the ground

[3] https://github.com/fyomee/bug_triaging_with_visual_explanation

truth assignment class during categorization.

Table 4.1: Data set details.

Data Source	**Mozilla Firefox**	**Mozilla Core**	**Google Chromium**
Date Range	July 1999 - June 2016	April 1998 - June 2016	August 2008 - July 2016
Bug Reports	ID:10954 - ID:1278030	ID:91 - ID:1278040	ID:2 - ID:633012
Total Bug Reports	162,307	314,388	383,104
Bugs For Learning Feature	138,093	186,173	263,936
Bugs for classifier	20,417	35,000	20,000

We downloaded data from Mani et al.'s site [4] and fixed some JSON formatting-related issues, such as NULL handling. Every bug report has an owner, title and description, which are unstructured. In our research, we considered the threshold for the minimum number of training samples per class as 20. However, we could not run the complete datasets for Chromium and Mozilla Core due to memory issues [5]. Therefore, we used the first 20,000 bug reports from Chromium and 35,000 from Mozilla Core as our dataset. The summary of the datasets is provided in Table 4.1.

4.2 Data Preprocessing

After combining the title and description field content for every bug report, this unstructured data is first processed by removing URLs, hex codes, stack trace information and converting all text to lowercase. Then the text is tokenized, and stop words are removed. We experimented with the use of stemming and lemmatization; however, we found that these steps did not improve the accuracy of the created recommenders.

When creating the training and testing datasets for evaluation, a 10-fold cross-validation model proposed by Betternburg et al. [50] is followed to remove the training bias. The data is arranged in chronological order. Typically, the developers in an open source project change over time; hence, chronological splitting ensures that the training and testing sets have overlapping developers [1]. We split the data into eleven partitions. Except for the first

[4] http://bugtriage.mybluemix.net/

[5] Experiments were run on a MacBook with a 2.6GHz 6 core intel core i7 processor and 32GB 2667 MHz DDR4 memory, running macOS Big Sur version 11.6.8.

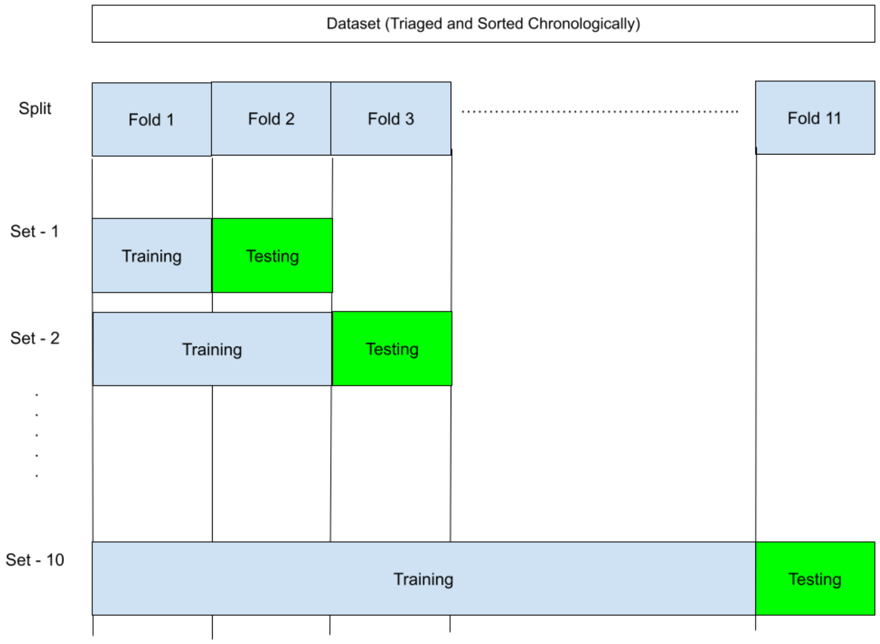


Figure 4.1: Data splitting for 10-fold cross-validation

one, all the folds are used as a testing set with the cumulation of previous folds for training, as shown in Figure 4.1.

We ensure that all the classes appearing in the testing set appear in the training set. At the same time, additional classes in the training data are unavailable in the testing set. In other words, for every developer in the testing data sets, the classifier is trained with other bugs fixed by the same developer.

4.3 Recommender Creation using TF-IDF

We use TF-IDF (term frequency-inverse document frequency) to create a word vector for each bug report. The flow diagram in Figure 4.2 highlights the significant steps we used in creating a recommender system using TF-IDF. The flow of employing a local model is highlighted in the figure using blue text.

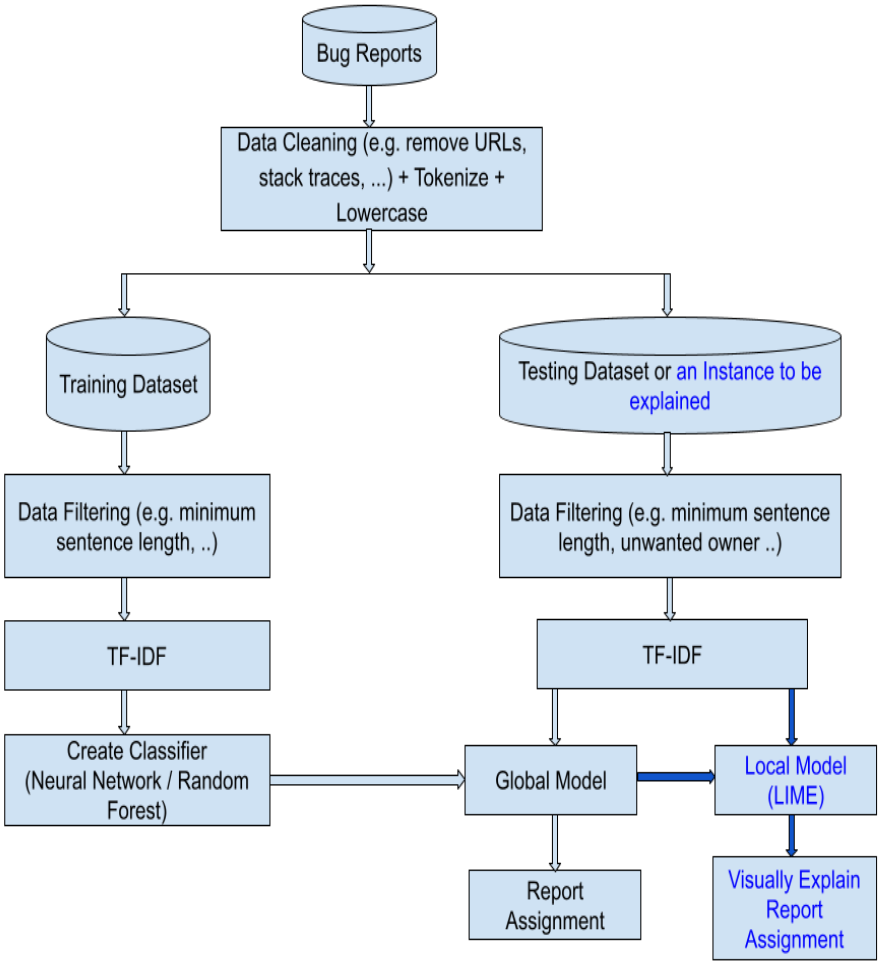

Bug Reports
Data Cleaning (e.g. remove URLs, stack traces, ...) + Tokenize + Lowercase
Training Dataset
Testing Dataset or an Instance to be explained
Data Filtering (e.g. minimum sentence length, ..)
Data Filtering (e.g. minimum sentence length, unwanted owner ..)
TF-IDF
TF-IDF
Create Classifier (Neural Network / Random Forest)
Global Model
Local Model (LIME)
Report Assignment
Visually Explain Report Assignment

Figure 4.2: The flow diagram for bug report assignment recommendation using TF-IDF

4.3.1 TF-IDF (Term Frequency-Inverse Document Frequency)

We use `TfidfVectorizer` with the following tuned parameters from `sklearn` [6]:

- **min_df** = 5

  Ignores words with a document frequency lower than the given threshold.

- **use_idf** = true

  We set this as true for enabling inverse-document-frequency reweighting.

Then we implement three global models using Random Forest, Bidirectional Neural

[6] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

Network and Multi-layer Perceptron Classifier.

4.3.2 Random Forest

We use `RandomForestClassifier` from `sklearn` [7].

Instead of allowing each classifier to vote for a single class like in the original publication [51], the `scikit-learn` method combines classifiers by averaging their probabilistic predictions. [52]

The tuned parameter settings we used to customize this classifier are the following:

- **n_estimators** = 1000

  The number of trees in the forest.

- **max_depth** = 300

  The maximum depth of the tree.

4.3.3 MLP Classifier

`MLPClassifier` in `sklearn` [8] implements a multi-layer perceptron (MLP), using gradient descent for training, and backpropagation is used to calculate the gradients. It minimizes the Cross-Entropy loss function for classification, giving a vector of probability estimates $P(y|x)$ per sample x. Using Softmax as the output function, `MLPClassifier` provides multi-class classification. The model supports multi-label categorization, allowing samples to belong to many classes. The raw output is processed using a logistic function for each class. The rounding of the output is used to determine the value of the predicted class [19].

We trained MLP with two arrays: the first one is a training sample represented as floating point feature vectors size of $X(n\_samples, n\_features)$, and the second one is the target

[7] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[8] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html #sklearn.neural_network.MLPClassifier

values (class labels) size of $y(n\_samples,)$ for the training samples. After training/fitting, the model predicts labels for new samples.

We used the following tuned parameters for customizing the models [53]:

- **hidden_layer_sizes** = 150
- **solver** = `adam` [9]

  We used `adam` as a solver, which is the default because it works relatively better for large datasets [53].

4.3.4 Bidirectional LSTM Neural Network

In this model, we utilize bidirectional LSTM (long-short-term memory), a technique that allows any neural network to store sequence data in both forward and backward directions. A bidirectional LSTM differs from a conventional LSTM in that input flows in both ways. We can direct input to flow in either a forward or a backward direction using the standard LSTM.

We first create a `sequential model` from the `Keras` [10] library. Then we add a `Bidirectional` class with an `LSTM layer`.

The `LSTM layer` used the following tuned parameter settings [11]:

- **units** = 1024

  The dimensionality of the output space.
- **activation** = `tanh`

  We used the default activation function hyperbolic tangent (tanh).
- **recurrent_activation** = `sigmoid`

  We used the sigmoid function as the function for the recurrent step.

[9] `adam` refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba

[10] https://keras.io/guides/sequential_model/

[11] https://keras.io/api/layers/recurrent_layers/lstm/#lstm-class

- **use_bias** = True

  We set this as true to ensure the layer uses a bias vector.

Then we added a `dropout layer` with a rate of 0.5. [12]

Finally, we added a `dense layer` with the following arguments [13]:

- **units** = len(class)

  The dimensionality of the output space. We used class length as a unit.

- **activation** = `softmax`

After creating the model, during the compilation, we used the following arguments [14]:

- **optimizer** =`rmsprop`

  We use `RMSprop` as an optimizer from `Keras` [15].

- **loss** = `categorical_crossentropy`

  We used categorical_crossentropy as a loss function.

- **metrics** = `accuracy`

  We used `accuracy` as a callable function with the signature result equal `fn(y_true, y_pred)`.

Then we trained the model with three-dimensional training data `(n_samples, n_timesteps, n_features)` and the target class with batch_size = 32, with 5, 6, and 10 epochs [16] for Mozilla Core, Mozilla Firefox and Google Chrome, respectively [17].

4.3.5 Testing

After creating and training these models, we calculated accuracy using the test set. The results from this step are presented in Chapter 5.

[12] https://keras.io/api/layers/regularization_layers/dropout/

[13] https://keras.io/api/layers/core_layers/dense/

[14] https://keras.io/api/models/model_training_apis/

[15] https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/RMSpropr

[16] Each iteration of the specified x and y data constitutes an epoch.

[17] https://keras.io/api/models/model_training_apis

4.3.6 Local Model

To explain the recommendation result, we implement the local model using LIME [12]. We tried to implement the local model using Pyexplainer [13] as well; however, during the implementation, we discovered issues with its use for our application (see Section 6.1 for details). Our implementation procedure is as follows:

1. Construct an explainer. For both the Random Forest and MLP classifier, we construct a `LimeTabularExplainer` with the following tuned parameters [54]:

    - **training_data:** NumPy 2d array
    - **mode:** classification
    - **training_labels:** the names of developers.
    - **feature_names:** list of words corresponding to the columns in the training data.
    - **class_names:** list of developers, ordered according to whatever the classifier uses.

    And for a bidirectional neural network, we construct a `RecurrentTabularExplainer` with the following tuned parameters [54]:

    - **training_data:** NumPy 3d array with shape `(n_samples, n_timesteps, n_features)`
    - **mode:** classification.
    - **training_labels:** the names of developers.
    - **feature_names:** list of words corresponding to the columns in the training data.
    - **class_names:** list of developers, ordered according to whatever the classifier uses.
    - **discretize_continuous:** We set it as true to ensure all non-categorical features will be discretized into quartiles.

2. Use the constructed explainer with the predict function of the predictive model to explain any instance. We implement `explain_instance` using the following tuned parameters[18]:
    - **data_row:** 2d numpy array, corresponding to a row.
    - **classifier_fn:** classifier prediction probability function, which takes a NumPy array and outputs prediction probabilities. For `ScikitClassifiers`, this is `classifier.predict_proba`.
    - **top_labels:** If not none, ignore labels and come up with explanations for the K labels that have the highest chance of being correct, where K is this parameter.
    - **num_features:** maximum number of features present in explanation.
3. Visualise the generated LIME explanation. We used a function called `show_in_notebook` for visualizing the explanation [54].

4.3.7 Word Cloud

Using the `WordCloud` package and the feature frequencies produced by the local model LIME, we made a word cloud image to visually explain the bug report assignment recommender systems.

4.4 Recommender Creation using Word2Vec

The flow diagram in Figure 4.3 highlights the significant steps we took in creating a recommender system using Word2Vec. With the exception of the data extraction and filtering stage, it is comparable to the TF-IDF technique. In the case of the TF-IDF dataset, we first filtered the information by determining the minimum length of the text string and eliminating any unwanted owners from the test datasets. We then extracted features using

[18] https://lime-ml.readthedocs.io/en/latest/lime.html

the filtered training datasets. While in Word2Vec, the feature is first extracted using untriaged bug reports with training datasets, and the data is subsequently filtered by deleting words that are not included in the Word2Vec vocabulary list. The unwanted owner was then removed from the testing datasets once we verified the minimal length of the text string.

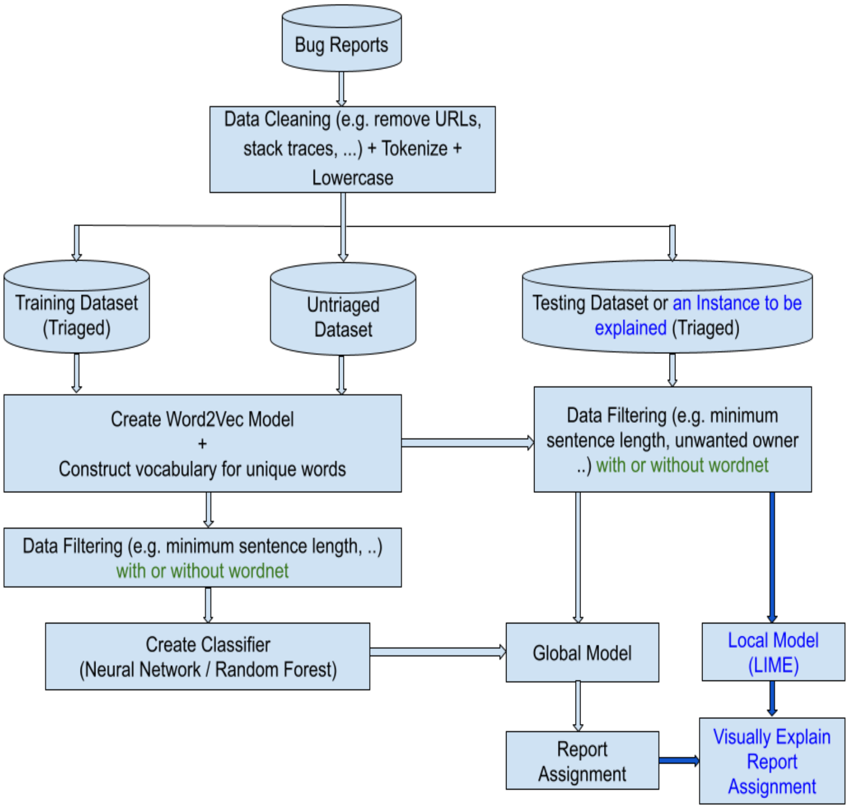

Bug Reports
Data Cleaning (e.g. remove URLs, stack traces, ...) + Tokenize + Lowercase
Training Dataset (Triaged)
Untriaged Dataset
Testing Dataset or an Instance to be explained (Triaged)
Create Word2Vec Model + Construct vocabulary for unique words
Data Filtering (e.g. minimum sentence length, unwanted owner ..) with or without wordnet
Data Filtering (e.g. minimum sentence length, ..) with or without wordnet
Create Classifier (Neural Network / Random Forest)
Global Model
Local Model (LIME)
Report Assignment
Visually Explain Report Assignment

Figure 4.3: The flow diagram of the Bug Triage using Word2Vec

4.4.1 Word2Vec

In our data split mechanism, the classifier testing data is unseen and cannot be used for training the Word2Vec model. Therefore we used untriaged [19] bug reports from the training data set for training the model.

For creating this model, we use the `gensim` library [55] with the following tuned parameters:

[19] Bug reports which are not assigned to any developer.

- **sentences:** A list of lists tokens.
- **vector_size** = 300

  Dimensionality of the word vectors.
- **window** = 5

  Maximum distance allowed within a sentence between the present and anticipated words.
- **min_count** = 5

  Ignores all words with a total frequency lower than this.

We get the vocabulary list from the Word2Vec model, which we use for filtering the training and testing datasets.

4.4.2 WordNet

We experimented with the use of WordNet. Figure 4.3 uses green text to emphasize the use of WordNet, which we included during the filtering process. In this stage, words that are not part of the Word2Vec vocabulary list are removed from the data. Before eliminating terms that are not in the vocabulary list, we use the `synsets` and `lemma_names` functions from `NLTK` to find their synonyms. If any of the synonyms appear in the vocabulary, we keep them. Later, we used that synonyms vector to construct the average feature vector for a bug report.

After that, we implemented three global models: Random Forest, MLP Classifier and Bidirectional Neural Network. We explained the results with the local models and word cloud as described in Section 4.3.

4.5 Summary

This chapter presented the approaches used in our investigation. First, we constructed six bug report recommender systems using two separate feature extractors, TF-IDF and

Word2Vec, along with three global models: Random Forest, MLP Classifier, and Bidirectional Neural Network. Then, we tried to increase the accuracy by including WordNet as a filter. Finally, we implemented a local model using LIME, which provided two types of visual explanation and created a word cloud to explain each model.

Chapter 5

Evaluation

This chapter presents the results of our evaluations for answering our research questions. Recall that our research questions are:

- RQ#1: Which classifier gives us better accuracy: Random Forest (RF), Multi-layer Perceptron (MLP) or Bidirectional Neural Network (BNN)?
- RQ#2: Which feature extraction approach gives us better accuracy: TF-IDF or Word2Vec?
- RQ#3: Does using WordNet improve the recommender accuracy?
- RQ#4: Which local model explanation approach, feature-based or rule-based, provides better explanations?
- RQ#5: Which visualization method appears to best support explanations of assignment recommendation: word cloud, positive-negative horizontal bar chart or feature table?

We will provide our findings for RQ#1, RQ#2 and RQ#3 in Section 5.1. Section 5.2 will present our findings to answer RQ #4 and RQ#5.

5.1 Recommender System Accuracy

The trained recommender system assigns each developer a probability value for a specific bug report, indicating their association with it. The Top-K accuracy evaluation metric, which measures the proportion of bug reports for which the actual developer is listed in the

Top-K recommendation, is used to compare the trained recommenders. Different classes, or groups of developers, are used throughout the cross-validation (CV) approach. As a result, the classes used for training and testing during CV#1 differ from those used for CV#2. Therefore, to evaluate the actual accuracy, we report the average accuracy of the Top-1, Top-5, and Top-10 cross-validation sets to understand the variance introduced in the model training [56].

For our three different datasets, Mozilla Firefox, Mozilla Core and Google Chromium, accuracy for Random Forest (RF), Multi-layer Perceptron (MLP) and Bidirectional Neural Network (BNN) classifiers are reported in Table 5.1.

Table 5.1: Average accuracy(%) over the cross-validation for Rank - 1, 5 and 10 are reported for all classifiers.

Rank		**Top-1**			**Top-5**			**Top-10**		
Classifiers		**RF**	**MLP**	**BNN**	**RF**	**MLP**	**BNN**	**RF**	**MLP**	**BNN**
Mozilla Firefox	**TF-IDF**	15.55	14.85	**17.35**	36.54	36.58	**39.08**	48.75	**48.82**	48.81
	Word2Vec	11.24	12.82	**14.29**	29.66	33.05	**34.1**	41.28	**45.66**	44.76
Mozilla Core	**TF-IDF**	19.52	18.12	**21.18**	42.99	41.54	**45.65**	54.08	53.41	**56.33**
	Word2Vec	14.63	15.1	**19.19**	35.12	38.48	**42.1**	46.86	51.08	**52.48**
Google Chromium	**TF-IDF**	17.01	17.03	**17.83**	35.96	35.85	**36.53**	**45.14**	44.17	44.69
	Word2Vec	11.03	11.53	**12.65**	26.91	**28.46**	27.74	34.95	**37.71**	35.06

5.1.1 RQ#1: Which classifier gives us better accuracy: Random Forest (RF), Multi-layer Perceptron (MLP) or Bidirectional Neural Network (BNN)?

From Table 5.1, we can see in most cases, a Bidirectional Neural Network(BNN) gives us better accuracy. For both the Mozilla Firefox dataset using TF-IDF and Word2Vec, the Top-10 accuracy MLP Classifier gives us the best result with 48.82% and 45.66%, respectively, where the accuracy when using BNN is 48.81% and 44.76%. However, in both cases, the accuracy between the approaches is less than 1%. We found that BNN always gives us the best accuracy for all three of the Top-Ks examined for Mozilla Core. For the Google Chrome dataset, when using TF-IDF, Random Forest gives us the best accuracy for the Top-10 (45.14%), and BNN gives us the next best accuracy (44.69%). However, the difference between these two recommenders is <0.5%. Using Word2Vec, the MLP classifiers give us better accuracy for Top-5 and Top-10, which are 28.46% and 37.71%, respectively, while the accuracy given by BNN is 27.74% and 35.06%. The difference between these results for Top-5 is 0.72%, and Top-10 is 2.65%.

Also, we can see from our results that which classifier gives us a better result depends on the dataset we used. However, the accuracy when using a particular approach, such as TF-IDF or Word2Vec, is almost similar for the different classifiers.

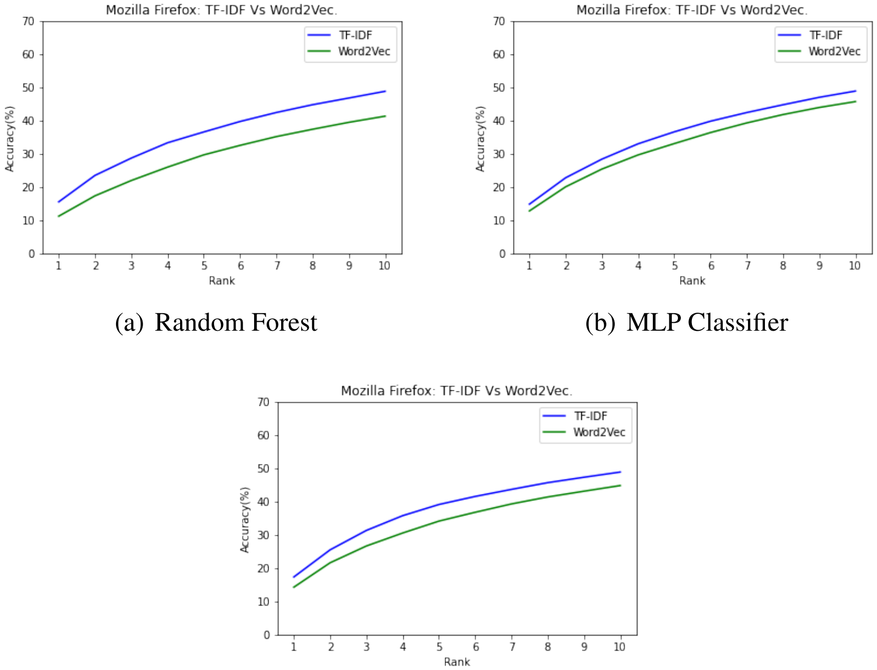
While implementing any neural network, such as MLP or BNN, is more complicated than RF, when using TF-IDF, the accuracy provided by RF was found to be similar to the best accuracy produced by either BNN or MLP, with the highest difference between the two being less than 3%. Table 5.2 shows that the training times for MLP and BNN are always three to five times and fifteen to twenty-six times longer than those for RF, respectively. As a result, we recommend that RF may be a better choice for creating such recommender systems.

Table 5.2: Average runtime for all the classifiers uses TF-IDF as a feature extractor.

Classifiers	**RF**	**MLP**	**BNN**
Mozilla Firefox	27.29 sec	134.99 sec	708.91 sec
Mozilla Core	77.47 sec	423.99 sec	1370.74 sec
Google Chromium	90.01 sec	304.79 sec	1434.64 sec

5.1.2 RQ#2: Which feature extraction approach gives us better accuracy: TF-IDF or Word2Vec?

The average CV10 accuracy for three separate datasets is shown in Figures 5.1, 5.2 and 5.3. Accuracy(%) is plotted on the y-axis, and Top-K rank is plotted on the x-axis. The green line shows accuracy when using Word2Vec as a feature extraction method, whereas the blue line shows accuracy when using TF-IDF as a feature extraction approach. From these figures and Table 5.1, we can see that using TF-IDF results in better accuracy.

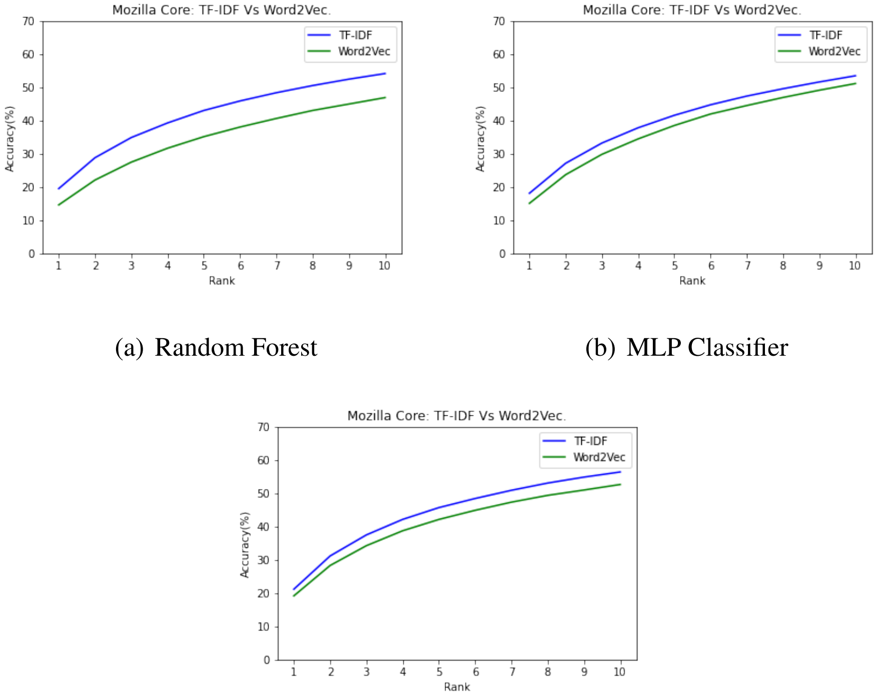


(a) Random Forest

(b) MLP Classifier

(c) Bidirectional Neural Network

Figure 5.1: Mozilla Firefox: CV10 average accuracy: TF-IDF Vs Word2Vec

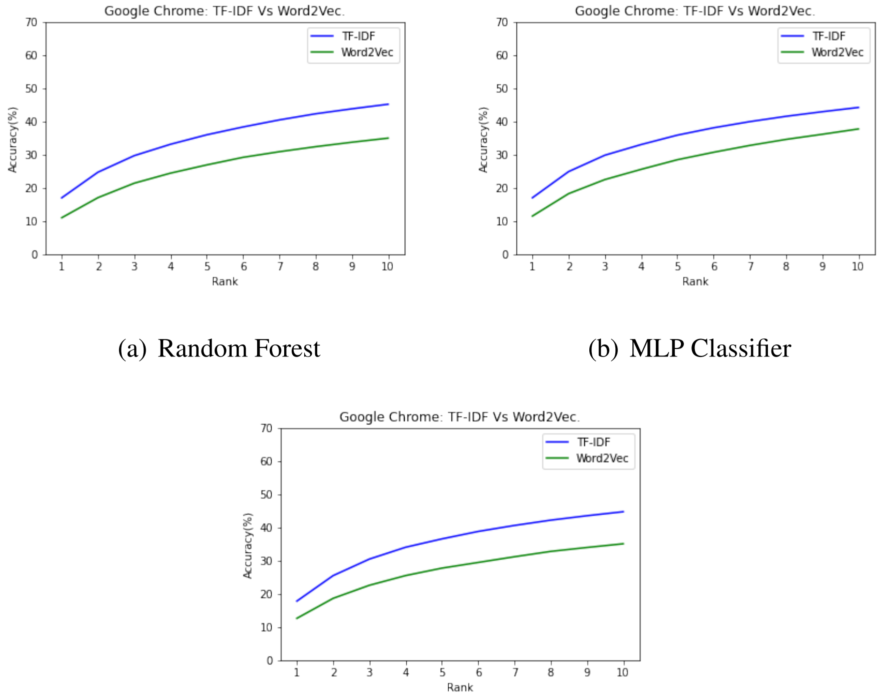


(a) Random Forest

(b) MLP Classifier

(c) Bidirectional Neural Network

Figure 5.2: Mozilla Core: CV10 average accuracy: TF-IDF Vs Word2Vec



(a) Random Forest

(b) MLP Classifier

(c) Bidirectional Neural Network

Figure 5.3: Google Chrome: CV10 average accuracy : TF-IDF Vs Word2Vec

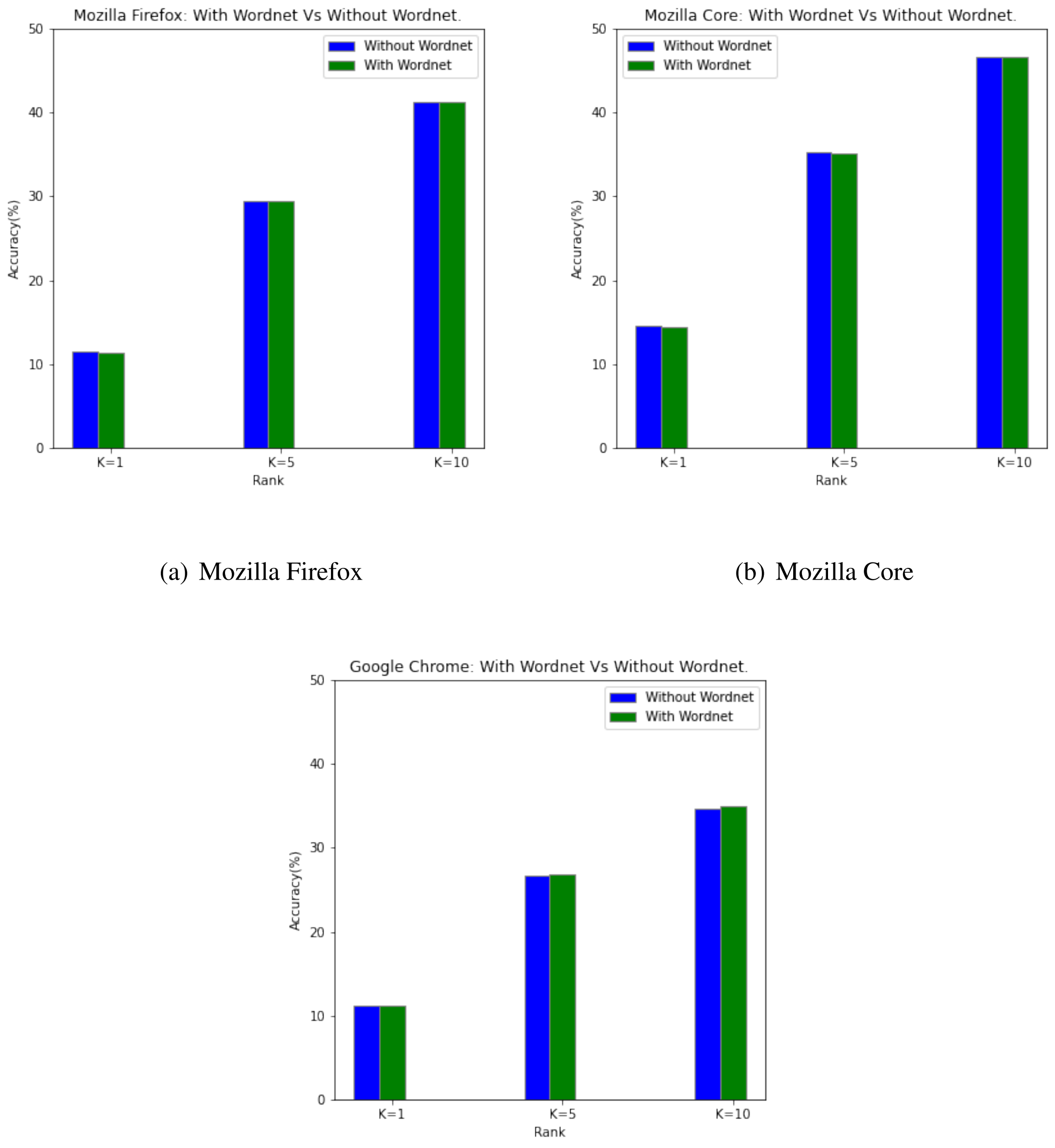
5.1.3 RQ#3: Does using WordNet improve the accuracy?

To answer this research question, we ran an experiment with the three datasets using Word2Vec with RF by setting the `random_state`[20] as 1. In this way, we ensured that the data model was the same for "with WordNet" and "without WordNet" experiments.

The Top-1, 5, and 10 accuracies for three separate datasets with and without WordNet are displayed in Figure 5.4. Accuracy(%) is plotted on the y-axis, and Top-K rank is plotted on the x-axis. Accuracy with or without using WordNet as a filter is shown by the green and blue bars, respectively. We can see from this figure and Table 5.3 that the accuracy is almost similar with and without the use of WordNet as a filter. Some datasets benefit from WordNet's improved accuracy, whereas others do not. However, in both cases, the accuracy

[20] Controls the sampling of the features to take into account when determining the optimum split at each node (if max features < n features), as well as the randomization of the bootstrapping of the samples, used to build trees (if bootstrap=True). For consistent results across several function calls, pass an int [57].

difference is smaller than 0.25%. We can therefore conclude that adding WordNet as a filter has no impact on the accuracy of bug report assignment recommender systems.



(a) Mozilla Firefox

(b) Mozilla Core

(c) Google Chromium

Figure 5.4: Top 1, 5 and 10 Accuracy: With WordNet Vs Without WordNet

Further analysis revealed that the utilization of WordNet's synonyms is less than 5% for the three datasets we used in this study. The reason may be that WordNet contains few

Table 5.3: Average accuracy over the cross-validation for Rank - 1, 5 and 10 are reported with or without using WordNet as a filter for RF.

Rank	**Mozilla Firefox**		**Mozilla Core**		**Google Chromium**	
	With WordNet	**Without WordNet**	**With WordNet**	**Without WordNet**	**With WordNet**	**Without WordNet**
Top-1	11.37	**11.53**	14.43	**14.49**	**11.23**	11.21
Top-5	**29.5**	29.49	35.17	**35.26**	**26.84**	26.62
Top-10	**41.23**	41.21	46.66	**46.67**	**34.91**	34.71

of the words frequently found in bug reports. Table 5.4 shows the average percentage of synonyms used per dataset.

Table 5.4: Average (%) of synonyms used in RF while filtering with WordNet.

Datasets	**Average Synonyms Used(%)**	
	Training	**Testing**
Mozilla Firefox	5.39	4.38
Mozilla Core	3.41	2.92
Google Chromium	1.27	1.15

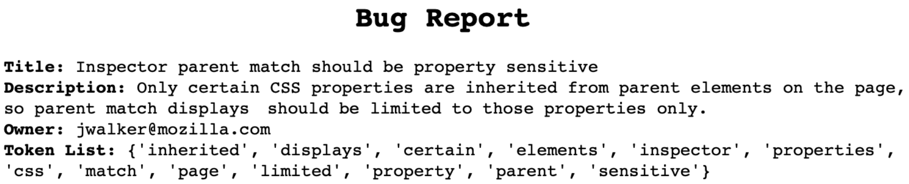
5.2 Explaining Assignment Recommendations

This section presents the results of our assessment of explaining the results using different local models and different visualizations.

5.2.1 RQ#4: Which local model explanation approach, feature-based or rule-based, provides better explanations?

To answer RQ#4, we choose LIME [12] as a feature-based approach and PyExplainer [13] as a rule-based approach. However, we discovered some limitations with PyExplainer, which prevented us from completing that portion of our investigation (see Section 6.1 for details). Therefore, we were only able to investigate the use of LIME and present our results of applying LIME to explain bug report assignment recommendations.

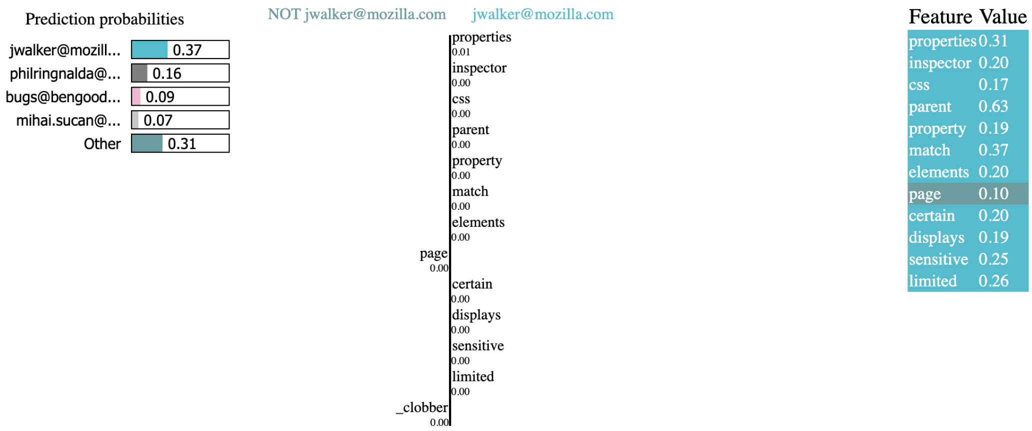
We examined the results for 6 randomly selected bug reports (see Appendix A) using an RF classifier trained using TF-IDF. Figure 5.5 is representative of these results, with Figure



Bug Report

Title: Inspector parent match should be property sensitive
Description: Only certain CSS properties are inherited from parent elements on the page, so parent match displays should be limited to those properties only.
Owner: jwalker@mozilla.com
Token List: {'inherited', 'displays', 'certain', 'elements', 'inspector', 'properties', 'css', 'match', 'page', 'limited', 'property', 'parent', 'sensitive'}

(a) Bug Report



Feature	Value
properties	0.31
inspector	0.20
css	0.17
parent	0.63
property	0.19
match	0.37
elements	0.20
page	0.10
certain	0.20
displays	0.19
sensitive	0.25
limited	0.26

(b) LIME Explanation

Figure 5.5: LIME Explanation for TF-IDF

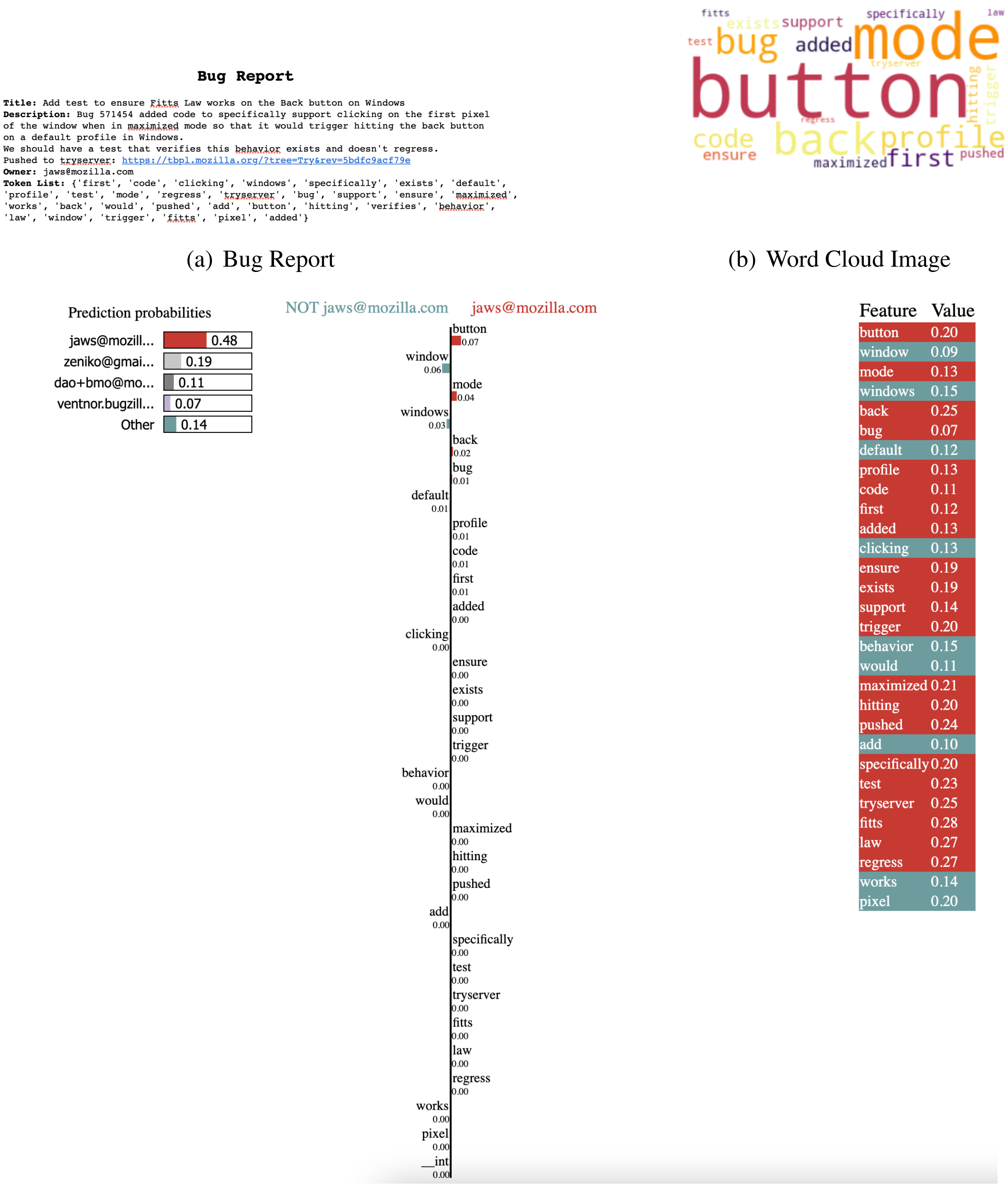
5.5(a) being the bug report description used and Figure 5.5(b) being the results from LIME. Figure 5.5(b)'s left side displays prediction probabilities using a vertical bar chart for the top 4 developers. The remaining developer names are grouped as "Others". The figure's central bar chart, known as a positive-negative horizontal bar chart, displays the contributions of the features for the prediction, which corresponds to the linear model's weights. On the right side, the values for the features are given with a colour code to indicate their relevance for specific developers. The colour coding is consistent across sections. It contains the values of the top 5 variables, the same as the vertical bar chart. The features displayed by LIME, such as "properties," "css," "inspector," and "parent," are members of the token list derived from the bug report shown in Figure 5.5(a). As a result, the user can make connections between the words of a particular bug report and the developers that LIME suggests.

5.2.2 RQ#5: Which visualization method appears to best support explanations of assignment recommendation: word cloud, positive-negative horizontal bar chart or feature table?

We generated three visualizations: a positive-negative horizontal bar chart, a feature table and a word cloud. The positive-negative horizontal bar chart and feature table were generated using LIME, and we created the word cloud using `WordCloud` package and feature frequencies generated by LIME. We examined the results for 15 randomly selected bug reports (see Appendix B) from the three datasets we used in our project. Figure 5.6 is representative of these results. Figure 5.6, respectively show these visualizations in turn. Based on our analysis of these figures, we noticed that although the positive-negative horizontal bar chart and feature table provide detailed information, they take a user more time and effort to comprehend which features are the most crucial for a recommendation. Since our feature size is limited to 50, we found that an image employing a word cloud enables the user to quickly comprehend the more pertinent features.

5.3 Summary

Based on the analysis of our evaluation results, we found that employing TF-IDF and Random Forest to develop a bug report assignment recommender system would be the best option, giving almost identical accuracy to the use of NN but requiring less training time. Due to the lack of software industry-related words in WordNet's lexical database, using WordNet has less impact on accuracy. For providing an explanation of the recommendations, a word cloud image was determined to be the best for giving a user a general understanding. If the user wants to dive deeper, a positive-negative bar chart and feature table could be made available.



Bug Report

Title: Add test to ensure Fitts Law works on the Back button on Windows
Description: Bug 571454 added code to specifically support clicking on the first pixel of the window when in maximized mode so that it would trigger hitting the back button on a default profile in Windows.
We should have a test that verifies this behavior exists and doesn't regress.
Pushed to tryserver: https://tbpl.mozilla.org/?tree=Try&rev=5bdfc9acf79e
Owner: jaws@mozilla.com
Token List: {'first', 'code', 'clicking', 'windows', 'specifically', 'exists', 'default', 'profile', 'test', 'mode', 'regress', 'tryserver', 'bug', 'support', 'ensure', 'maximized', 'works', 'back', 'would', 'pushed', 'add', 'button', 'hitting', 'verifies', 'behavior', 'law', 'window', 'trigger', 'fitts', 'pixel', 'added'}

(a) Bug Report

(b) Word Cloud Image

Feature	Value
button	0.20
window	0.09
mode	0.13
windows	0.15
back	0.25
bug	0.07
default	0.12
profile	0.13
code	0.11
first	0.12
added	0.13
clicking	0.13
ensure	0.19
exists	0.19
support	0.14
trigger	0.20
behavior	0.15
would	0.11
maximized	0.21
hitting	0.20
pushed	0.24
add	0.10
specifically	0.20
test	0.23
tryserver	0.25
fitts	0.28
law	0.27
regress	0.27
works	0.14
pixel	0.20

(c) LIME Explanation

Figure 5.6: Visual Explanation for a Bug Report

Chapter 6

Discussion

This chapter presents a discussion of the limitations discovered with using PyExplainer for bug report assignment recommendations, our recommendations for explanation visualization and threats to validity.

6.1 Limitations of PyExplainer

In conducting our investigation of the local model explanation, we selected PyExplainer as representative of a rule-based approach. However, we were not able to complete this investigation due to two discovered limitations with PyExplainer: a limit on the number of rules and a lack of support for recommender creation using BNN.

We used textual information, such as the title and description, from a bug report as our dataset. When we run this dataset through PyExplainer, it produces more than 15 rules due to the vast amount of features it contains. However, combining positive and negative rules, PyExplainer only supports up to 15 Top-K rules [58]. Therefore the limitation on the number of rules makes it inappropriate for use with a text classification problem. If instead of using bug report text as the feature set, the attributes of the bug report were used, then PyExplainer may work.

Additionally, because PyExplainer only supports the supervised classification models from `sklearn` [59], we were unable to train our third recommender system (BNN), which was provided by the `keras` library.

6.2 Problems explaining Word2Vec Recommender Systems

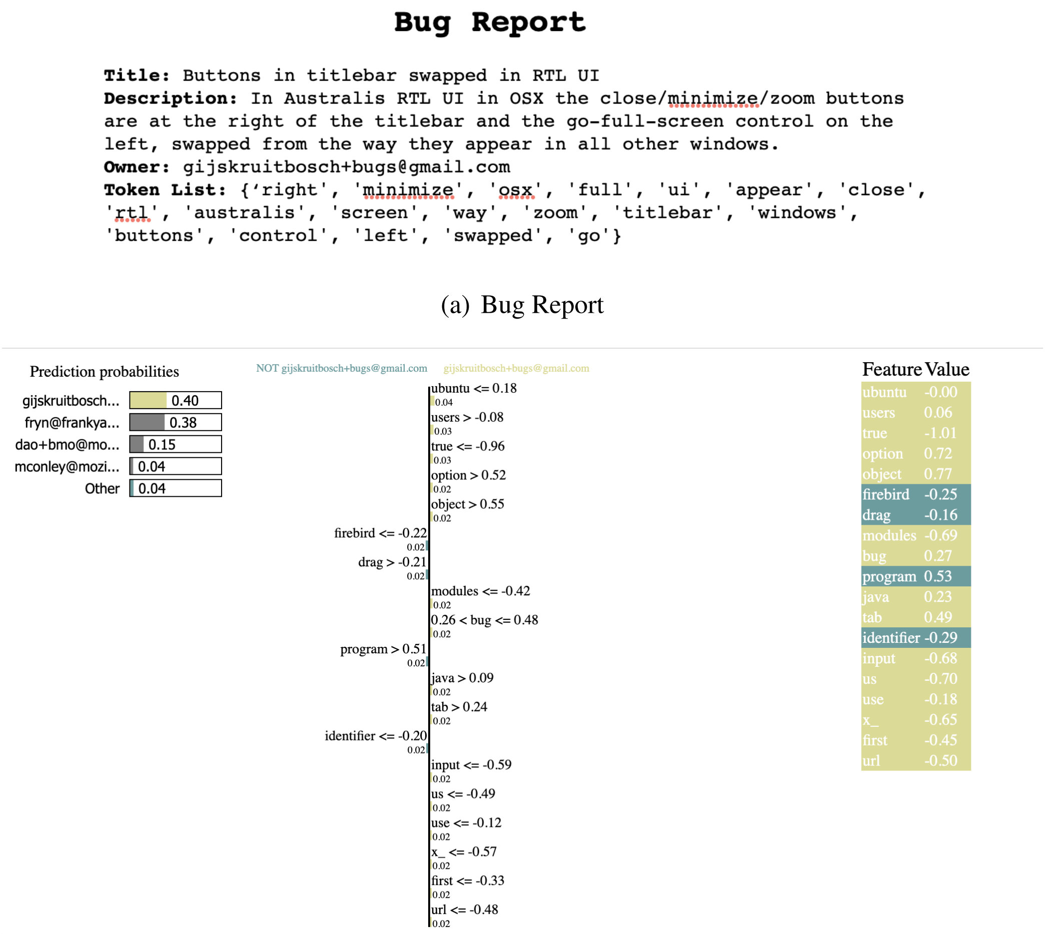
As a feature extraction method, we employed Word2Vec, which can identify a word's context inside a document, semantic and syntactic similarity, and relationship to other word sets. The Word2Vec models learn word associations using a neural network model. Therefore, Word2Vec based its recommendations on words that are similar but not necessarily the same as those in the bug report text. Consequently, there is a disconnect between the explanation of the recommendation and the bug report. Figure 6.1(b) shows how features from the LIME explanation, such as "ubuntu," "users," "true," and "option," are not present in the token list for the actual bug report, which is shown in Figure 6.1(a). As this disconnect does not occur with the use of TF-IDF, this technique was used when answering RQ#4 and RQ#5.

6.3 Creation of a bug report assignment recommender system

Based on the analysis of our evaluation results, we concluded that the best option would be to use TF-IDF and Random Forest to create a bug report assignment recommender system. This approach provides nearly identical accuracy as BNN while requiring less training time. Furthermore, Random Forest is less computationally expensive than Neural Networks.

6.4 Recommendations for visualization of explanation

In their work, Bhyan et al. [47] investigated the use of three visualizations to explain the results of a Multinomial Naive Bayes classifier for bug report assignment. The three visualizations examined were a pie chart, bar chart, and data table. According to their user study, there was a slight preference for the stacked bar chart over the pie chart, and participants also thought the data table to be more informative than the pie chart. Based on this work, we chose to investigate the use of feature tables, bar charts, and word clouds in our experiments.



Bug Report

Title: Buttons in titlebar swapped in RTL UI
Description: In Australis RTL UI in OSX the close/minimize/zoom buttons are at the right of the titlebar and the go-full-screen control on the left, swapped from the way they appear in all other windows.
Owner: gijskruitbosch+bugs@gmail.com
Token List: {'right', 'minimize', 'osx', 'full', 'ui', 'appear', 'close', 'rtl', 'australis', 'screen', 'way', 'zoom', 'titlebar', 'windows', 'buttons', 'control', 'left', 'swapped', 'go'}

(a) Bug Report

Feature	Value
ubuntu	-0.00
users	0.06
true	-1.01
option	0.72
object	0.77
firebird	-0.25
drag	-0.16
modules	-0.69
bug	0.27
program	0.53
java	0.23
tab	0.49
identifier	-0.29
input	-0.68
us	-0.70
use	-0.18
x_	-0.65
first	-0.45
url	-0.50

(b) LIME Explanation

Figure 6.1: LIME Explanation for Word2Vec

Similar to their results, we concluded that users could obtain a general impression of which features are more significant from the word cloud image. If the user wants a more detailed rationale, this can be done by looking at the positive-negative horizontal bar chart and feature tables. As a result, we recommend that the user be given a word cloud image to give them a broad explanation but that a positive-negative horizontal bar chart and feature table be made available for when the user wants to drill down more.

6.5 Threats to Validity

In this experiment, during the CV10 process, we were unable to ensure that the developers who fixed at least a predetermined number(5/10/20) of bug reports were only included in the training and testing datasets. This may have impacted the accuracy of the recommender systems.

We were unable to train recommenders using the entire dataset for Google Chrome and Mozilla Core due to memory limitations. This may have affected the accuracy of the recommenders. This may be especially true for MLP and BNN, as NN is known to have increased accuracy with more data.

When assessing the usefulness of different visualizations, we performed the assessment ourselves. A more thorough investigation involving a user study is needed to answer this research question more confidently.

Chapter 7

Conclusion

The goals of bug report triage are to evaluate, prioritize and assign the reports. A project member validates the bug reports according to the severity of the defect and assigns a developer to fix it if needed. For a given bug report, identifying an appropriate developer who could potentially fix the bug is a critical but time-consuming task. If the project receives a large number of bug reports daily, this process becomes overwhelming.

Researchers have investigated implementing different recommender systems to automate the bug report assignment process, which recommends an appropriate developer for a particular bug report. Such recommenders can help the project member to save their efforts. However, most of these systems used black-box models, which can have better accuracy. As a result, not knowing the rationale for a recommendation makes it difficult for the user to trust the process. Bhyan et al. investigated visualization for explaining bug report assignment recommenders and, from their user study, found that the visual explanations increased developers' understanding and helped them trust the process [47]. This dissertation further investigates the visualization of assignment recommendations.

In our work, we found that BNN created a more accurate bug report assignment recommender than MLP and RF. However, as the difference in accuracy between RF and the NN models was slim, we recommend using RF as it trains more quickly. We also found that the use of WordNet has no significant impact on bug report assignment recommender accuracy. Finally, we found that using TF-IDF instead of Word2Vec leads to better accuracy and explanation.

We tried to implement two local models, LIME and PyExplainer. However, we found that PyExplainer has limitations that make it unsuitable for use with the commonly used text-classification approach for bug report assignment recommender.

We compared images representing bug reports and their corresponding explanation using the visualizations produced by LIME (positive-negative horizontal bar chart and feature table) and word cloud. From this, we determined that the positive-negative horizontal bar chart and feature table generated by LIME give us more detailed information. However, as it takes time to understand which features are more important, we concluded that a word cloud image is better, as the user can understand easily and quickly which features are more important.

7.1 Future Work

Although we implemented three recommender systems and tried to explain them using LIME and word cloud, a number of future research directions were identified:

- Due to memory limitations, we were unable to use the entire dataset for Mozilla Core and Google Chromium for training, which can be done in future.
- Instead of using bug report text as the feature set, the attributes of the bug report can be used to implement PyExplainer as a local model.
- Extending PyExplainer to use a library that supports BNN.
- In the future, we can do a user study to determine how much this visualization actually helps the user. To do that, we can construct the recommender system with an explanation targeted to a certain project and test its effectiveness by asking predefined questions.

Bibliography

[1] S. Mani, A. Sankaran, and R. Aralikatte, "Deeptriage: Exploring the effectiveness of deep learning for bug triaging," *CoRR*, vol. abs/1801.01275, 2018.

[2] R. T. S, S. M. Patil, M. S, D. M, P. C. B. H, and D. S. K. S, "Recommendation of right developer for bug detection," *High Technology Letters*, vol. 27, p. 507–514, 2021.

[3] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Trans. on SE and Methodology*, vol. 20, 2011.

[4] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," *In International Conference on Software Engineering*, p. 25–35, 2012.

[5] P. Bhattacharya and I. Neamtiu, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging," *In International Conference on Software Maintenance*, pp. 1–10, 2010.

[6] J. Xuan, H. Jiang, Y. Hu, Z. Ren, W. Zou, Z. Luo, and X. Wu, "Towards effective bug triage with software data reduction techniques," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, p. 264–280, 2015.

[7] S. Wang, W. Zhang, and Q. Wang, "Fixercache: unsupervised caching active developers for diverse bug triage," *In ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, p. 25, 2014.

[8] L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh, and P. Runeson, "Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts," *Empirical Software Engineering*, vol. 21, p. 1533–1578, 2016.

[9] G. D. P. Regulation, "Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data and repealing directive 95/46," *Official J. Eur. Union*, vol. 59, no. 3, pp. 1–88, 2016.

[10] H. K. Dam, T. Tran, and A. Ghose, "Explainable software analytics," *Proc. Int. Conf. Softw. Eng.: New Ideas Emerg. Results*, pp. 53–56, 2018.

[11] K. W. Miller and D. K. Larson, "Agile software development: Human values and culture," *IEEE Technol. Soc. Mag.*, vol. 24, no. 4, pp. 36–42, 2005.

[12] M. T. Ribeiro, S. Singh, and C. Guestrin, “Why should i trust you?: Explaining the predictions of any classifier.,” *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. ACM*, 2016.

[13] C. Pornprasit, C. Tantithamthavorn, J. Jiarpakdee, M. Fu, and P. Thongtanunam, “Pyexplainer: Explaining the predictions of just-in-time defect models,” *In Proceedings of the International Conference on Automated Software Engineering (ASE)*, 2021.

[14] J. Jiarpakdee, C. K. Tantithamthavorn, H. K. Dam, and J. Grundy, “An empirical study of model-agnostic techniques for defect prediction models,” *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 166–185, 2022.

[15] L. Breiman, “Random forests,” *Machine Learning*, vol. 45.

[16] L. Breiman, “Bagging predictors,” *Machine Learning*, 1996.

[17] `https://www.knowledgehut.com/blog/data-science/bagging-and-random-forest-in-machine-learning`.

[18] `https://www.solver.com/xlminer/help/neural-networks-classification-intro`.

[19] `https://scikit-learn.org/stable/modules/neural_networks_supervised.html`.

[20] S. Siami-Namini, N. Tavakoli, and A. Namin, “The performance of lstm and bilstm in forecasting timeseries,” *In Proceedings of the 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA,USA*, p. 3285–3292, 2019.

[21] A. Graves and J. Schmidhuber, “Framewise phoneme classification with bidirectional lstm and other neuralnetwork architectures,” *Neural Netw.*, 2005.

[22] Z. Cui, R. Ke, Z. Pu, and Y. Wang, “Deep bidirectional and unidirectional lstm recurrent neural network for network-wide traffic speed prediction,” *ArXiv*, vol. arXiv/1801.02143, 2018.

[23] `https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-docu`

[24] D. Jurafsky and J. H. Martin, *Speech and Language Processing*. 2021. 3rd ed. draft.

[25] `https://code.google.com/archive/p/word2vec/`.

[26] G. A. Miller, “Wordnet: a lexical database for english,” *Communications of the ACM*, vol. 38, p. 39–41, 1995.

[27] T. Miller, “Explanation in artificial intelligence: insights from the social sciences,” *Artificial intelligence*, vol. 267, no. 1, p. 1–38, 2019.

[28] Z. C. Lipton, “The mythos of model interpretability: in machine learning, the concept of interpretability is both important and slippery,” *Queue*, vol. 16, no. 3, p. 31–57, 2018.

[29] B. Y. Lim, Q. Yang, A. M. Abdul, and D. Wang, "Why these explanations? selecting intelligibility types for explanation goals," *IUI Workshops*, 2019.

[30] S. French, "Decisionanalysis," *Wiley StatsRef: Statistics Reference Online, Hoboken, NJ, USA:Wiley*, 2014.

[31] J. R. Quinlan, "Simplifying decision trees," *Int. J. Man-Mach. Stud.*, vol. 27, no. 3, pp. 221–234, 1987.

[32] J. H. Friedman and B. E. Popescu, "Predictive learning via rule ensembles," *The Annals of Applied Statistics*, vol. 2, p. 916–954, 2008.

[33] C. Molnar, *Interpretable Machine Learning*. 2019.

[34] `https://towardsdatascience.com/understanding-model-predictions-with-lime-a582f`

[35] C. Tantithamthavorn and J. Jiarpakdee, *Explainable AI for Software Engineering*. Monash University, 2021. Retrieved 2021-05-17.

[36] `https://en.wikipedia.org/wiki/Tag_cloud`.

[37] A. Tamrawi, T. T. Nguyen, J. Al-Kofahi, and T. N. Nguyen, "Fuzzy set-based automatic bug triaging: Nier track," *In International Conference on Software Engineering*, p. 884–887, 2011.

[38] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation," *In Working Conference on Mining Software Repositories*, pp. 2–11, 2013.

[39] A. S. Badashian, A. Hindle, and E. Stroulia, "Crowdsourced bug triaging," *In International Conference on Software Maintenance and Evolution*, pp. 506–510, 2015.

[40] H. Alharthi and D. Inkpen, "Content-based recommender system enriched with wordnet synsets," in *CICLing*, 2015.

[41] B. Magnini and C. Strapparavar, "Using wordnet to improve user modelling in a web document recommender system," *In Proceedings of the NAACL 2001 Workshop on WordNet and Other Lexical Resources*, 2001.

[42] J. Jiarpakdee, C. Tantithamthavorn, H. K. Dam, and J. Grundy, "An empirical study of model-agnostic techniques for defect prediction models," *IEEE Transactions on Software Engineering (TSE)*, 2020.

[43] C. Pornprasit and C. Tantithamthavorn, "Jitline: A simpler, better, faster, finer-grained just-in-time defect prediction," *in Proceedings of the International Conference on Mining Software Repositories (MSR)*, 2021.

[44] S. Wattanakriengkrai, P. Thongtanunam, C. Tantithamthavorn, H. Hata, and K. Matsumoto, "Predicting defective lines using a model-agnostic technique," *IEEE Transactions on Software Engineering (TSE)*, 2020.

[45] S. Amal, M. Adam, P. Brusilovsky, E. Minkov, and T. Kuflik, "Enhancing explainability of social recommendation using 2d graphs and word cloud visualizations," *International Conference on Intelligent User Interfaces*, pp. 21–22, 2019.

[46] N. Liu, Y. Ge, L. Li, X. Hu, R. Chen, and S.-H. Choi, "Explainable recommender systems via resolving learning representations," *ACM International Conference on Information and Knowledge Management*, p. 895–904.

[47] S. A. Bhyan and J. Anvik, "Evaluating visual explanation of bug report assignment recommendations," *International Conference on Software Engineering & Knowledge Engineering*, 2021.

[48] N. Xie, G. Ras, M. van Gerven, and D. Doran, "Explainable deep learning: A field guide for the uninitiated," *ArXiv*, vol. abs/2004.14545, 2020.

[49] A. Wan, L. Dunlap, D. Ho, J. Yin, S. Lee, S. Petryk, S. A. Bargal, and J. Gonzalez, "Nbdt: Neural-backed decision tree," in *ICLR*, 2021.

[50] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful ... really?," *In International conference on Software maintenance*, p. 337–345, 2008.

[51] Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[52] `https://scikit-learn.org/stable/modules/ensemble.html#forest`.

[53] `https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier`.

[54] `https://lime-ml.readthedocs.io/en/latest/lime.html`.

[55] `https://radimrehurek.com/gensim/models/word2vec.html`.

[56] R. K. et al, "A study of cross-validation and bootstrap for accuracy estimation and model selection.," *17th Inter. Conf. on Eval. and Assess. in SE*, vol. 14, p. 1137–1145, 1995.

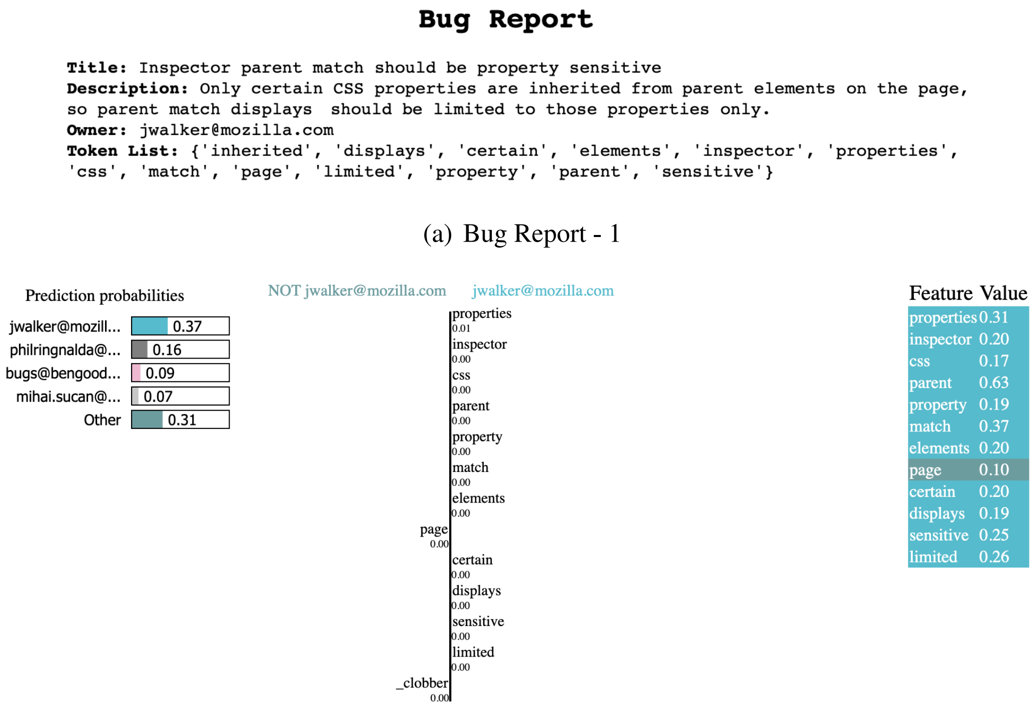
[57] `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`.

[58] `https://github.com/awsm-research/PyExplainer/blob/master/pyexplainer/pyexplainer_pyexplainer.pypyexplainerTutorial`.

[59] `https://github.com/awsm-research/PyExplainer/blob/master/TUTORIAL.ipynb`.

Appendix A

LIME Explanation for a Bug Report Assignment Recommender System using TF-IDF



Bug Report

```
Title: Inspector parent match should be property sensitive
Description: Only certain CSS properties are inherited from parent elements on the page,
so parent match displays  should be limited to those properties only.
Owner: jwalker@mozilla.com
Token List: {'inherited', 'displays', 'certain', 'elements', 'inspector', 'properties',
'css', 'match', 'page', 'limited', 'property', 'parent', 'sensitive'}
```

(a) Bug Report - 1

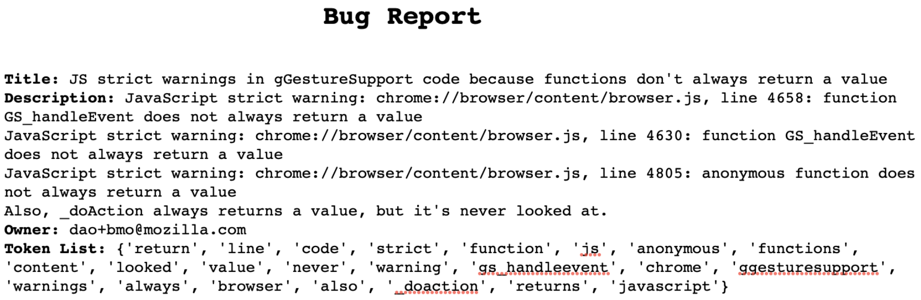
Prediction probabilities

jwalker@mozill...	0.37
philringnalda@...	0.16
bugs@bengood...	0.09
mihai.sucan@...	0.07
Other	0.31

Feature	Value
properties	0.31
inspector	0.20
css	0.17
parent	0.63
property	0.19
match	0.37
elements	0.20
page	0.10
certain	0.20
displays	0.19
sensitive	0.25
limited	0.26

(b) LIME Explanation

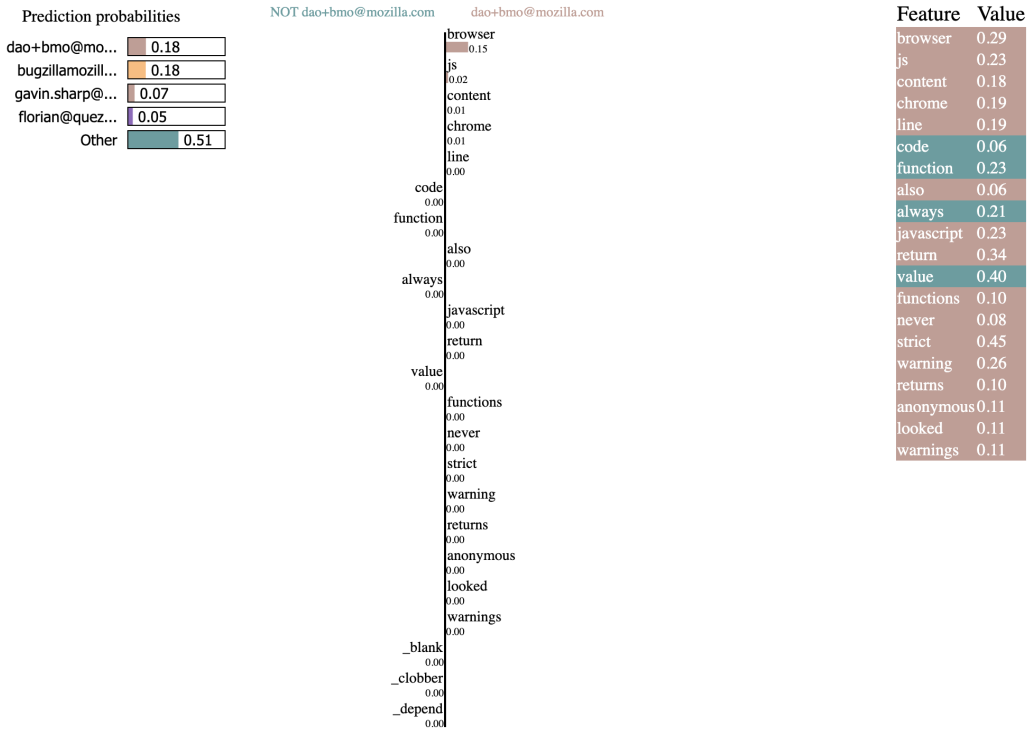
Figure A.1: Bug Report - 1 : LIME Explanation for TF-IDF



Bug Report

```
Title: JS strict warnings in gGestureSupport code because functions don't always return a value
Description: JavaScript strict warning: chrome://browser/content/browser.js, line 4658: function
GS_handleEvent does not always return a value
JavaScript strict warning: chrome://browser/content/browser.js, line 4630: function GS_handleEvent
does not always return a value
JavaScript strict warning: chrome://browser/content/browser.js, line 4805: anonymous function does
not always return a value
Also, _doAction always returns a value, but it's never looked at.
Owner: dao+bmo@mozilla.com
Token List: {'return', 'line', 'code', 'strict', 'function', 'js', 'anonymous', 'functions',
'content', 'looked', 'value', 'never', 'warning', 'gs_handleevent', 'chrome', 'ggesturesupport',
'warnings', 'always', 'browser', 'also', '_doaction', 'returns', 'javascript'}
```

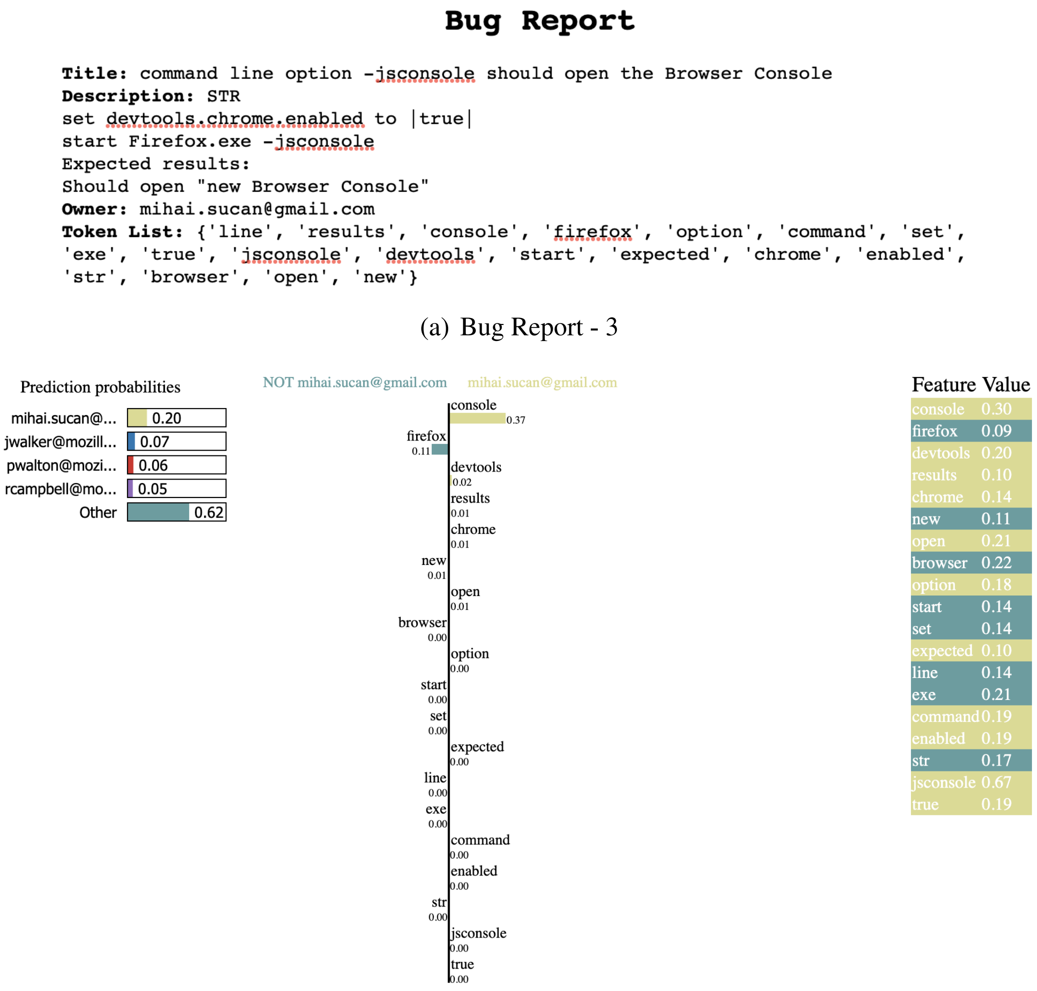
(a) Bug Report - 2



Feature	Value
browser	0.29
js	0.23
content	0.18
chrome	0.19
line	0.19
code	0.06
function	0.23
also	0.06
always	0.21
javascript	0.23
return	0.34
value	0.40
functions	0.10
never	0.08
strict	0.45
warning	0.26
returns	0.10
anonymous	0.11
looked	0.11
warnings	0.11

(b) LIME Explanation

Figure A.2: Bug Report - 2 : LIME Explanation for TF-IDF



Bug Report

Title: command line option -jsconsole should open the Browser Console
Description: STR
set devtools.chrome.enabled to |true|
start Firefox.exe -jsconsole
Expected results:
Should open "new Browser Console"
Owner: mihai.sucan@gmail.com
Token List: {'line', 'results', 'console', 'firefox', 'option', 'command', 'set', 'exe', 'true', 'jsconsole', 'devtools', 'start', 'expected', 'chrome', 'enabled', 'str', 'browser', 'open', 'new'}

(a) Bug Report - 3

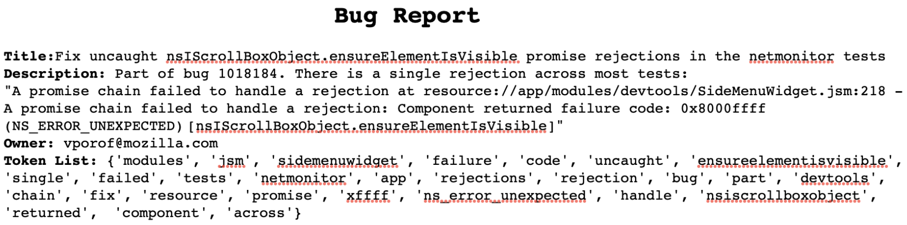
Prediction probabilities

mihai.sucan@...	0.20
jwalker@mozill...	0.07
pwalton@mozi...	0.06
rcampbell@mo...	0.05
Other	0.62

Feature	Value
console	0.30
firefox	0.09
devtools	0.20
results	0.10
chrome	0.14
new	0.11
open	0.21
browser	0.22
option	0.18
start	0.14
set	0.14
expected	0.10
line	0.14
exe	0.21
command	0.19
enabled	0.19
str	0.17
jsconsole	0.67
true	0.19

(b) LIME Explanation

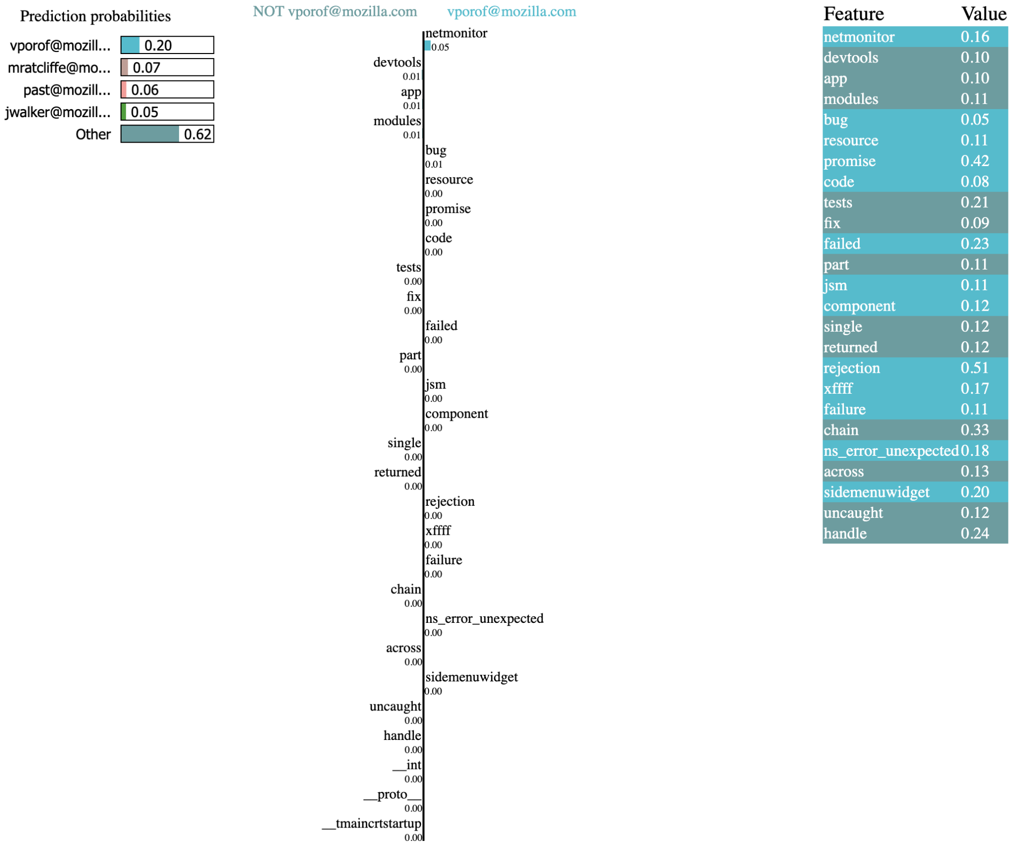
Figure A.3: Bug Report - 3: LIME Explanation for TF-IDF



Bug Report

Title: Fix uncaught nsIScrollBoxObject.ensureElementIsVisible promise rejections in the netmonitor tests
Description: Part of bug 1018184. There is a single rejection across most tests:
"A promise chain failed to handle a rejection at resource://app/modules/devtools/SideMenuWidget.jsm:218 - A promise chain failed to handle a rejection: Component returned failure code: 0x8000ffff (NS_ERROR_UNEXPECTED) [nsIScrollBoxObject.ensureElementIsVisible]"
Owner: vporof@mozilla.com
Token List: {'modules', 'jsm', 'sidemenuwidget', 'failure', 'code', 'uncaught', 'ensureelementisvisible', 'single', 'failed', 'tests', 'netmonitor', 'app', 'rejections', 'rejection', 'bug', 'part', 'devtools', 'chain', 'fix', 'resource', 'promise', 'xffff', 'ns_error_unexpected', 'handle', 'nsiscrollboxobject', 'returned', 'component', 'across'}

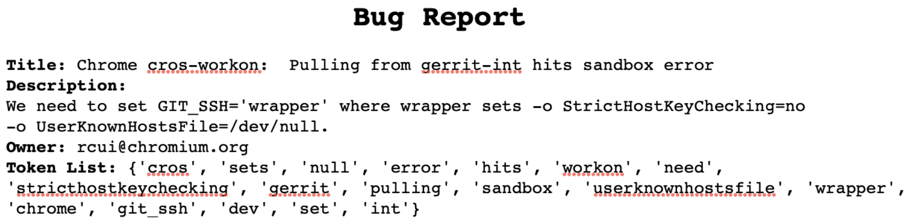
(a) Bug Report - 4



Feature	Value
netmonitor	0.16
devtools	0.10
app	0.10
modules	0.11
bug	0.05
resource	0.11
promise	0.42
code	0.08
tests	0.21
fix	0.09
failed	0.23
part	0.11
jsm	0.11
component	0.12
single	0.12
returned	0.12
rejection	0.51
xffff	0.17
failure	0.11
chain	0.33
ns_error_unexpected	0.18
across	0.13
sidemenuwidget	0.20
uncaught	0.12
handle	0.24

(b) LIME Explanation

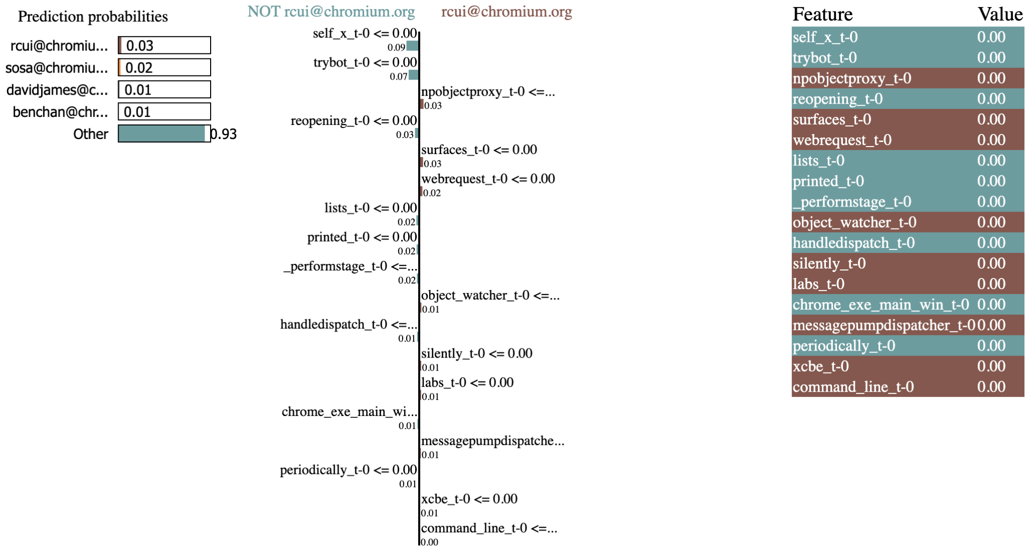
Figure A.4: Bug Report - 4: LIME Explanation for TF-IDF



Bug Report

Title: Chrome cros-workon: Pulling from gerrit-int hits sandbox error
Description:
We need to set GIT_SSH='wrapper' where wrapper sets -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null.
Owner: rcui@chromium.org
Token List: {'cros', 'sets', 'null', 'error', 'hits', 'workon', 'need', 'stricthostkeychecking', 'gerrit', 'pulling', 'sandbox', 'userknownhostsfile', 'wrapper', 'chrome', 'git_ssh', 'dev', 'set', 'int'}

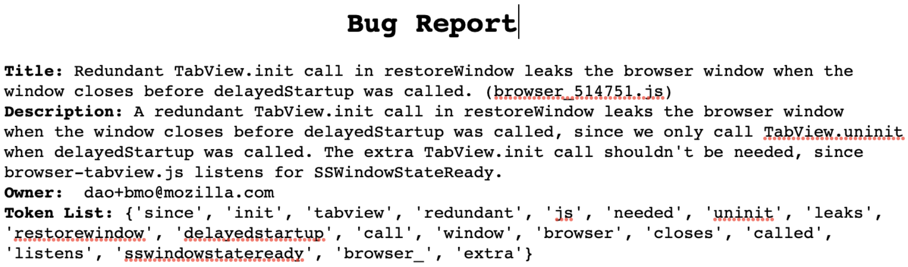
(a) Bug Report - 5



Feature	Value
self_x_t-0	0.00
trybot_t-0	0.00
npobjectproxy_t-0	0.00
reopening_t-0	0.00
surfaces_t-0	0.00
webrequest_t-0	0.00
lists_t-0	0.00
printed_t-0	0.00
_performstage_t-0	0.00
object_watcher_t-0	0.00
handledispatch_t-0	0.00
silently_t-0	0.00
labs_t-0	0.00
chrome_exe_main_win_t-0	0.00
messagepumpdispatcher_t-0	0.00
periodically_t-0	0.00
xcbe_t-0	0.00
command_line_t-0	0.00

(b) LIME Explanation

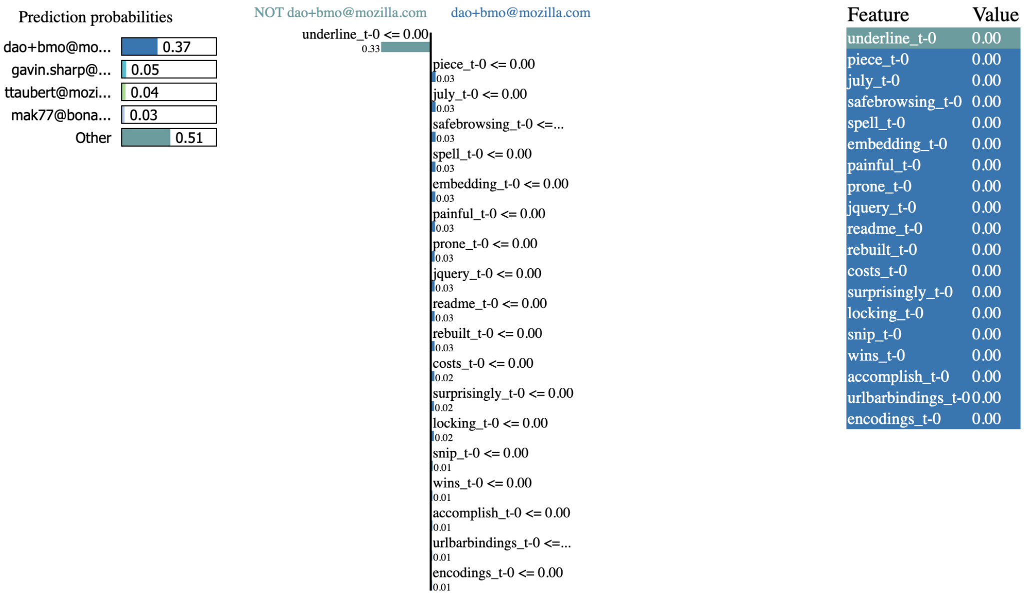
Figure A.5: Bug Report - 5: LIME Explanation for TF-IDF



Bug Report

Title: Redundant TabView.init call in restoreWindow leaks the browser window when the window closes before delayedStartup was called. (browser_514751.js)
Description: A redundant TabView.init call in restoreWindow leaks the browser window when the window closes before delayedStartup was called, since we only call TabView.uninit when delayedStartup was called. The extra TabView.init call shouldn't be needed, since browser-tabview.js listens for SSWindowStateReady.
Owner: dao+bmo@mozilla.com
Token List: {'since', 'init', 'tabview', 'redundant', 'js', 'needed', 'uninit', 'leaks', 'restorewindow', 'delayedstartup', 'call', 'window', 'browser', 'closes', 'called', 'listens', 'sswindowstateready', 'browser_', 'extra'}

(a) Bug Report - 6



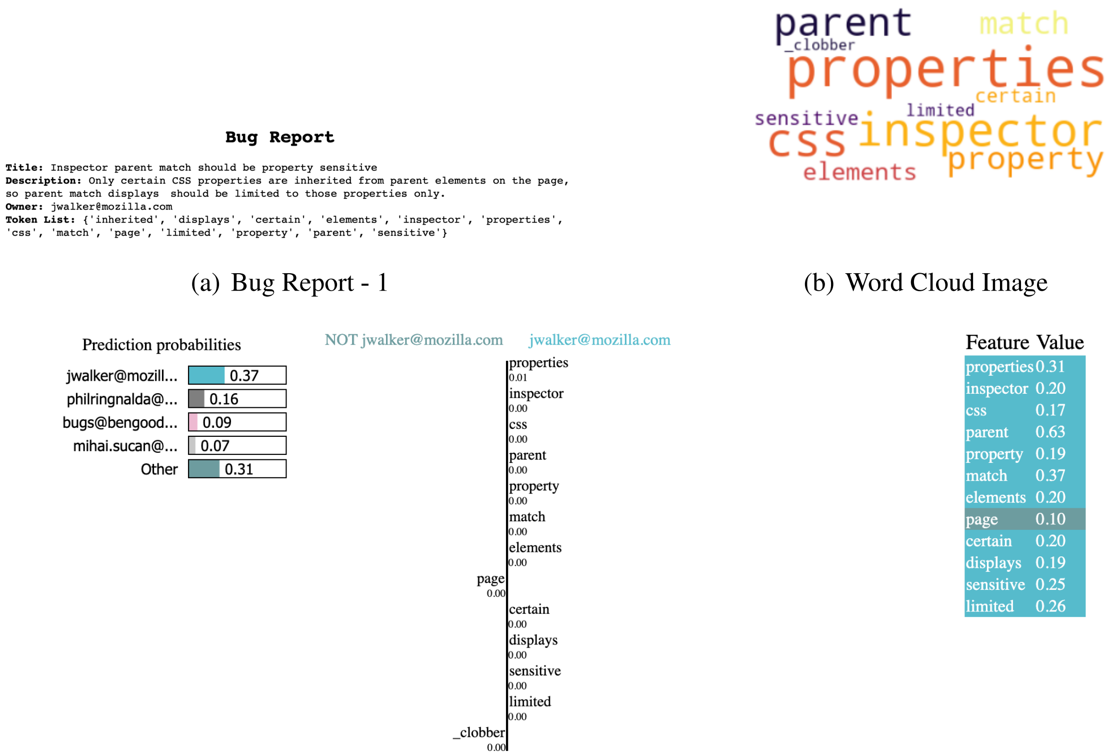
Feature	Value
underline_t-0	0.00
piece_t-0	0.00
july_t-0	0.00
safebrowsing_t-0	0.00
spell_t-0	0.00
embedding_t-0	0.00
painful_t-0	0.00
prone_t-0	0.00
jquery_t-0	0.00
readme_t-0	0.00
rebuilt_t-0	0.00
costs_t-0	0.00
surprisingly_t-0	0.00
locking_t-0	0.00
snip_t-0	0.00
wins_t-0	0.00
accomplish_t-0	0.00
urlbarbindings_t-0	0.00
encodings_t-0	0.00

(b) LIME Explanation

Figure A.6: Bug Report - 6: LIME Explanation for TF-IDF

Appendix B

Visual Explanation for Bug Report Assignment Recommender System



Bug Report

Title: Inspector parent match should be property sensitive
Description: Only certain CSS properties are inherited from parent elements on the page, so parent match displays should be limited to those properties only.
Owner: jwalker@mozilla.com
Token List: {'inherited', 'displays', 'certain', 'elements', 'inspector', 'properties', 'css', 'match', 'page', 'limited', 'property', 'parent', 'sensitive'}

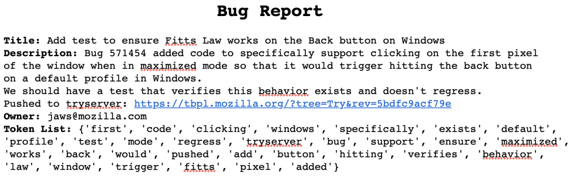
(a) Bug Report - 1

(b) Word Cloud Image

Feature	Value
properties	0.31
inspector	0.20
css	0.17
parent	0.63
property	0.19
match	0.37
elements	0.20
page	0.10
certain	0.20
displays	0.19
sensitive	0.25
limited	0.26

(c) LIME Explanation

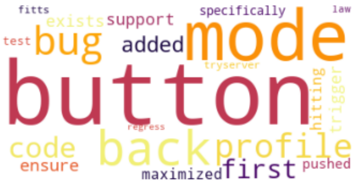
Figure B.1: Visual Explanation for Bug Report 1



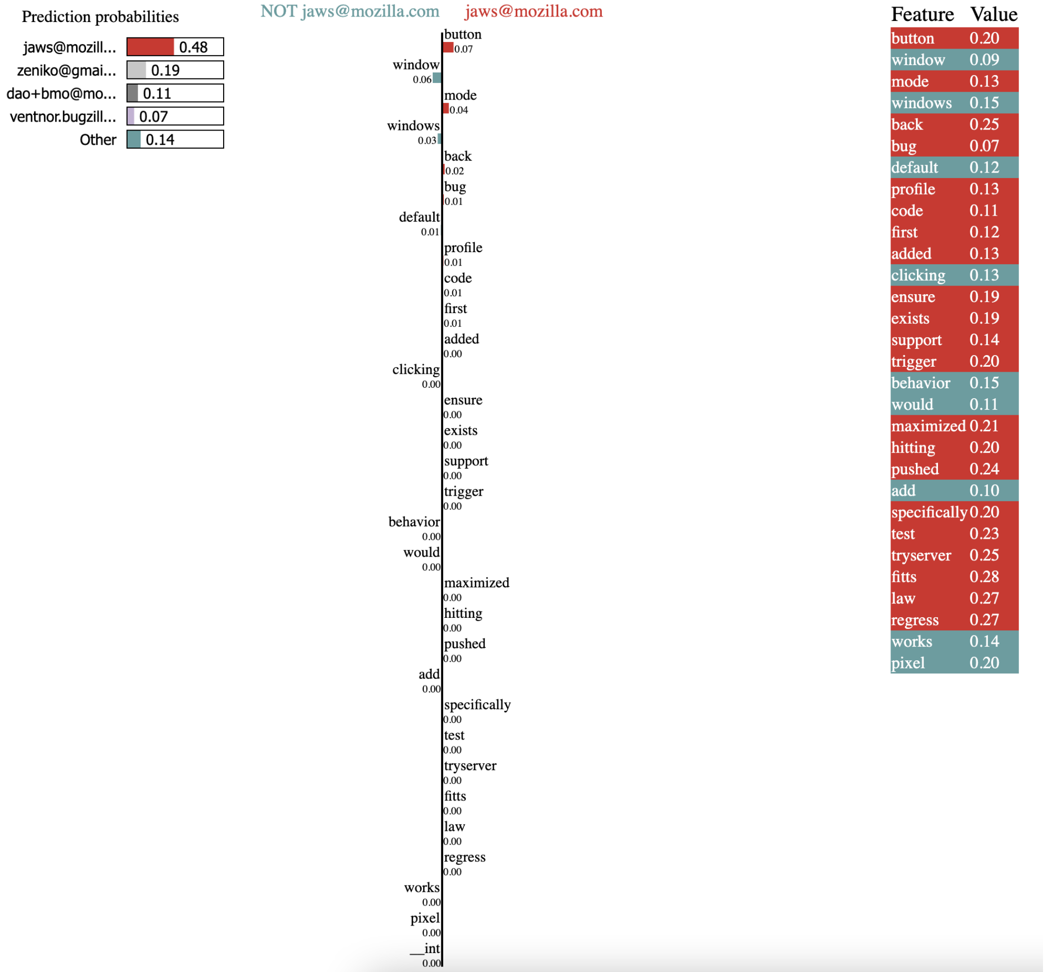
Bug Report

Title: Add test to ensure Fitts Law works on the Back button on Windows
Description: Bug 571454 added code to specifically support clicking on the first pixel of the window when in maximized mode so that it would trigger hitting the back button on a default profile in Windows.
We should have a test that verifies this behavior exists and doesn't regress.
Pushed to tryserver: https://tbpl.mozilla.org/?tree=Try&rev=5bdfc9acf79e
Owner: jaws@mozilla.com
Token List: {'first', 'code', 'clicking', 'windows', 'specifically', 'exists', 'default', 'profile', 'test', 'mode', 'regress', 'tryserver', 'bug', 'support', 'ensure', 'maximized', 'works', 'back', 'would', 'pushed', 'add', 'button', 'hitting', 'verifies', 'behavior', 'law', 'window', 'trigger', 'fitts', 'pixel', 'added'}

(a) Bug Report - 2



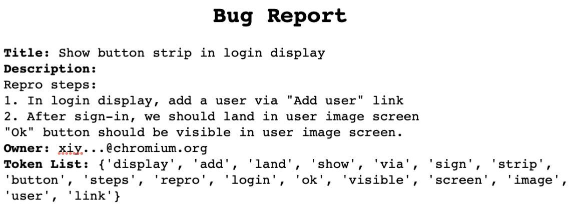
(b) Word Cloud Image



Feature	Value
button	0.20
window	0.09
mode	0.13
windows	0.15
back	0.25
bug	0.07
default	0.12
profile	0.13
code	0.11
first	0.12
added	0.13
clicking	0.13
ensure	0.19
exists	0.19
support	0.14
trigger	0.20
behavior	0.15
would	0.11
maximized	0.21
hitting	0.20
pushed	0.24
add	0.10
specifically	0.20
test	0.23
tryserver	0.25
fitts	0.28
law	0.27
regress	0.27
works	0.14
pixel	0.20

(c) LIME Explanation

Figure B.2: Visual Explanation for Bug Report 2



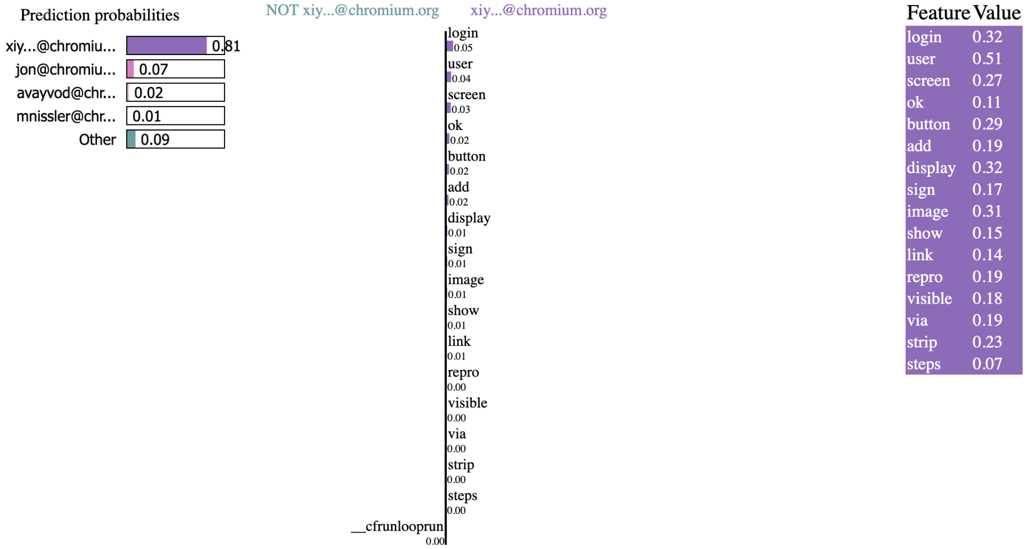
Bug Report

Title: Show button strip in login display
Description:
Repro steps:
1. In login display, add a user via "Add user" link
2. After sign-in, we should land in user image screen
"Ok" button should be visible in user image screen.
Owner: xiy...@chromium.org
Token List: {'display', 'add', 'land', 'show', 'via', 'sign', 'strip', 'button', 'steps', 'repro', 'login', 'ok', 'visible', 'screen', 'image', 'user', 'link'}

(a) Bug Report - 3

add
extension
script
object
folks
csp
fully
limit
objects
want
rather
hosts
https
scripts
manifest
attribute

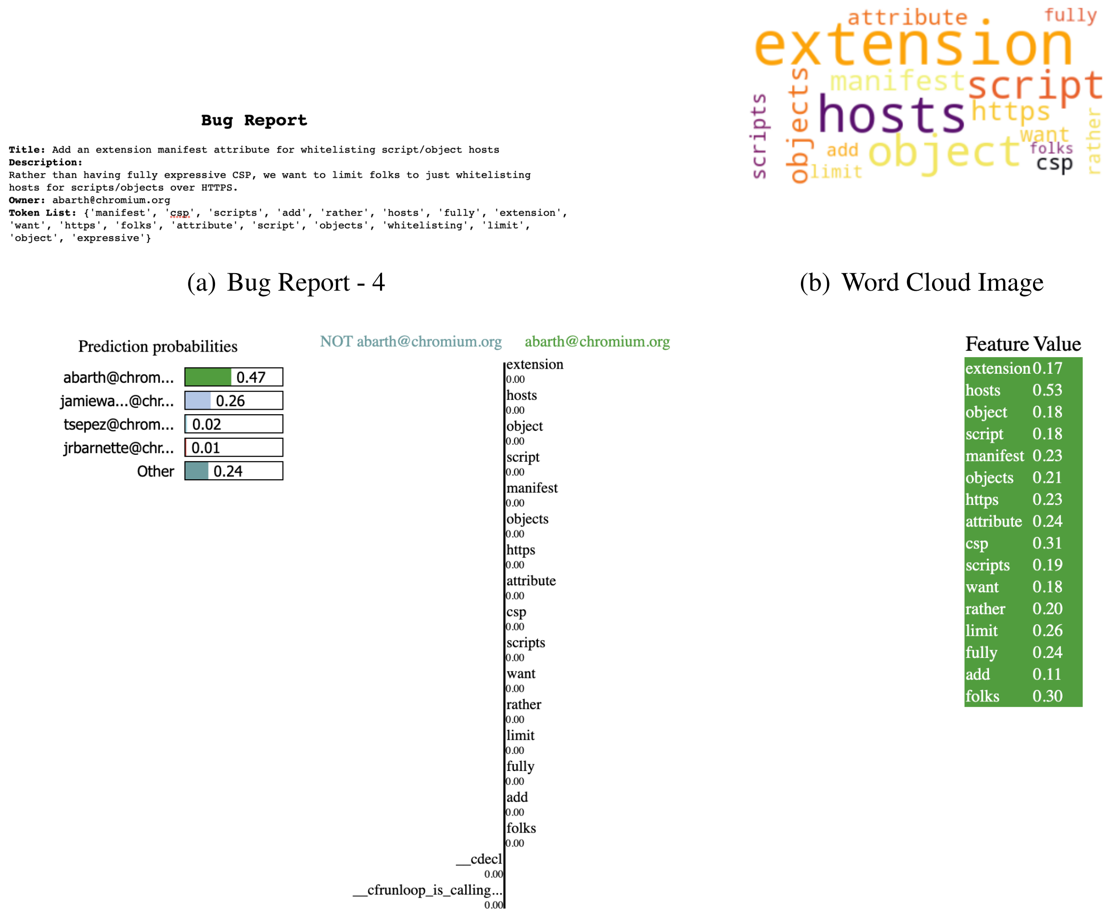
(b) Word Cloud Image



Feature	Value
login	0.32
user	0.51
screen	0.27
ok	0.11
button	0.29
add	0.19
display	0.32
sign	0.17
image	0.31
show	0.15
link	0.14
repro	0.19
visible	0.18
via	0.19
strip	0.23
steps	0.07

(c) LIME Explanation

Figure B.3: Visual Explanation for Bug Report 3



Bug Report

Title: Add an extension manifest attribute for whitelisting script/object hosts
Description:
Rather than having fully expressive CSP, we want to limit folks to just whitelisting hosts for scripts/objects over HTTPS.
Owner: abarth@chromium.org
Token List: {'manifest', 'csp', 'scripts', 'add', 'rather', 'hosts', 'fully', 'extension', 'want', 'https', 'folks', 'attribute', 'script', 'objects', 'whitelisting', 'limit', 'object', 'expressive'}

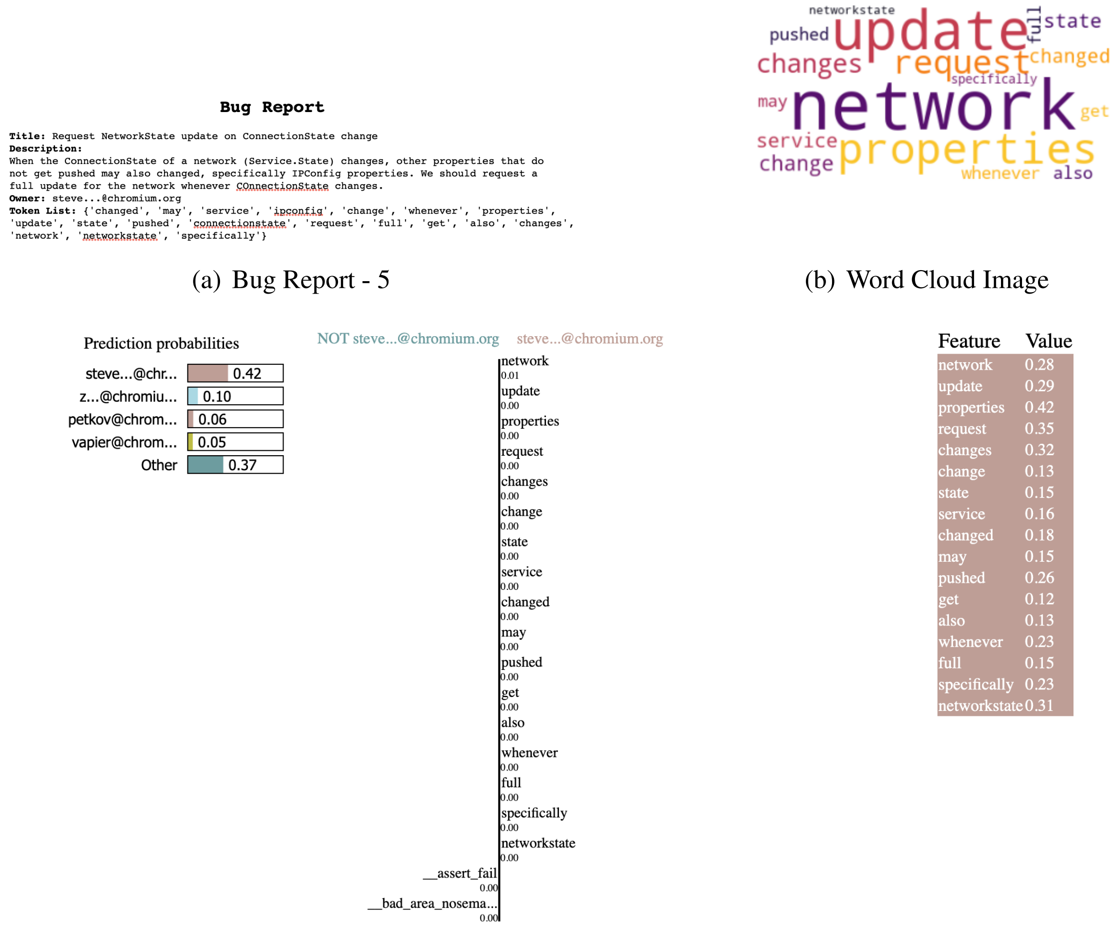
(a) Bug Report - 4

(b) Word Cloud Image

Feature	Value
extension	0.17
hosts	0.53
object	0.18
script	0.18
manifest	0.23
objects	0.21
https	0.23
attribute	0.24
csp	0.31
scripts	0.19
want	0.18
rather	0.20
limit	0.26
fully	0.24
add	0.11
folks	0.30

(c) LIME Explanation

Figure B.4: Visual Explanation for Bug Report 4



Bug Report

Title: Request NetworkState update on ConnectionState change
Description:
When the ConnectionState of a network (Service.State) changes, other properties that do not get pushed may also changed, specifically IPConfig properties. We should request a full update for the network whenever COnnectionState changes.
Owner: steve...@chromium.org
Token List: {'changed', 'may', 'service', 'ipconfig', 'change', 'whenever', 'properties', 'update', 'state', 'pushed', 'connectionstate', 'request', 'full', 'get', 'also', 'changes', 'network', 'networkstate', 'specifically'}

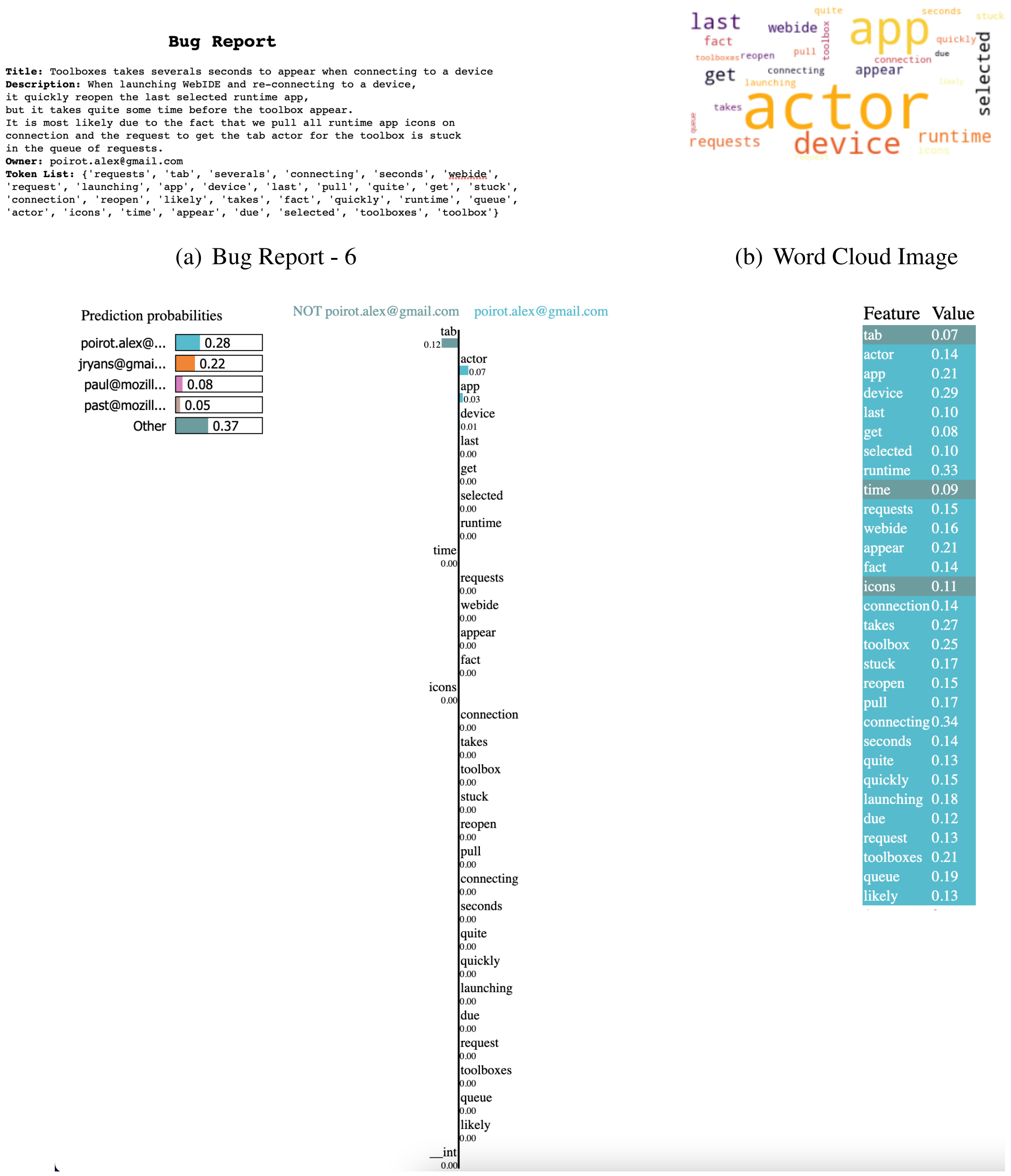
(a) Bug Report - 5

(b) Word Cloud Image

Feature	Value
network	0.28
update	0.29
properties	0.42
request	0.35
changes	0.32
change	0.13
state	0.15
service	0.16
changed	0.18
may	0.15
pushed	0.26
get	0.12
also	0.13
whenever	0.23
full	0.15
specifically	0.23
networkstate	0.31

(c) LIME Explanation

Figure B.5: Visual Explanation for Bug Report 5



Bug Report

```
Title: Toolboxes takes severals seconds to appear when connecting to a device
Description: When launching WebIDE and re-connecting to a device,
it quickly reopen the last selected runtime app,
but it takes quite some time before the toolbox appear.
It is most likely due to the fact that we pull all runtime app icons on
connection and the request to get the tab actor for the toolbox is stuck
in the queue of requests.
Owner: poirot.alex@gmail.com
Token List: {'requests', 'tab', 'severals', 'connecting', 'seconds', 'webide',
'request', 'launching', 'app', 'device', 'last', 'pull', 'quite', 'get', 'stuck',
'connection', 'reopen', 'likely', 'takes', 'fact', 'quickly', 'runtime', 'queue',
'actor', 'icons', 'time', 'appear', 'due', 'selected', 'toolboxes', 'toolbox'}
```

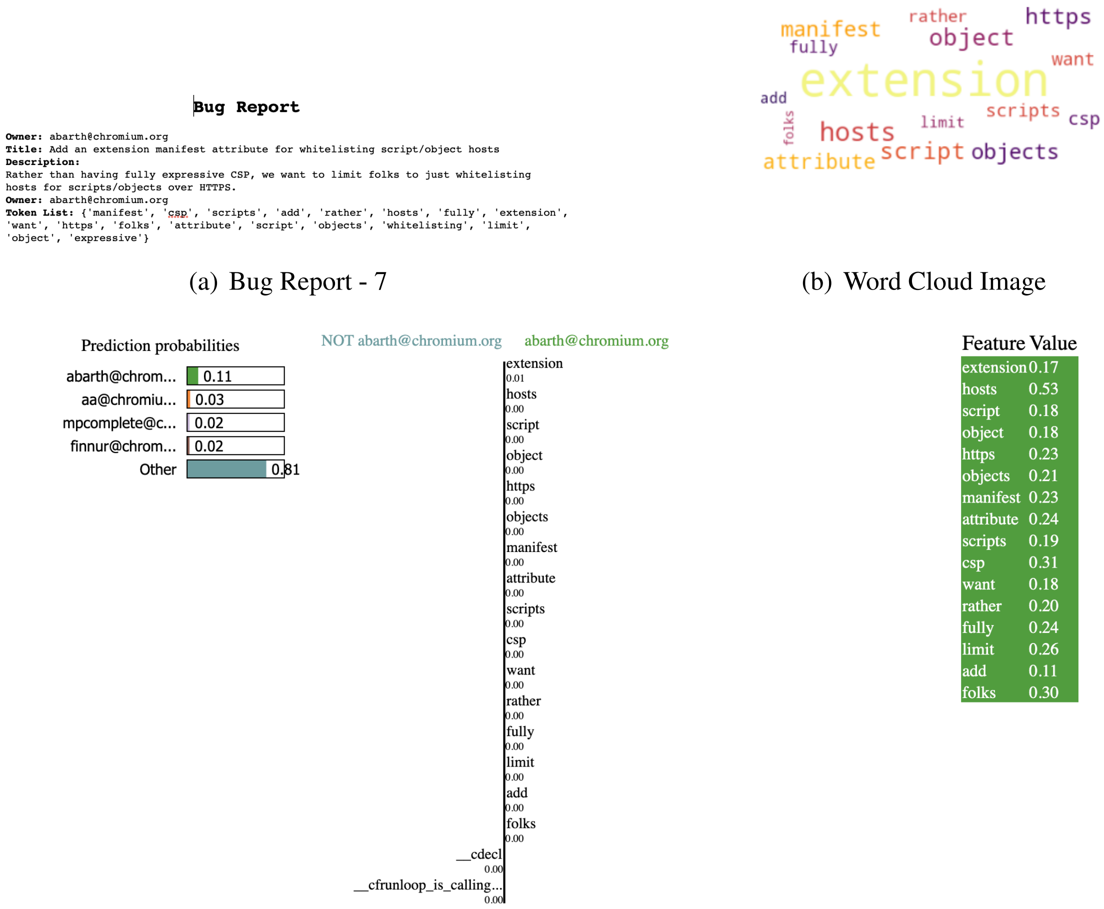
(a) Bug Report - 6

(b) Word Cloud Image

Feature	Value
tab	0.07
actor	0.14
app	0.21
device	0.29
last	0.10
get	0.08
selected	0.10
runtime	0.33
time	0.09
requests	0.15
webide	0.16
appear	0.21
fact	0.14
icons	0.11
connection	0.14
takes	0.27
toolbox	0.25
stuck	0.17
reopen	0.15
pull	0.17
connecting	0.34
seconds	0.14
quite	0.13
quickly	0.15
launching	0.18
due	0.12
request	0.13
toolboxes	0.21
queue	0.19
likely	0.13

(c) LIME Explanation

Figure B.6: Visual Explanation for Bug Report 6



Bug Report

Owner: abarth@chromium.org
Title: Add an extension manifest attribute for whitelisting script/object hosts
Description:
Rather than having fully expressive CSP, we want to limit folks to just whitelisting hosts for scripts/objects over HTTPS.
Owner: abarth@chromium.org
Token List: {'manifest', 'csp', 'scripts', 'add', 'rather', 'hosts', 'fully', 'extension', 'want', 'https', 'folks', 'attribute', 'script', 'objects', 'whitelisting', 'limit', 'object', 'expressive'}

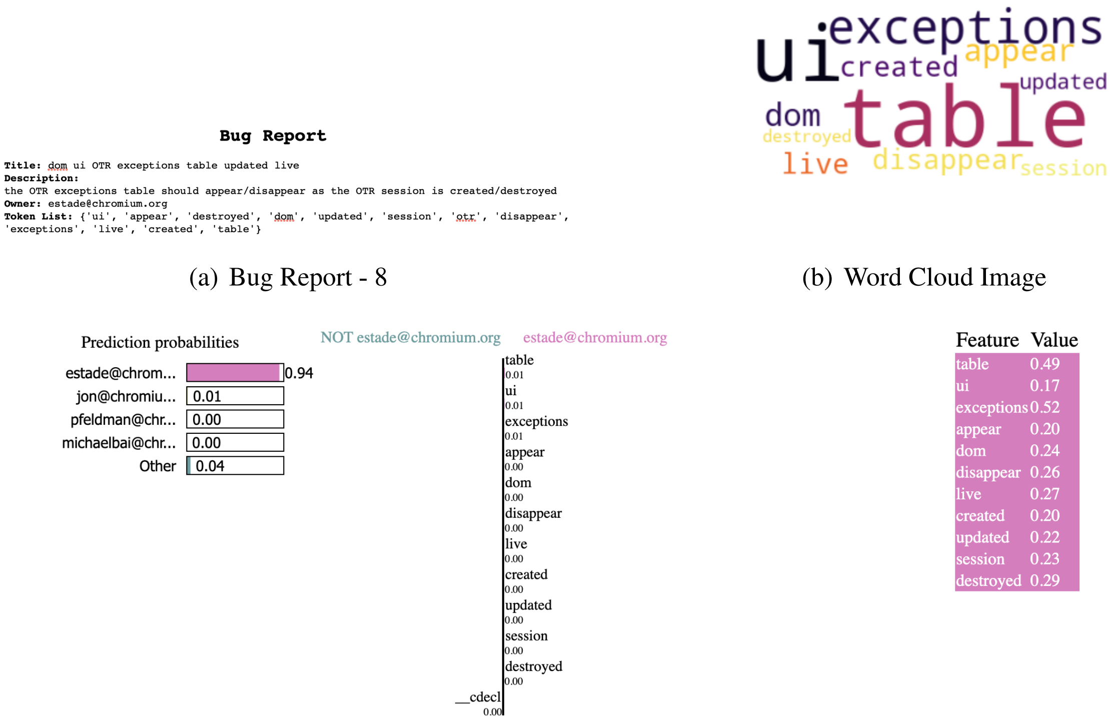
(a) Bug Report - 7

(b) Word Cloud Image

Feature	Value
extension	0.17
hosts	0.53
script	0.18
object	0.18
https	0.23
objects	0.21
manifest	0.23
attribute	0.24
scripts	0.19
csp	0.31
want	0.18
rather	0.20
fully	0.24
limit	0.26
add	0.11
folks	0.30

(c) LIME Explanation

Figure B.7: Visual Explanation for Bug Report 7



Bug Report

Title: dom ui OTR exceptions table updated live
Description:
the OTR exceptions table should appear/disappear as the OTR session is created/destroyed
Owner: estade@chromium.org
Token List: {'ui', 'appear', 'destroyed', 'dom', 'updated', 'session', 'otr', 'disappear', 'exceptions', 'live', 'created', 'table'}

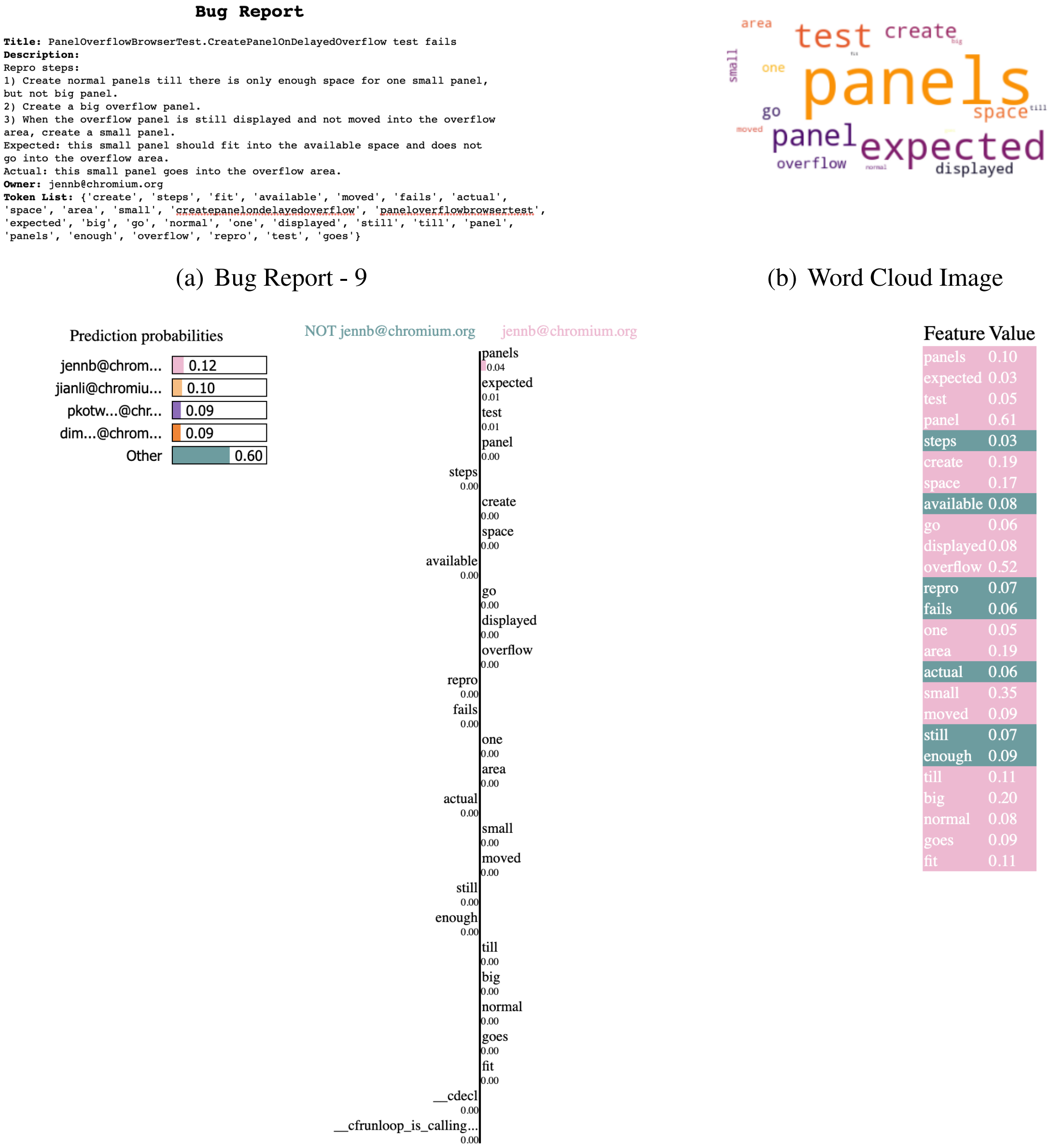
(a) Bug Report - 8

(b) Word Cloud Image

Feature	Value
table	0.49
ui	0.17
exceptions	0.52
appear	0.20
dom	0.24
disappear	0.26
live	0.27
created	0.20
updated	0.22
session	0.23
destroyed	0.29

(c) LIME Explanation

Figure B.8: Visual Explanation for Bug Report 8



Bug Report

```
Title: PanelOverflowBrowserTest.CreatePanelOnDelayedOverflow test fails
Description:
Repro steps:
1) Create normal panels till there is only enough space for one small panel,
but not big panel.
2) Create a big overflow panel.
3) When the overflow panel is still displayed and not moved into the overflow
area, create a small panel.
Expected: this small panel should fit into the available space and does not
go into the overflow area.
Actual: this small panel goes into the overflow area.
Owner: jennb@chromium.org
Token List: {'create', 'steps', 'fit', 'available', 'moved', 'fails', 'actual',
'space', 'area', 'small', 'createpanelondelayedoverflow', 'paneloverflowbrowsertest',
'expected', 'big', 'go', 'normal', 'one', 'displayed', 'still', 'till', 'panel',
'panels', 'enough', 'overflow', 'repro', 'test', 'goes'}
```

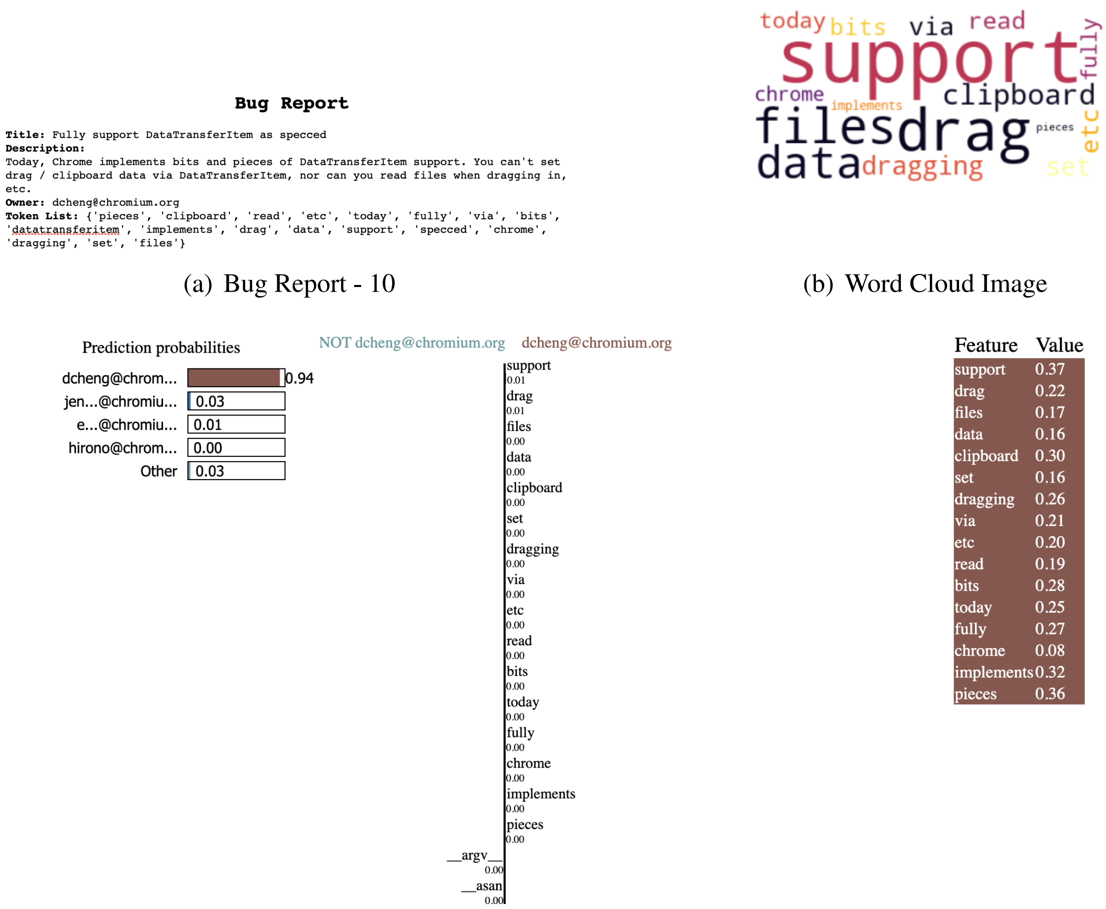
(a) Bug Report - 9

(b) Word Cloud Image

Feature	Value
panels	0.10
expected	0.03
test	0.05
panel	0.61
steps	0.03
create	0.19
space	0.17
available	0.08
go	0.06
displayed	0.08
overflow	0.52
repro	0.07
fails	0.06
one	0.05
area	0.19
actual	0.06
small	0.35
moved	0.09
still	0.07
enough	0.09
till	0.11
big	0.20
normal	0.08
goes	0.09
fit	0.11

(c) LIME Explanation

Figure B.9: Visual Explanation for Bug Report 9



Bug Report

Title: Fully support DataTransferItem as specced
Description:
Today, Chrome implements bits and pieces of DataTransferItem support. You can't set drag / clipboard data via DataTransferItem, nor can you read files when dragging in, etc.
Owner: dcheng@chromium.org
Token List: {'pieces', 'clipboard', 'read', 'etc', 'today', 'fully', 'via', 'bits', 'datatransferitem', 'implements', 'drag', 'data', 'support', 'specced', 'chrome', 'dragging', 'set', 'files'}

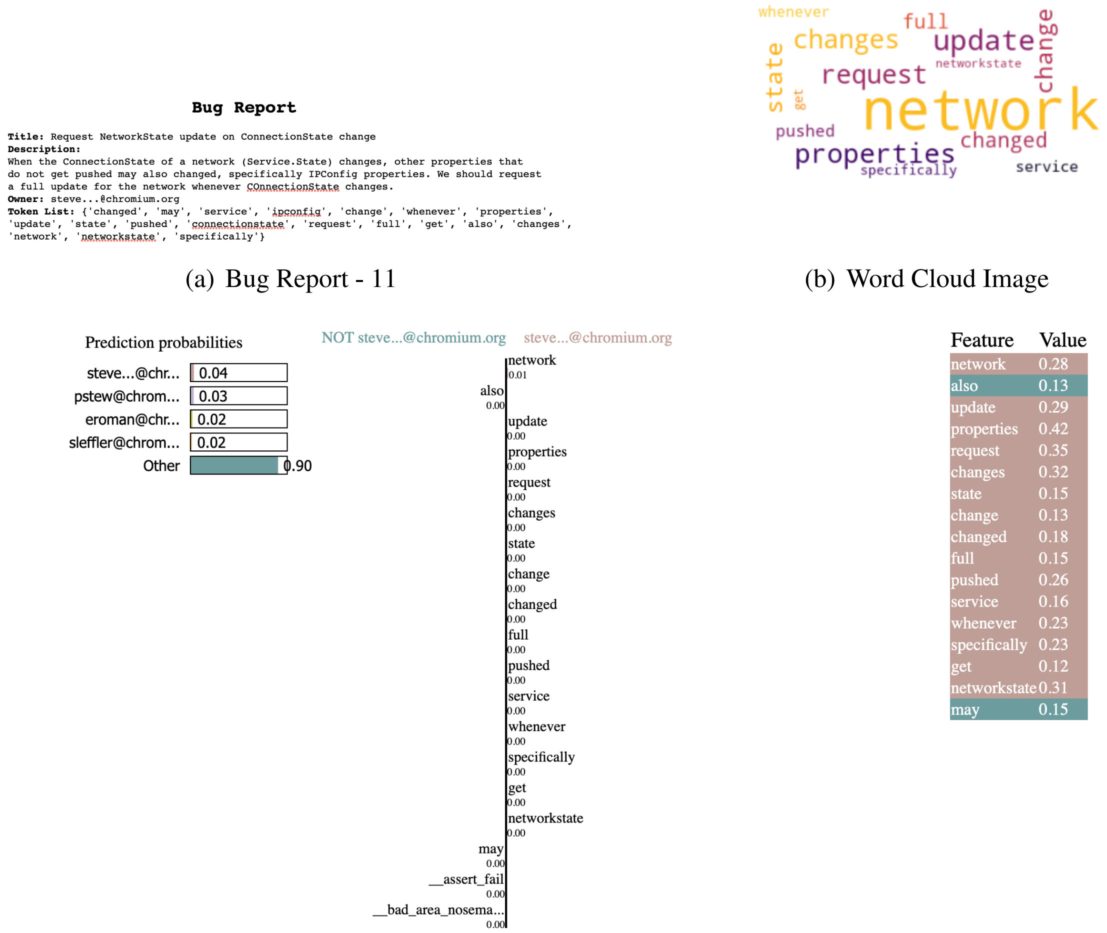
(a) Bug Report - 10

(b) Word Cloud Image

Feature	Value
support	0.37
drag	0.22
files	0.17
data	0.16
clipboard	0.30
set	0.16
dragging	0.26
via	0.21
etc	0.20
read	0.19
bits	0.28
today	0.25
fully	0.27
chrome	0.08
implements	0.32
pieces	0.36

(c) LIME Explanation

Figure B.10: Visual Explanation for Bug Report 10



Bug Report

Title: Request NetworkState update on ConnectionState change
Description:
When the ConnectionState of a network (Service.State) changes, other properties that do not get pushed may also changed, specifically IPConfig properties. We should request a full update for the network whenever COnnectionState changes.
Owner: steve...@chromium.org
Token List: {'changed', 'may', 'service', 'ipconfig', 'change', 'whenever', 'properties', 'update', 'state', 'pushed', 'connectionstate', 'request', 'full', 'get', 'also', 'changes', 'network', 'networkstate', 'specifically'}

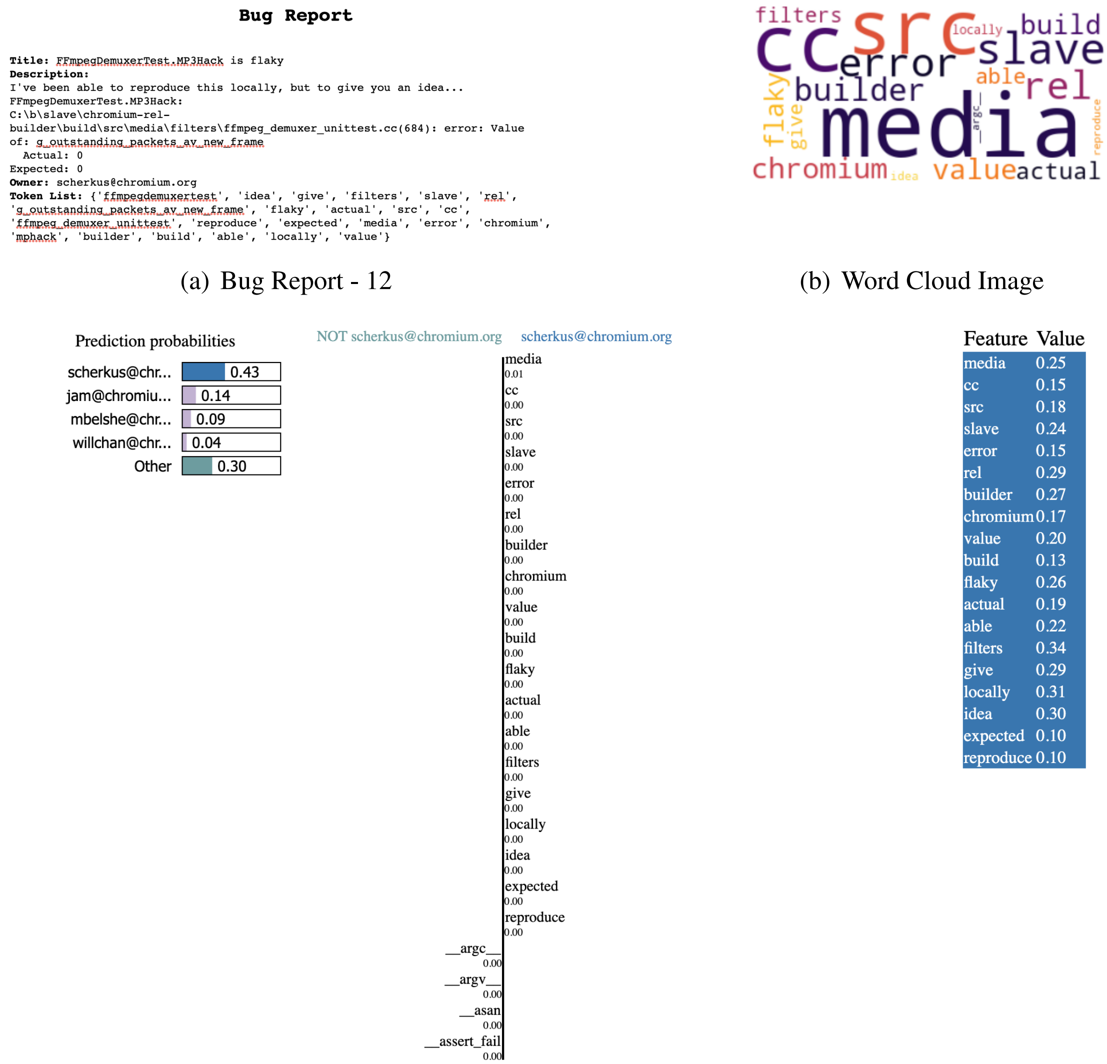
(a) Bug Report - 11

(b) Word Cloud Image

Feature	Value
network	0.28
also	0.13
update	0.29
properties	0.42
request	0.35
changes	0.32
state	0.15
change	0.13
changed	0.18
full	0.15
pushed	0.26
service	0.16
whenever	0.23
specifically	0.23
get	0.12
networkstate	0.31
may	0.15

(c) LIME Explanation

Figure B.11: Visual Explanation for Bug Report 11



Bug Report

```
Title: FFmpegDemuxerTest.MP3Hack is flaky
Description:
I've been able to reproduce this locally, but to give you an idea...
FFmpegDemuxerTest.MP3Hack:
C:\b\slave\chromium-rel-
builder\build\src\media\filters\ffmpeg_demuxer_unittest.cc(684): error: Value
of: g_outstanding_packets_av_new_frame
  Actual: 0
Expected: 0
Owner: scherkus@chromium.org
Token List: {'ffmpegdemuxertest', 'idea', 'give', 'filters', 'slave', 'rel',
'g_outstanding_packets_av_new_frame', 'flaky', 'actual', 'src', 'cc',
'ffmpeg_demuxer_unittest', 'reproduce', 'expected', 'media', 'error', 'chromium',
'mphack', 'builder', 'build', 'able', 'locally', 'value'}
```

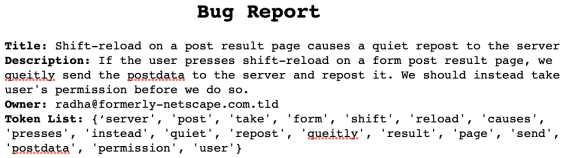
(a) Bug Report - 12

(b) Word Cloud Image

Feature	Value
media	0.25
cc	0.15
src	0.18
slave	0.24
error	0.15
rel	0.29
builder	0.27
chromium	0.17
value	0.20
build	0.13
flaky	0.26
actual	0.19
able	0.22
filters	0.34
give	0.29
locally	0.31
idea	0.30
expected	0.10
reproduce	0.10

(c) LIME Explanation

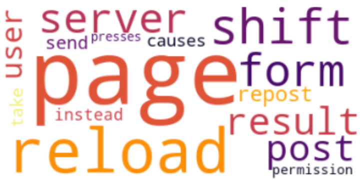
Figure B.12: Visual Explanation for Bug Report 12



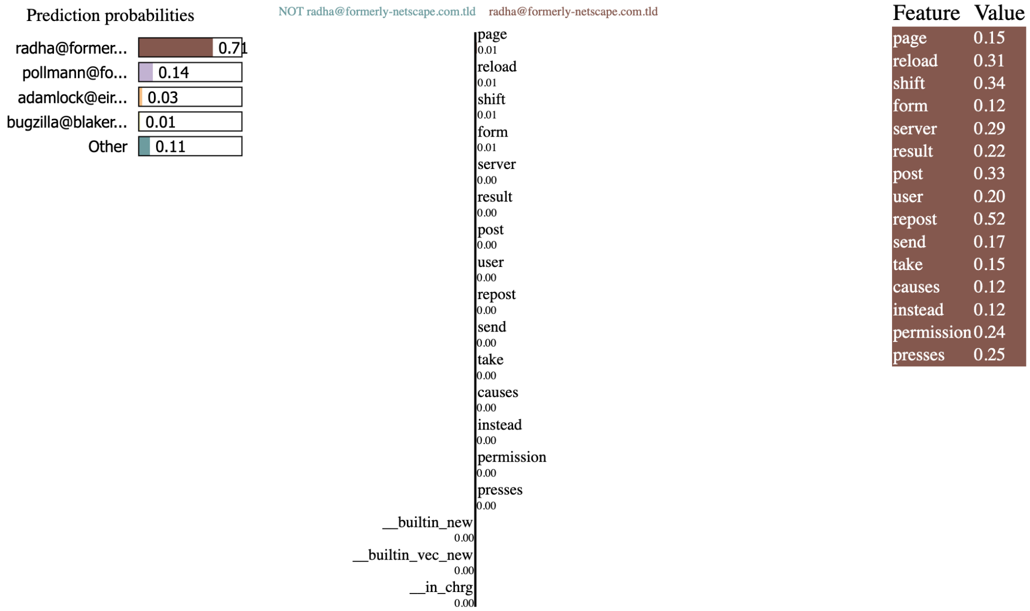
Bug Report

Title: Shift-reload on a post result page causes a quiet repost to the server
Description: If the user presses shift-reload on a form post result page, we queitly send the postdata to the server and repost it. We should instead take user's permission before we do so.
Owner: radha@formerly-netscape.com.tld
Token List: {‘server', 'post', 'take', 'form', 'shift', 'reload', 'causes', 'presses', 'instead', 'quiet', 'repost', 'queitly', 'result', 'page', 'send', 'postdata', 'permission', 'user'}

(a) Bug Report - 13



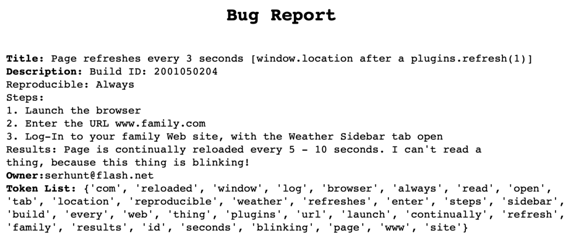
(b) Word Cloud Image



Feature	Value
page	0.15
reload	0.31
shift	0.34
form	0.12
server	0.29
result	0.22
post	0.33
user	0.20
repost	0.52
send	0.17
take	0.15
causes	0.12
instead	0.12
permission	0.24
presses	0.25

(c) LIME Explanation

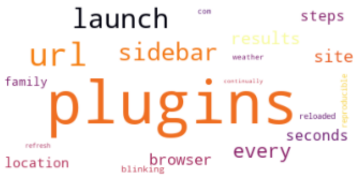
Figure B.13: Visual Explanation for Bug Report 13



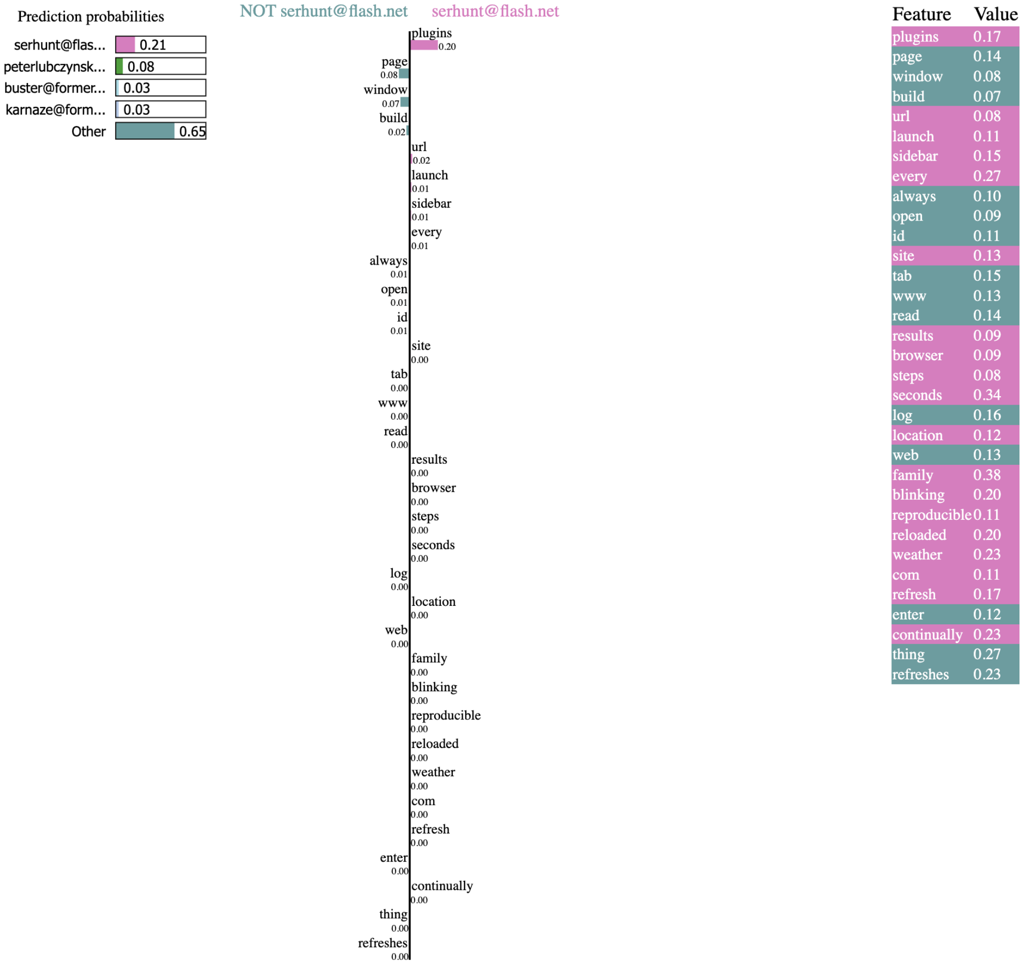
Bug Report

Title: Page refreshes every 3 seconds [window.location after a plugins.refresh(1)]
Description: Build ID: 2001050204
Reproducible: Always
Steps:
1. Launch the browser
2. Enter the URL www.family.com
3. Log-In to your family Web site, with the Weather Sidebar tab open
Results: Page is continually reloaded every 5 - 10 seconds. I can't read a thing, because this thing is blinking!
Owner:serhunt@flash.net
Token List: {'com', 'reloaded', 'window', 'log', 'browser', 'always', 'read', 'open', 'tab', 'location', 'reproducible', 'weather', 'refreshes', 'enter', 'steps', 'sidebar', 'build', 'every', 'web', 'thing', 'plugins', 'url', 'launch', 'continually', 'refresh', 'family', 'results', 'id', 'seconds', 'blinking', 'page', 'www', 'site'}

(a) Bug Report - 14



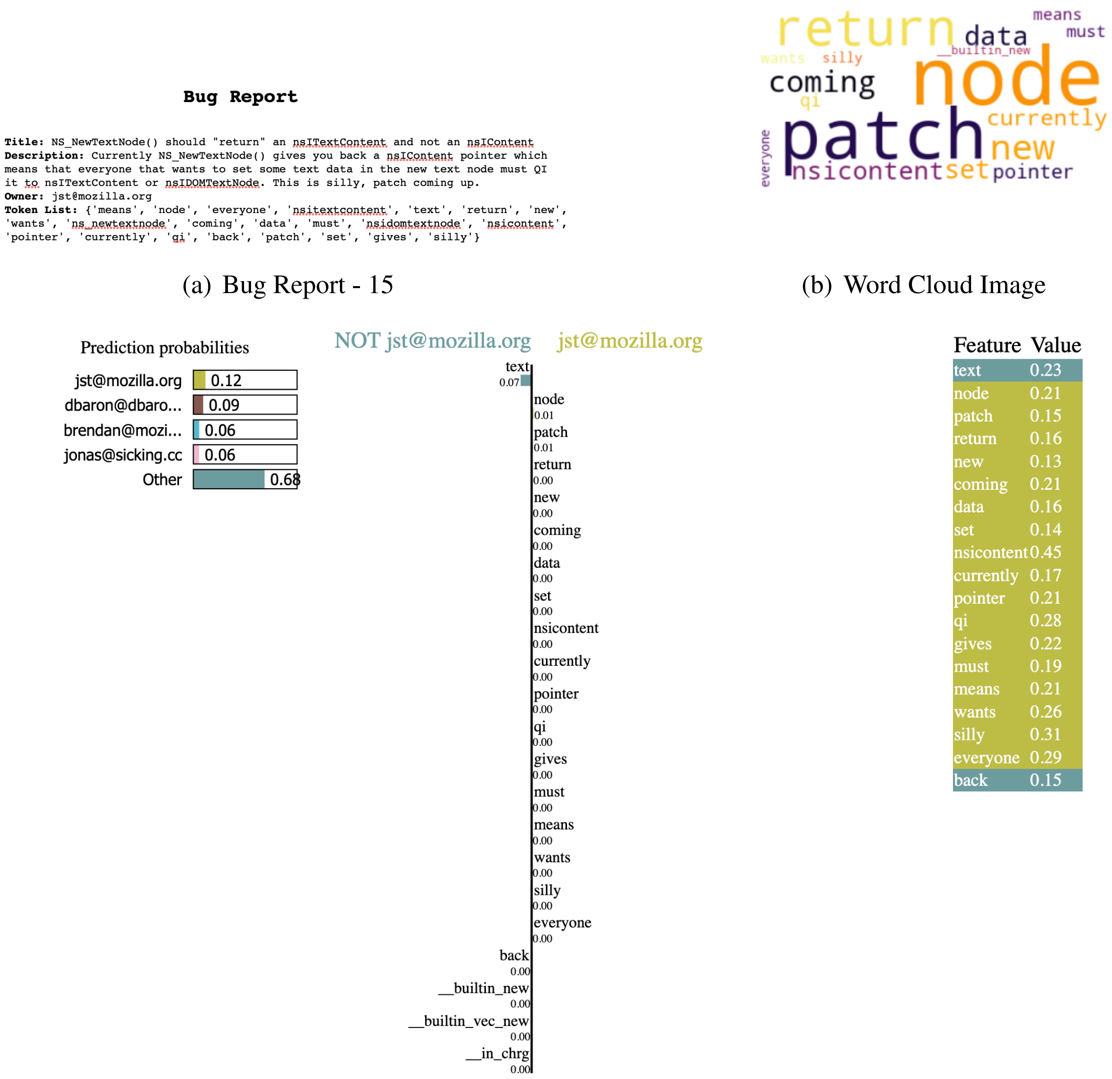
(b) Word Cloud Image



Feature	Value
plugins	0.17
page	0.14
window	0.08
build	0.07
url	0.08
launch	0.11
sidebar	0.15
every	0.27
always	0.10
open	0.09
id	0.11
site	0.13
tab	0.15
www	0.13
read	0.14
results	0.09
browser	0.09
steps	0.08
seconds	0.34
log	0.16
location	0.12
web	0.13
family	0.38
blinking	0.20
reproducible	0.11
reloaded	0.20
weather	0.23
com	0.11
refresh	0.17
enter	0.12
continually	0.23
thing	0.27
refreshes	0.23

(c) LIME Explanation

Figure B.14: Visual Explanation for Bug Report 14



Bug Report

Title: NS_NewTextNode() should "return" an nsITextContent and not an nsIContent
Description: Currently NS_NewTextNode() gives you back a nsIContent pointer which means that everyone that wants to set some text data in the new text node must QI it to nsITextContent or nsIDOMTextNode. This is silly, patch coming up.
Owner: jst@mozilla.org
Token List: {'means', 'node', 'everyone', 'nsitextcontent', 'text', 'return', 'new', 'wants', 'ns_newtextnode', 'coming', 'data', 'must', 'nsidomtextnode', 'nsicontent', 'pointer', 'currently', 'qi', 'back', 'patch', 'set', 'gives', 'silly'}

(a) Bug Report - 15

(b) Word Cloud Image

Feature	Value
text	0.23
node	0.21
patch	0.15
return	0.16
new	0.13
coming	0.21
data	0.16
set	0.14
nsicontent	0.45
currently	0.17
pointer	0.21
qi	0.28
gives	0.22
must	0.19
means	0.21
wants	0.26
silly	0.31
everyone	0.29
back	0.15

(c) LIME Explanation

Figure B.15: Visual Explanation for Bug Report 15