CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Improving Defect Localization by Classifying the Affected Asset using Machine Learning

Master's thesis in Software Engineering

Sam Halali

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Master's thesis 2018

Improving Defect Localization by Classifying the Affected Asset using Machine Learning

Sam Halali

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2018

Improving Defect Localization by Classifying the Affected Asset using Machine Learning
SAM HALALI

Supervisor: Miroslaw Staron, Department of Computer Science and Engineering
Examiner: Jan-Philipp Steghöfer, Department of Computer Science and Engineering

Master's Thesis 2018
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2018

Improving Defect Localization by Classifying the Affected Asset using Machine Learning
SAM HALALI
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Today's market demands complex large-scale software to be developed and delivered at an increased pace. The increase in software complexity increases the cost of maintenance which on average accounts for 60 percent of software costs. Corrective maintenance accounts for 21 percent of the maintenance costs which includes receiving a defect report describing a defect, diagnosing and removing the described defect. A vital part of a defect's resolution is the task of defect localization. Defect localization is the task of finding the exact location of the defect in the system. The defect report, in particular the asset attribute, help the assigned entity to limit the search space when investigating the exact location of the defect. However, research has shown that oftentimes reporters initially assign values to these attributes that provide incorrect information.

In this thesis, using machine learning to classify the source asset for a given defect report at a telecom company was evaluated. Following design science research, two iterations were conducted. The first iteration evaluated classification models for classifying the source asset after submission of a defect report. By training a SVM with features constructed from both categorical and textual attributes of the defect reports an accuracy of 58.52% was achieved. The second iteration evaluated classification models for providing the reporter with recommendations of likely assets. By using recommendations provided by a SVM trained with features from both categorical and textual attributes of the defect reports the precision could be significantly increased.

Keywords: Machine Learning, Defect Localization, Defect predictions, Supervised Learning, Text Classification, Recommendation Systems

Acknowledgements

I would like to thank Miroslaw Staron, my supervisor, for providing valuable feedback and suggestions throughout the thesis process. I would also like to thank the telecom company at which this thesis was conducted. Finally, I would like to thank a long list of people providing me with motivation, help and joy. Thank you:

Micael Caiman, Wilhelm Meding, Per Sundvall, Mahmoud Halali, Maria Ahmadikhatir, Emma Ahlberg, Fredrik Rahn, Simon Kindström, Marko Solunac, Patrik Olsson, Michaela Fritiofsson and Jenin Grill.

Sam Halali, Gothenburg, June, 2018

Contents

List of Figures

List of Tables

1

Introduction

1.1 Problem Background

Today's market demands complex large-scale software to be developed and delivered at an increased pace. The increase in software complexity increases the cost of maintenance [1] which on average accounts for 60 percent of software costs [2]. Corrective maintenance accounts for 21 percent of the maintenance costs which includes receiving a defect report, diagnosing and removing it. In most cases, when a defect is found it is initially documented, assigned to a team and then submitted into a *Defect Tracking System* (DTS).

A defect report consists of several attributes that are vital during the life cycle of a defect. Among these attributes are the textual description of the defect, the asset attribute, which indicates the system part containing the defect, and the severity attribute which indicates the highest impact the defect could or did cause. These attributes are both utilized when assigning the task of removing the defect to a team and when the assigned team investigates the location of the defect and the corrective measures for removing it.

A vital part of a defect's resolution is the task of defect localization. Defect localization is the task of finding the exact location of the defect in the system. This task relies on the developer's expertise of the system and ability to identify and prioritize, during investigation, assets which may contain the defect [3]. The defect report, in particular the asset attribute, should help the assigned entity to limit the search space when investigating the exact location of the defect. However, research has shown that oftentimes reporters initially assign values to these attributes that provide incorrect information [4][5].

A study conducted at Microsoft by Guo et al. [6] showed that oftentimes the reporter of the defect does not have the required expertise for identifying the asset containing the defect which is one of the main reasons behind the attribute being assigned an incorrect value. The issue of incorrect values is also reflected in large-scale organizations, other than Microsoft, with many mission-critical parts, where a delay can have big impact on cost and customer perception. Furthermore, in organizations that have dedicated development teams, responsible for specific assets, the defect assignment process is intertwined with the task of defect localization [7]. In such organizations an incorrect asset value may also delay the defect assignment.

1.2 Problem Statement

The problem addressed in this thesis is that the asset attribute of a defect report is often assigned an incorrect value. In large software organizations, it is common that the reporter of the defect does not have the required expertise for identifying the asset containing the defect. This results in the asset attribute being assigned an incorrect value, which results in unnecessary delayed defect resolution which has a large impact on maintenance costs, the speed of software development and the quality of the final product.

1.3 Purpose of the Study

The purpose of this thesis to improve defect localization by classifying the correct asset from which a defect originates by using machine learning. The intention is to build a classification model, based on the results of previous research, in an industrial context. In further detail, the aim is to reduce the defect localization time by providing a more accurate recommendation of which asset that contains the defect. This will aid the assigned entity to limit the search space when investigating the exact location of the defect.

1.4 Limitations and Delimitations

Like every study, this study is also subject to limitations. One limitation is that defect reports of a telecom company will be used. The attributes of the defect report that will be considered is a subset of those specified by IEEE 1044 [8] that are recorded upon initial documentation of a defect. These attributes include *Description*, *Artifact*, *Severity* and *Detection Activity* and will be described in further detail in Chapter 2. Therefore the study will only be reproducible given that the previously mentioned attributes exists in the used defect reports.

One delimitation is that only data from defect reports will be used. Other research in the topic has shown that data from other sources, such as version control systems, can be utilized upon prediction [9]. These sources could contain useful features, such as *affected files* and *commit messages*, for classifying the asset but will not be considered during the study due to the time constraint of this thesis.

2
Background

2.1 Defects

According to [8] a defect is: "An imperfection or deficiency in a work product where that work product does not meet its requirements or specifications and needs to be either repaired or replaced.". When a defect is detected its attributes are classified and documented in a *defect report*. The attributes of a defect can be seen in Table 2.1, where the names of the attributes might differ depending on the organization. Furthermore, the values of some attributes are added and changed over time as the organization addresses the defect. The recorded attributes of a defect

Attribute	**Definition**
Defect ID	Unique identifier for the defect.
Description	Description of what is missing, wrong, or unnecessary.
Status	Current state within defect report life cycle.
Asset	The software asset (product, component, module, etc.) containing the defect.
Artifact	The specific software work product containing the defect.
Version detected	Identification of the software version in which the defect was detected.
Version corrected	Identification of the software version in which the defect was corrected.
Priority	Ranking for processing assigned by the organization responsible for the evaluation, resolution, and closure of the defect relative to other reported defects.
Severity	The highest failure impact that the defect could (or did) cause, as determined by (from the perspective of) the organization responsible for software engineering.
Probability	Probability of recurring failure caused by this defect.
Effect	The class of requirement that is impacted by a failure caused by a defect.
Type	A categorization based on the class of code within which the defect is found or the work product within which the defect is found.
Mode	A categorization based on whether the defect is due to incorrect implementation or representation, the addition of something that is not needed, or an omission.
Insertion activity	The activity during which the defect was injected/inserted (i.e., during which the artifact containing the defect originated).
Detection activity	The activity during which the defect was detected (i.e., inspection or testing).
Failure reference(s)	Identifier of the failure(s) caused by the defect.
Change reference	Identifier of the corrective change request initiated to correct the defect.
Disposition	Final disposition of defect report upon closure.

Table 2.1: The attributes of a defect © 1993 IEEE

serve both the reporter and the receiver, who is responsible for removing the defect. Therefore defect reports are usually stored in a database known as *Defect Tracking System (DTS)*. The DTS is used by employees of the organization, both engineers and managers, to understand the defect and follow it's resolution process. When a defect has been documented and classified, it is assigned to an entity, for instance

a development team, that is capable of removing the defect. Once the corrective measures has been taken to resolve the defect, the changes are added to a planned release from which the defect is removed. The defect life cycle is depicted in Figure 2.1 and the relationship between defects and several conceptual entities is shown in Figure 2.2.

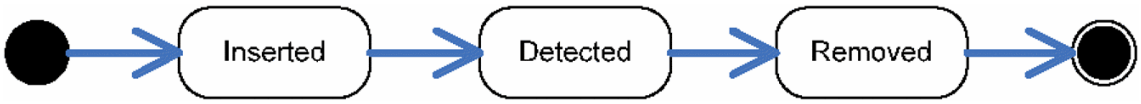


Figure 2.1: The life cycle of a defect © 1993 IEEE

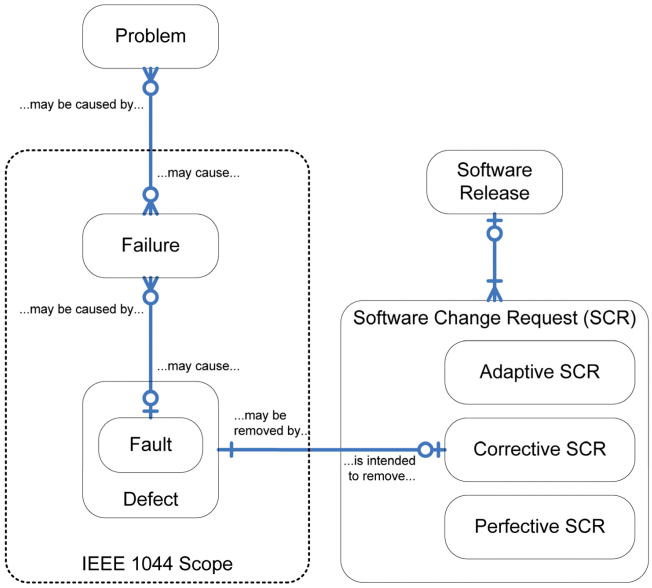


Figure 2.2: The relationship between defects and other entities in maintenance © 1993 IEEE

At the telecom company where this study was conducted, defects are detected during several stages of the life cycle of their products. Defects can for instance be detected by end-users or during the internal testing of the product. When defects are detected by an end-user, the unit for customer support is contacted. Customer support investigates the reported problem and determines if the problem occurred due to a defect in the product. If customer support determines that the problem occurred due to a defect a defect report is created and submitted in the company's DTS.

Regardless of who the reporter is a web-portal is used to record the attributes of and describe the defect. The web-portal looks similar to the one shown in Figure 2.3. The attributes shown in Figure 2.3 are the attributes that are mandatory to record before submitting the defect report. The attributes are: *Artifact*, *Detection Activity*, *Severity*, *Asset* and *Description*.

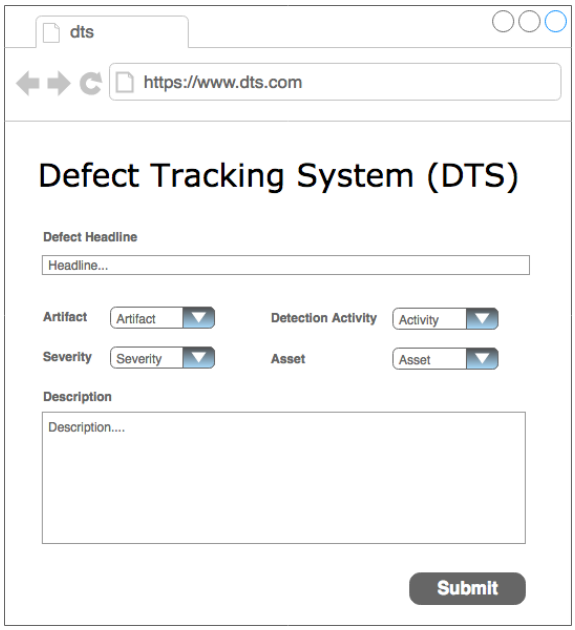


Figure 2.3: Example view of when reporting a defect.

The life cycle of a submitted defect report at the telecom company can be seen in Figure 2.4. When the defect report has been submitted it is assigned to team capable of removing it. The assignment process varies depending of what type of defect it is. Sometimes there are units responsible for certain assets of the product. These units receive all the submitted defect reports regarding the asset that they are responsible for and decide which team will be responsible for investigating and removing the defect.

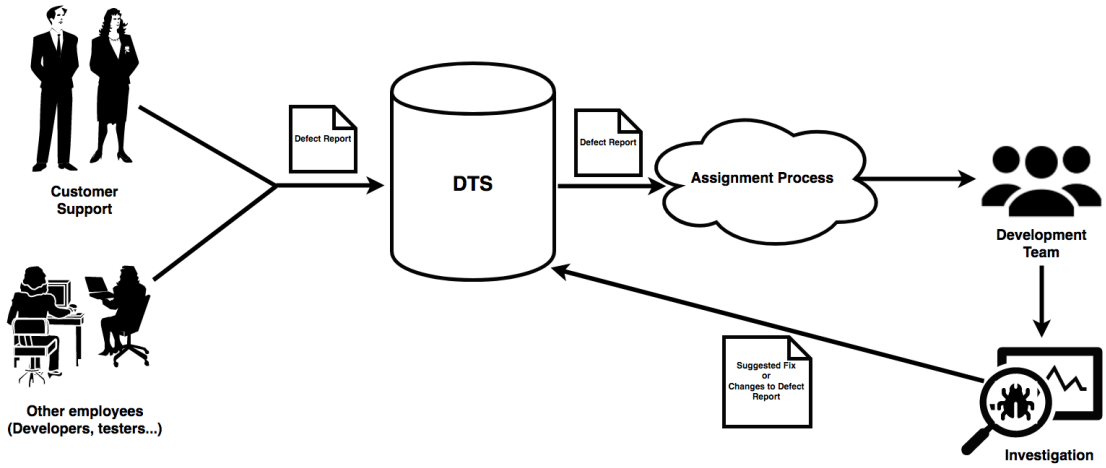


Figure 2.4: The life cycle of a defect report at the telecom company

Once the defect report has been assigned, it is investigated. The ideal outcome of the investigation is that a corrective measure is suggested which leads to the re-

moval of the defect. However, this is not always the case. Oftentimes, the attributes of the defect are assigned incorrect values. If the incorrect values are detected by the assigned team, the team will suggest changes in the defect report which can lead to a reassignment of the defect report.

2.2 Machine Learning

The goal of Machine Learning is to develop computational functions that learn through accumulated experiences [10]. The two most common methods of learning are: *Unsupervised Learning* and *Supervised Learning*.

In unsupervised learning the goal is to learn structures of data sets without an identified output[11]. For instance, given a data set containing 100 defect reports originating from 5 different assets, the data needs to be grouped based on the assets without any explicit indicator of which asset the defect originates from. In the absence of *labels*, which are explicit indicators of what group each data point belongs to, the groups are learned by maximizing the similarities of the entries within a group and minimizing the similarities between the groups. One of the reasons behind the growth of the learning method is that, in certain problem domains, large unlabeled data sets are available but labeling each entry would be time consuming and require domain expertise [12]. For instance, when developing a recommendation system for a news-platform, a large number of articles would be available but manually labeling each article as economy, sports or politics would be impractical.

The goal of supervised learning is to train a model to *predict* one or more outputs for unseen inputs, given that there is a functional relationship between the inputs and the outputs [13]. This is achieved by training the model with a *labeled data set* which is a data set that contains correct outputs given certain inputs. According to Louridas et al.[14] this can be compared to giving a student a set of problems with their respective solutions for learning how to solve future unseen problems.

The output variables to be predicted can be either *categorical variables* or *numerical variables* [13]. Categorical variables can be either nominal, ordinal or dichotomous. Numerical variables can be either discrete or continuous and can take on values that are obtained through measuring or counting. Descriptions of the different variable types can be seen in Table 2.2.

Variable Type	**Description**	**Example**
Nominal	A categorical variable that has more than two possible values which does not have a natural order.	$Asset = \{UI, Login, Validator\}$
Ordinal	A categorical variable which values have a natural order.	$Severity = \{A, B, C\}$
Dichotomous	Like a nominal variable with only two possible values.	$State = \{Open, Closed\}$
Discrete	A numerical variable that can take a value from a set of whole numbers.	#Lines Affected={1, 2, 3, ...}
Continuous	A numerical variable that can take any value between a set of real numbers.	Time={2018-04-06 04:01, ... , 2018-04-06 04:01:02,....}

Table 2.2: Description of the different variable types

The task of predicting an output variable varies depending on the type of the variable. If the output is a numerical variable, as when trying to predict the number of defects reported per week, the problem is a *regression* problem. However, if the output is a categorical variable, as when trying to determine the severity of a defect, the problem is a *classification* problem.

As the intention of this thesis is to build a model for determining the asset from which a defect originates, which is a nominal variable, the presented algorithms will be discussed in the context of classification. Furthermore, since all the provided defect data is labeled only supervised learning algorithms will be presented.

2.2.1 Naive Bayes

Naive Bayes is a probabilistic classification algorithm that assigns an input with the label that is most likely for that specific input [15]. The algorithm is based on *Bayes Theorem* which can be seen in Equation 2.1 where $P(A|B)$ can be read as the probability of A given that event B has occurred.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{P(A \cap B)}{P(B)} \tag{2.1}$$

When classifying the unseen entry seen in Table 2.3, Bayes Theorem would be used as seen in Equation 2.2 where y is the classified label. Calculating the conditional probability $P(Login|A \cap Customer)$ would be calculated as seen in Equation 2.3.

Severity	**Detection Activity**	**Asset**
A	Customer	-
A	Customer	Login
A	System Test	Validator

Table 2.3: Example of categorical defect report data with an unseen entry used for a Naive Bayes model.

$$y = argmax\ P(Login|A \cap Customer), P(Validator|A \cap SystemTest) \tag{2.2}$$

$$P(Login|A \cap Customer) = \frac{P(A \cap Customer|Login)P(Login)}{P(A \cap Customer)} \tag{2.3}$$

Since Naive Bayes makes the naive assumption that all inputs are independent, the calculation of the conditional probability is simplified as seen in Equation 2.4. Furthermore the denominator is excluded from the calculation since it is constant when comparing different conditional probabilities for the same inputs.

$$P(Login|A \cap Customer) = \frac{P(A|Login)P(Customer|Login)P(Login)}{P(A)P(Customer)} \tag{2.4}$$

Now the classification task can be simplified as seen in Equation 2.5, where X is the vector of inputs $\{x_1, ..x_n\}$ and C_k is the target variable with the possible values $\{C_1, ...C_p\}$.

$$y = \underset{k \in \{1,..p\}}{argmax}\ P(C_k) \prod_{i=1}^{n} P(x_i|C_k) \tag{2.5}$$

A Naive Bayes model, trained with the labeled data seen Table 2.3, would classify the unseen entry as *Login*. This is because the entry has a greater likelihood of belonging to class *Login*, as seen in Equation 2.6 and 2.7.

$$P(Login)P(A|Login)P(Customer|Login) = 0.5 \cdot 1 \cdot 1 = 0.5 \tag{2.6}$$

$$P(Validator)P(A|Validator)P(Customer|Validator) = 0.5 \cdot 1 \cdot 0 = 0 \tag{2.7}$$

Naive Bayes offers fast training, implementation and classification [16][15]. However, it usually has a lower accuracy in interdependent data sets, compared to other classification algorithms, due to the assumption of independence amongst the features. Furthermore the model offers interpretable insights in both decision-making when classifying and knowledge gained during training.

2.2.2 Logistic Regression

Logistic regression is a classification method borrowed from the field of statistics. The method combines the logistic sigmoid function seen in Formula 2.8 and a linear model for classifying entries using the model seen in Formula 2.9 [17].

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.8}$$

$$P(x) = \frac{1}{1 + e^{-(\beta_0 x + \beta_1)}} \tag{2.9}$$

When predicting a binary target variable, the output of the function can be seen as $P(x) = P(Y = A|X)$ where X is labeled A if $P(X) > 0.5$. During training the slope β_0 and intercept β_1, seen in Formula 2.9, are decided by maximizing the likelihood function seen in Formula 2.10.

$$\ell(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1 - p(x_{i'})) \tag{2.10}$$

When using categorical variables each unique value adds another dimension. Therefore, to provide an intuitive example, a different example than the example used for describing Naive Bayes is provided. Given the data seen in Table 2.4, the logistic regression model would learn the intercept and slope for the sigmoid function seen in Formula 2.11. Furthermore a plot of the function together with the data used for training can be seen in Figure 2.5. When classifying the unseen entry $\mathbf{x}$ seen in Table 2.4, the model would label the entry, using the learned function, as $Overload = \text{NO}$ since $P(2.7) \approx 0.19 < 0.5$.

$$P(x) = \frac{1}{1 + e^{-(2.8329x + 9.1050)}} \tag{2.11}$$

ID	Avg Sent (Mbit/s)	Overload
x	**2.7**	**-**
1	0.5	NO
2	1.5	NO
3	2	NO
4	4.5	YES
5	5	YES
6	3.5	YES
7	3	NO
8	3	YES
9	3.5	NO
10	4	YES

Table 2.4: Example of server metric data which are labeled with whether the server is overloaded or not.

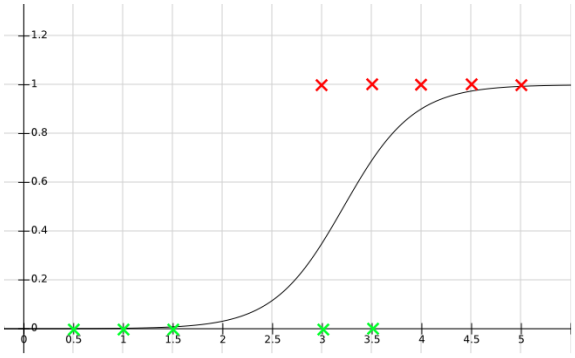


Figure 2.5: A Plot of the function seen in Formula 2.11 and the data points seen in Table 2.4.

Similar to Naive Bayes, Logistic Regression provides interpretable classifications since the output of the sigmoid function can be interpreted as a probability of class membership. Furthermore Logistic Regression offers fast classification and makes no assumptions of independence amongst the features. However, the algorithm is limited to linearly separable data since the decision boundary can be seen as linear.

2.2.3 Decision Trees

Decision trees is a classification algorithm that uses a tree of logical decisions based on the attributes of the data to predict a label [15]. Much like a flowchart, the data to be classified starts at the root node and traverses down the tree based on the attributes of the data. At each branch of the tree there is a *decision node* which indicates a decision to be made based on a attribute. The leaf node, which is also known as the *terminal node*, indicates the classified label of the data.

The data set shown in Table 2.5 consists of 4 entries containing categorical defect report data. Assume this data was used to train a decision tree model with the field *Asset* as the target variable. The resulting decision tree would look like Figure 2.6.
One of the main advantages of using decision trees is the *interpretability* of the classifications [18]. Assume the decision tree is used to classify the entry with $ID =$

ID	Severity	Detection Activity	Asset
1	A	Customer	Login
2	B	System Test	Validator
3	B	Customer	Login
4	A	System Test	Load Balancer

Table 2.5: Example data set used for training a decision tree model

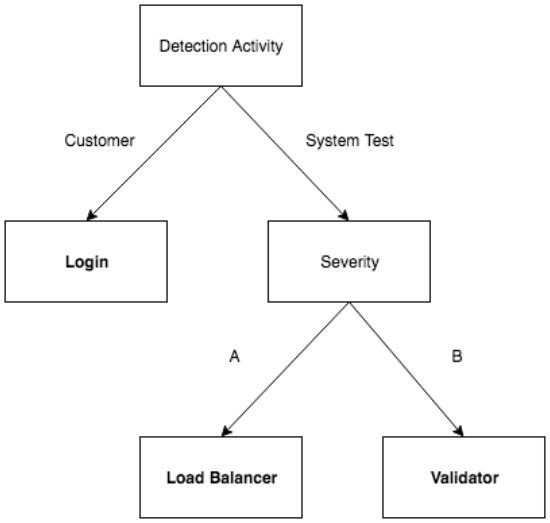


Figure 2.6: Example of a decision tree

3, the model would classify the asset as *Login*. If someone would ask why the asset is classified as such the answer would be *"because the Detection Activity is Customer"*. This type of insight in the classification is not common amongst the different classification algorithms and is viewed as one of decision trees strong suits. Furthermore decision trees have a fast classification phase since an unseen entry only traverses down the tree based on the path decided by the decision nodes, until a terminal node is reached[15]. However, decision trees are both sensitive to noise and redundant features[16] which can lead to overly complex trees.

2.2.4 Support Vector Machines

A Support Vector Machine (SVM) is a classification method that constructs a linear boundary, called a hyperplane, that separates the entries by their labels into two partitions [15] [19]. The class of an unseen entry is decided based on which partition the entry is located in.

Similar to when using Logistic Regression, using categorical inputs adds another dimension for each unique value of the variable. Therefore, the problem of classifying if a server is overloaded is used as an example. The provided training data and the unseen entry $\mathbf{x}$ can be seen in Table 2.6. Since the data is linearly separable, a SVM would identify infinite number of possible hyperplanes during training. Three of the possible hyperplanes, denoted as **A**, **B** and **C**, are shown in Figure 2.7.

The problem of deciding between different possible hyperplanes is solved by search-

ID	#Users (*1000)	Avg Sent (Mbit/s)	Overload
x	**1.5**	**5**	**-**
1	2	1	NO
2	1	2.5	NO
3	3	4	YES
4	4	3.5	YES
5	4	5	YES
6	1	1.5	NO
7	2	3	NO
8	3	3.5	YES
9	1.5	2	NO
10	3.5	4.5	YES

Table 2.6: Example of server metric data which are labeled with whether the server is overloaded or not.

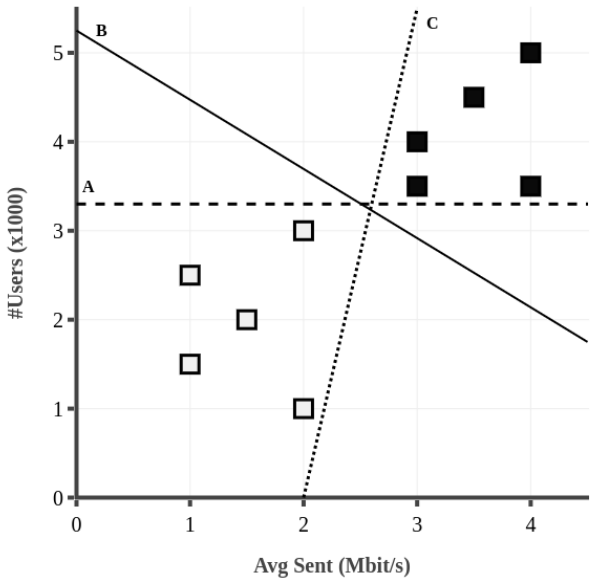


Figure 2.7: Example of possible hyperplanes for data linearly separable data seen in Table 2.6.

ing for the *Maximum Margin Hyperplane (MMH)*. The MMH is the hyperplane that creates the greatest separation between the two classes and is most likely generalize best to future unseen data [15]. Furthermore, the hyperplane is represented by a set of support vectors, which are the points that lie on the hyperplane's maximum margin. Given the hyperplanes provided in Figure 2.7, the SVM would determine that **B** is the MHH and use it as a boundary between the partitions. The coordinates of the unseen entry **x**, seen in Table 2.6, would be $(1.5, 5)$ which is above the partition boundary **B**. Therefore the entry would be labeled as $Overload = \text{YES}$.

SVMs can also be applied to data that is not linearly separable. This can be done by

using either a *slack variable* or the *kernel trick*. The slack variable allows misclassification of entries for a cost C per entry. Now instead of searching for the MMH the goal of the algorithm is to minimize the total cost. The kernel trick is the process of using kernel functions to transform the data to a higher dimension space, to trick the algorithm to view the data as linearly separable. The kernel function adds new features based on mathematical relationships of the existing features.

Overall SVMs have been used in a wide variety of domains where they have provided highly accurate models. Furthermore, the classification phase of SVMs is considered to be fast [16] [15]. This is because the label of an unseen entry is determined by its relation to the MMH. However, finding the MMH comes with great computational cost which slows down the training phase significantly.

2.2.5 Artificial Neural Networks

Artificial Neural Networks (ANNs) originate from attempting to represent the human brain as a mathematical model [20] [21] [15] and are used to solve a variety of problems in both supervised and unsupervised learning. The artificial neuron as seen in Figure 2.8 behaves much like a biological neuron. The neuron receives a set of input signals denoted as x_i which are multiplied by their respective weights w_i and summed to a single value. The total is then used by the activation function denoted as f which results in the signal $y(x)$. The formal definition of a artificial neuron can be seen in Formula 2.12.

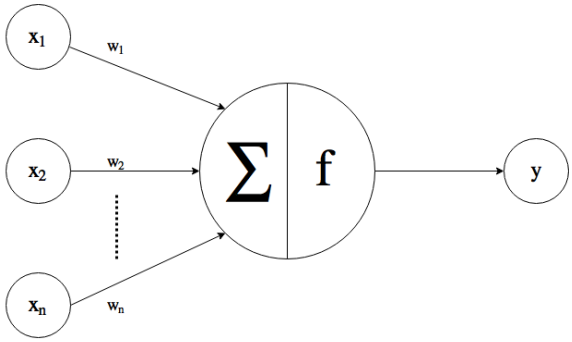


Figure 2.8: An artificial neuron

$$y(x) = f(\sum_{i=1}^{n} w_i x_i) \tag{2.12}$$

The main characteristics of a ANN is the activation function, the network topology and the training algorithm. The activation function calculates the output of the neuron which is forwarded to the other connected neurons. The most simple activation function is the threshold activation function. A typical threshold activation function outputs a signal when the sum of inputs reach a certain threshold and does nothing otherwise. However because of the discrete nature of a threshold activation function the most common activation function is the logistic sigmoid activation function which shares a similar shape but is continuous.

The network topology of an ANN is defined by the number of layers, the allowed directions of dataflow and the number of nodes in each layer. The capabilities of an ANN of finding subtle patterns in complex data sets is generally determined by the size of the network and the composition of nodes. In ANNs data can be allowed to flow backwards which can enable the network to learn patterns in sequences of events over time. The weights of the inputs is determined during the training phase with the use of some training algorithm with most common one being *backpropagation*.

2.2.6 K-Nearest Neighbors

Given a training data set, the Nearest Neighbor algorithm assigns an unseen entry **x**, the same label as the training entry that is nearest given a distance metric d [15][11]. There are several distance metrics used for determining the nearest neighbor, such as the *manhattan distance* and the *euclidean distance* for continuos features and the *hamming distance*[22] for nominal features. To avoid misclassification in noisy domains, more than one of the nearest neighbors are used for deciding the label which is why the algorithm is most commonly known as *k-Nearest Neighbors (kNN)*.

When classifying the label of entry **x** seen in Table 2.7, given the training examples below **x**, the algorithm would:

1. Calculate the distance (hamming distance) between **x** and the training examples.
2. Identify the k entries that closest to **x**.
3. Identify the most common label c amongst the k entries.
4. Label **x** as c.

Assume $k = 3$, given the data in Table 2.7 that has been encoded to the data seen in Table 2.8, the algorithm would identify the entries *1*, *3* and *5* as the k-nearest neighbors. The most common label amongst the nearest neighbors is *Login*, therefore the entry **x** would be labeled as *Login*.

ID	**Severity**	**Detection Activity**	**Asset**
x	**C**	**Customer**	**-**
1	A	Customer	Login
2	B	System Test	Validator
3	B	Customer	Login
4	A	System Test	Load Balancer
5	C	Function Test	Input Parser

Table 2.7: Categorical defect report data with unseen entry **x**

The k-Nearest Neighbors algorithm has a fast training phase but is slow when classifying new entries [15][16]. During the training phase, the algorithm does not learn from the training data, instead it stores it in memory which makes the training phase inherently fast. This is undesirable when dealing with large amounts of data

ID	A	B	C	Customer	System Test	Function Test	Asset	Hamming Distance
x	**0**	**0**	**1**	**1**	**0**	**0**	**-**	
1	1	0	0	1	0	0	Login	2
2	0	1	0	0	1	0	Validator	4
3	0	1	0	1	0	0	Login	2
4	1	0	0	0	1	0	Load Balancer	4
5	0	0	1	0	0	1	Input Parser	2

Table 2.8: The categorical defect report seen in Table 2.7 that has been encoded using the one-hot method.

since the storage requirement scales with the number of training examples. The slow classification of an unseen entry is due to the great computational cost that comes with calculating the distance between the entry and all training examples, and finding the k examples with the shortest distance to the entry.

2.2.7 Performance Evaluation of Classification Models

There are several ways of evaluating the performance of a trained classifier. The most common measures of performance is the *error rate* and *accuracy* [18]. The accuracy of a trained classifier is derived by dividing the correctly labeled entities with the total number of labeled entities. The error rate is derived by dividing the incorrectly labeled entities with the total number of labeled entities. This metric works well for balanced data sets however, it is not an accurate measure of performance for unbalanced data sets.

Assume that we are predicting a output variable X. The variable can take on two values, A and B. The distribution of the data is unbalanced, 98% of the values are labeled A and 2% of the values are labeled B. A classifier that always predicts A would have an accuracy of 98% however most people would agree that this classifier is of no use.

If the class distribution is unbalanced, the accuracy metric needs to be complemented or substituted by other metrics, such as precision and recall. Precision is calculated by dividing the number of correct predictions of a class by the number of times the class was predicted[18]. The precision formula for a class can be seen in Formula 2.13, where *true positives* are denoted as T_P and *false positives* as F_P. Futhermore the precision metric of a class A, can be interpreted as the probability of the classifier being correct, when labeling an unseen entry as A.

$$Precision = Pr = \frac{T_P}{T_P + F_P} \tag{2.13}$$

The precision of a class A, given a data set with 98 entries labeled B, 2 entries labeled A and a classifier that always labels entries as B, except one correct prediction of A, would be 1. Therefore the precision measure is complemented with the *recall* metric, which measures the probability of the classifier recognizing the class A given an instance of the class A. Recall is calculated by dividing the number of correct

predictions of the class by the number of instances of the class[18]. The formula of the recall metric can be seen in Formula 2.14, where *false negatives* are denoted as F_N. The measures recall and precision can be combined into a single metric called the F_1-*score*, which is the harmonic mean of both measures. The formula of the F1-score can be seen in Formula 2.15.

$$Recall = Re = \frac{T_P}{T_P + F_N} \tag{2.14}$$

$$F_1 = 2 \cdot \frac{Pr \cdot Re}{Pr + Re} \tag{2.15}$$

To aggregate a metric that has been measured for all classes, to a single value, two different methods of averaging can be used, namely micro-averaging and macro-averaging. The micro-average of a metric is calculated by dividing the sum of numerators with the sum of the denominators of the measurements for each class. An example of micro-averaging the precision metric for the classes A and B, can be seen in Formula 2.16 given the measures seen in Formula 2.17. This averaging method is biased towards the most populated class.

$$Pr = \frac{T_{P_A} + T_{P_B}}{T_{P_A} + F_{P_A} + T_{P_B} + F_{P_B}} \tag{2.16}$$

$$\begin{aligned} Pr_A &= \frac{T_{P_A}}{T_{P_A} + F_{P_A}} \\ Pr_B &= \frac{T_{P_B}}{T_{P_B} + F_{P_B}} \end{aligned} \tag{2.17}$$

The macro-average is calculated by dividing the sum of the class metrics by the number of classes. Given the measures seen in Formula 2.17, the macro-average of the precision metric for the classes A and B would be expressed as seen in Formula 2.18. This averaging method is useful for unbalanced data sets since it weighs each class equally.

$$Pr = \frac{Pr_A + Pr_B}{2} \tag{2.18}$$

Another metric that has been used to evaluate the performance of classification models, on unbalanced data sets, is the Matthews Correlation Coefficient (MCC) [23][24]. The metric's value ranges between -1 and 1. The value indicates the degree of correlation between the predictions and the results, where 1 indicates complete correlation, 0 indicates no correlation and -1 indicates negative correlation. The formula for the Matthews Correlation Coefficient can be seen in Formula 2.19.

$$MCC = \frac{T_P \cdot T_N + F_P \cdot F_N}{\sqrt{(T_P + F_P)(T_P + F_N)(T_N + F_P)(T_N + F_N)}} \tag{2.19}$$

Furthermore the Matthews Correlation Coefficient has been generalized to cover cases where the target variable is non-binary. Given a $K \times K$ Confusion Matrix, as

seen in Formula 2.20, where the rows indicate the predicted class and the columns indicate the actual class, the metric can be formulated as seen in Formula 2.21.

$$K \times K \text{ Confusion Matrix} = \begin{bmatrix} C_{11} & C_{12} & \dots \\ \vdots & \ddots & \\ C_{k1} & & C_{kk} \end{bmatrix} \tag{2.20}$$

$$MCC = \frac{\sum_k \sum_l \sum_m C_{kk} C_{lm} - C_{kl} C_{mk}}{\sqrt{\sum_k (\sum_l C_{kl})(\sum_{k'|k' \neq k} \sum_{l'} C_{k'l'})} \sqrt{\sum_l C_{lk})(\sum_{k'|k' \neq k} \sum_{l'} C_{l'k'})}} \tag{2.21}$$

2.2.8 Pitfalls in Supervised Learning

When developing a classification model there are a number of pitfalls that needs to be taken into consideration. This section will describe the different pitfalls and how to avoid them.

2.2.8.1 Performance Validation of Classification Models

As the goal of a classification model is to adequately classify a target variable for unseen entries, the classifier should be properly evaluated. Other than deciding appropriate performance measures for the problem domain it is important to consider how the measures translate beyond the training data. Evaluating the model on the training data is misleading since performing well on the training data is easy [25]. For instance, a classifier that memorizes the training data will have a accuracy of 100% when evaluated on the training data. Therefore, it is important to evaluate the model on data that has not been used for training the model. This is done by splitting the data set into a test set and a training set so that the model is evaluated on unseen data.

To avoid the influence of the selection of the test set *k-Fold Cross-Validation* is used. In k-Fold Cross-Validation the data is split into k subsets of same size which is used by the learning algorithms k times. Lets denote the set of subsets $\{x_1, x_2, ..., x_k\}$ as X. For each run, from $i = 1$ to $i = k$, the algorithm uses all subsets $\{x_j \in X | j \neq i\}$ for training and x_i for testing. The final score, used for evaluating the model, is the average of all test scores. Furthermore, if the data set share the same class distribution as the data encountered in the problem domain, randomly splitting the data can yield a pessimistic performance estimate [26]. In such cases, each fold can be stratified to maintain the same class distribution as the entire data set. For instance, when splitting a data set that contains 90 instances of class A and 10 instances of class B into 10 stratified folds, each fold would contain 9 instances of class A and 1 instance of class B. Using stratified folds with K-fold Cross-Validation is known as *Stratified K-Fold Cross-Validation.*

2.2.8.2 Missing Attribute Values

In many data sets the values of some attributes are missing, which can cause problems for certain classification algorithms. For instance, given an entry with one or

more missing values, an artificial neuron will fail to calculate the input of the activation function $\sum_i w_i x_i$. The entries with missing attribute values can be removed, but in some data sets this will result in removing a majority of the data set [18]. The other option is to fill in the missing values which can be done using different strategies[11].

One strategy is to decide the value based on the observed values of the attribute. This can be done by choosing the most frequent value, the average of all values or a random value chosen from the attribute's distribution. However, this can be misleading especially in cases where the attributes are dependent. For instance, when given a defect report with the values $Status = Open$ where $Status \in \{Open, Closed\}$ and $Action = ?$ where $Action \in \{Pending, Fixed\}$. Setting the *Action* attribute to *Fixed* would confuse the model as this data point would be considered as noise.

Another strategy is to train a model to determine the values of the examples with missing attribute values. This is done training a model with all examples that have values for the attribute of interest. When the model has been trained, the values of the examples with missing values are determined by the model. This adds another layer of complexity since each attribute with missing values adds a new machine learning problem.

2.2.8.3 Class Imbalance Problem

A data set where the distribution of classes is unbalanced is said to suffer the *Class Imbalance Problem* [11]. For instance, when given a training set for classifying malicious network requests, the class distribution will most likely be unbalanced. As the imbalance of the class distributions grows, the accuracy when classifying the minority class decreases because of the model's bias towards the majority class[18].

Given a classification task where the goal is to maximize the accuracy and where the provided training and test data share the same class distribution, the class imbalance problem is not deemed as meaningful[11]. However, when classifying malicious network requests, the minority class is of interest. Therefore the model needs to be evaluated with other metrics, that highlights the classifying performance of the minority classes, such as MCC or F1-score.

Substituting the accuracy metric with MCC or F1-score only helps to properly evaluate the model and detect the model's biases towards different classes. However, to mitigate the class imbalance problem the training data needs to be re-sampled. There are two common methods for re-sampling the training data, namely *undersampling* and *oversampling*[18].

When undersampling the training data, examples from the majority class are removed from the data set until the class distribution is balanced. The examples that should be removed can be selected at random or by identifying noisy instances of the majority class. However, in some cases the training set is small and therefore undersampling the data set is impractical. In such cases oversampling is used, where

more examples from the minority class are added to the training set to balance the class distribution. The new examples are created by copying existing examples from the minority class. The copies are then either kept as is or minor modifications are made to the continuous attributes of the copy.

2.2.8.4 Overfitting

Overfitting is when the model fits well to the training set but fails to generalize to unseen data[27]. This is due the model trying to describe all the data points rather than the underlying distribution of the data [11] as seen in Figure 2.9. A common approach for avoiding overfitting is *regularization*.

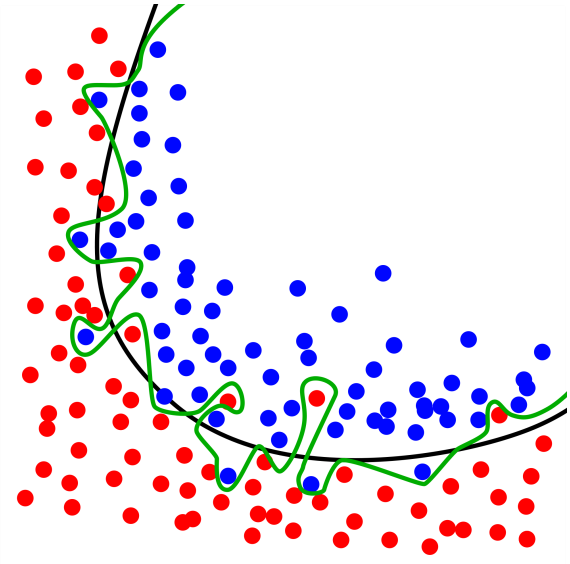


Figure 2.9: Example of a complex overfitted model.

In flexible algorithms such as Logistic Regressors and SVMs, the parameters that maximizes the training score are selected during training. This favours overly complex models as seen in Figure 2.9. To penalize complex models, such as SVMs and Logistic Regressors, *regularization* is used during training. Regularization is the method of using a regularizer, which quantifies the complexity of a model, together with the performance measures when selecting a model.

2.2.8.5 Feature Engineering

Feature engineering is the process of constructing, extracting and selecting features. Features are the characteristics of an object, that are representations of or calculations made with the object's attributes. Consider the data seen in Table 2.9, where the nominal attribute *Detection Activity* is used as feature without any modifications. Training a SVM with the given data would not be possible, since the inputs cannot be placed in a numerical space. Therefore the attributes of an object, has to be represented in a way that can be interpreted by the learning algorithm.

ID	Detection Activity	Asset
1	Customer	Login
2	Function Test	Validator

Table 2.9: Example of nominal defect data

A common way of constructing features from nominal attributes is *One-Hot encoding*. One-hot encoding creates a new dichotomous feature for each possible value of the attribute, where the value 1 indicates the presence of that value. One-hot encoding the *Detection Activity* attribute would yields the features seen in Table 2.10, which now can be used for training a SVM.

ID	Function Test	Customer	Asset
1	0	1	Login
2	1	0	Validator

Table 2.10: Example of features constructed from nominal defect data from Table 2.9 using one-hot encoding

Now consider the data seen in Table 2.11, where the attribute *Description* contains human-written text. Treating the attribute as a nominal attribute and constructing features from it using one-hot encoding would yield the data seen in Table 2.12. When calculating the hamming distance between the unseen entry **x** and other entries, the resulting distance would be the same. This is because the similarities between the texts are not quantified, either the texts are identical or not. Therefore text is usually represented as a *bag of words* [11], which is a feature vector that describes the occurence of each word in the text.

ID	Description	Asset
x	**Login fails**	**-**
1	Login not working	Login
2	Validate function incorrect	Validator

Table 2.11: Example of human-written defect description.

ID	Login not working	Validate function incorrect	Asset
x	**0**	**0**	**-**
1	1	0	Login
2	0	1	Validator

Table 2.12: Example of features constructed from human-written text using one-hot encoding

A simple bag of words representation is constructed by first creating a vocabulary. The vocabulary is constructed by collecting the words of all entries used for

training into a set. The set will now contain all the distinct words found in all documents. After the vocabulary has been constructed, the number of occurrences of each word in the vocabulary is counted for each entry. The resulting vectors are then used as a representation of each document. Applying this technique to the data seen in Table 2.11 would yield the features seen in Table 2.13. Now when calculating the hamming distance, the closest entry to the unseen entry **x** is entry **1**.

ID	**Login**	**not**	**working**	**Validate**	**function**	**incorrect**	**Asset**
x	**1**	**0**	**0**	**0**	**0**	**0**	**-**
1	1	1	1	0	0	0	Login
2	0	0	0	1	1	1	Validator

Table 2.13: Example of features constructed from human-written text using a simple bag of words model

To highlight the words that distinguish each textual entry, also known as documents, a method called Term Frequency - Inverse Document Frequency (TF-IDF) can be used. Similar to the simple bag of words model, features are constructed by first creating a vocabulary. After the vocabulary has been constructed, the frequency of each word in the vocabulary is calculated for each document. The term frequency is calculated by using the function seen in Formula 2.22 for each word and document where w denotes a word and d denotes a document.

$$tf(w,d) = \frac{\#\text{ occurences of } w \text{ in } d}{\#\text{ words in } d} \tag{2.22}$$

To weight each word based on the occurrence across all documents, the inverse document frequency is calculated for each word. The inverse document frequency is calculated by using the logarithmic function seen in Formula 2.23 where w denotes a word. By using a logarithmic function the words that occur frequently across all documents receive a weight close to zero and the words that occur rarely receive a higher weight.

$$idf(w) = \frac{\#\text{ number of documents}}{\#\text{ number of documents containing } w} \tag{2.23}$$

Each position of the final feature vector, which denotes a word in the vocabulary, is derived by multiplying the term frequency with the inverse document frequency for each document. The complete function for TF-IDF can be seen in Formula 2.24.

$$tf - idf(w,d) = tf(w,d) \cdot idf(w) \tag{2.24}$$

3

Related Work

Previously no research has been conducted in predicting the asset attribute of a defect report. However, using machine learning for predicting other defect attributes and improving the process of defect resolution is not uncommon. In the systematic mapping study conducted by Cavalcanti et al. [28], it was reported that most research on the the topic of defect classification is centered on defect assignment and duplicate detection. Furthermore, Cavalcanti et al. noted that previous research could be extended to classify other defect attributes, such as asset and severity, that are recorded manually. This could be helpful for less experienced reporters when reporting a defect.

3.1 Defect Assignment

Automating the task of defect assignment is a well-researched topic. In a systematic review [29], 75 papers researching the topic was reviewed where one of the most common approaches of automating defect assignment being machine learning.

Previous studies have evaluated several classification techniques with the most common evaluation metric being accuracy [29]. The reported accuracies of individual classifiers range from 25 % [30] to 64 % [31] with a higher accuracy being reported by with the use of meta-algorithms [32][7]. The most common features are constructed from the descriptions of defect reports by using TF-IDF [29]. However categorical attribute such as *Asset* [31][30][33][32][34], *Artifact* [30][31], *Submitter*[35][34] and *Detection Activty* [34] are also used.

When evaluating the developed classification model Banitaan et al. [33] reported that the features constructed from the asset attribute were the most influential features for three out of four data sets that were used. Similar findings were reported by Annvik et al. [31], who improved the accuracy of their classification model for Eclipse by 48%, when using features constructed from the asset attribute. However, both papers validated the performance of their classification models using resolved defect reports. Therefore, the effect of the asset attribute being assigned an incorrect value was never observed.

3.2 Duplicate Defect Report Detection

A duplicate defect report describes a defect that already has been reported and submitted into the DTS. Duplicate defects are not wanted in the DTS since investigating it would inherently mean that two different assigned entities are doing the same work. Therefore, organizations often have designated staff investigating the incoming defect reports for duplicates [28].

To automate the process of detecting duplicate defect reports several methods have been proposed utilizing the description of the defect report to measure the similarity between defect reports [36][37][38][39]. By using TF-IDF to construct features from the defect description Jalbert et. al [37] developed a model that managed to filter out 8 % of the duplicate defect reports before reaching the DTS. With a detection rate of 8%, the model could not replace the manual labour required for detecting duplicates. However the model could reduce the number of defect reports to be inspected manually. Since the model's detection rate of non-duplicates was 100 %, the only cost of deploying the model would be the cost of classifying each defect report. The time required for classifying a defect report was reported to be 20 seconds.

Tian et al. [39] developed a classification model with a detection rate of 24 % by also including features constructed from the categorical attributes of a defect report. However, the improvement of the detection rate led to a loss of 9 % of the non-duplicate detection rate.

3.3 Text Classification

Text classification is the task of organizing unlabeled documents into categories. In a paper about text classification with SVMs, Thorsten [40] identified some of the challenges of text classification. Even when removing the stop words of a text corpus the number of remaining words will still be a considerable amount. This high dimensional input space can lead to problems with generalizability for some algorithms. A technique for reducing the dimensions of the input space is to remove features that are irrelevant. However, Thorsten [40] noted that few of the features constructed from the words of the corpus are irrelevant and therefore removing the features would result in information loss.

The algortihms discussed in the Background section namely Naive Bayes, Logistic Regression, Decision Tree, Support Vector Machines, Artificial Neural Networks and K-Nearest Neighbors can all be used for text classification [41]. However, according to Thorsten [40] most text classification problems are linearly separable which benefits algorithms using linear functions to separate the categories. Therefore, linear SVMs, Naive Bayes and Logistic regression are common algorithms used for text classification. However, as stated by the No Free Lunch Theorem [11] there is no one algorithm that works best for every problem.

3.4 Defect Prediction In Industry

Prediction models in the domain of software defects are widely researched where most of the previously presented studies are related to defects in open source software. However, in the domain of large-scale software projects in industry, research has also been conducted. Among the problems studied in this domain are: Defect Inflow Prediction [42][43] and Software Defect Prediction [44].

Defect Inflow Prediction is the task of predicting the number of non-redundant defects being reported into the DTS [43]. In study conducted by Staron et al. [43] a model for predicting the defect inflow during the planning phase for up to 3 weeks in advance was proposed. The model was constructed using multivariate linear regression and modeled the defect inflow as a function of characteristics of work packages. The results showed that the model could support project managers to estimate the work effort needed for completing the project by providing a prediction accuracy of defect inflow of 72%.

To make the testing phase more efficient a *Software Defect Prediction (SDP)* model can be used. SDP is the task of predicting software assets which are prone to defects. By using a SDP model organizations can make testing more efficient by allocating more resources to the predicted assets [44]. Predicting assets prone to defect can be done using machine learning. Rana et al. [44] highlights that the problem can be either a classification or a regression problem.

A classification model for SDP classifies modules, that are represented by software metrics and code attributes, as fault-prone or non-fault-prone based on previous projects [45]. For this task several algorithms including Naive Bayes, Logistic Regression, Decision Trees, Support Vector Machines, Artificial Neural Networks and K-Nearest Neighbors has been used and shown significant results. However, the comparative study conducted by Lessmann et al. [45] concluded that the classification algorithm had little importance when comparing the performance of the 17 most accurate classifiers that were studied.

Even though Machine Learning has been used to developing models for SDP, Rana et al. [44] identified that the adoption of Machine Learning for SDP in industry has been limited. The study showed that attributes other attributes than predictive accuracy such as cost, reliability and generalizability affect the willingness to adopt the technology. To accelerate the adoption of Machine Learning in industry, Rana et al. [44] developed a framework for comparing Machine Learning based techniques to existing systems. Using the framework would help industry to make informed decisions and reflect on their strengths and areas of improvement with respect to a given technology.

4

Research Design

4.1 Research Questions

The aim of this research is to develop a model for classifying the asset from which a defect originate using historic defect reports. Furthermore, the model is evaluated using metrics that are appropriate for the given data set and by measuring the potential time-savings at a large telecom company. This leads to the main research question:

RQ 1: How can the source asset of a given defect report be identified using machine learning?

In order to develop a classification model for classifying the asset from which a defect originates, qualitative features need to be constructed. This leads to the first sub question:

RQ 1.1: Which attributes of the defect reports can be used for constructing features?

When the available attributes have been determined, the initial set of features are constructed. However, some features might not contain information that describes the functional relationship between the defect report and the asset which might decrease the performance of the classification model. Therefore, a suitable feature set needs to be determined. This leads to the second sub question:

RQ 1.2: Which set of features provides the best results using k-fold cross-validation?

This question is addressed by evaluating a number of supervised classification algorithms with a combination of different subsets of features. The outcome will show which combination provides the most accurate classifications.

The answers of the sub questions will be used to answer the main research question. In further detail, the answers will provide a classification model for classifying the asset from which defect originates, which will be evaluated by measuring accuracy, recall, precision, F1-score, the Matthews Correlation Coefficient.

4.2 Research Methodology

The research conducted followed the design science research methodology seen in Figure 4.1. The research started by analyzing the defect reports of the organization which confirmed that, similar to Microsoft [6], the asset attribute oftentimes were assigned incorrect values. This identified the need of aiding the defect reporter by recommending a value for the asset attribute.

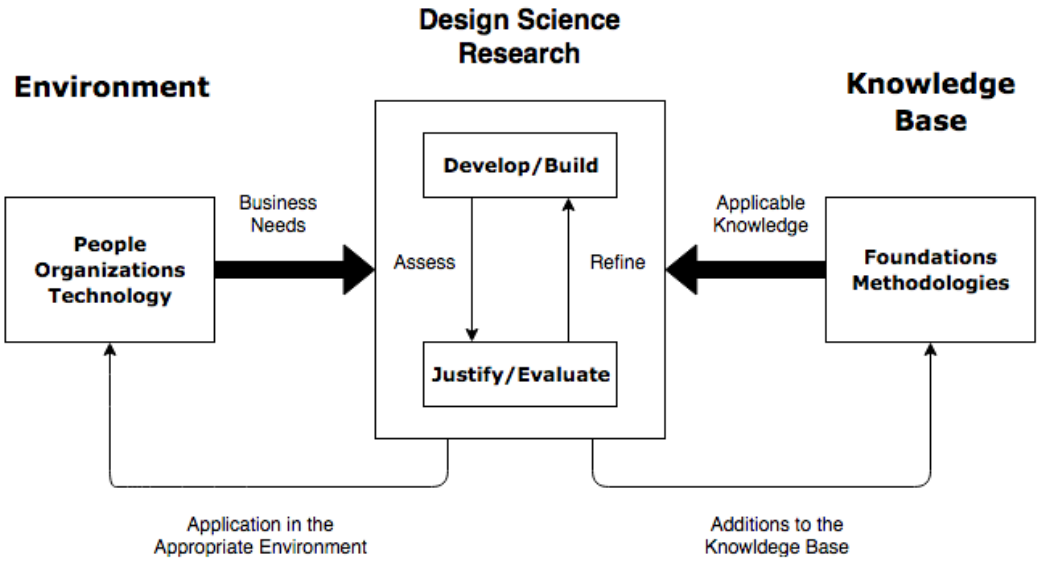


Figure 4.1: Design science research methodology as explained by Hevnet et al. [46]

To gain applicable knowledge for developing a classification model, capable of classifying the asset from which a defect originates from, a literature review was conducted. The outcome of the literature review was information about classification algorithms to consider, the process of developing and evaluating a model, previously applied techniques for predicting defect attributes and the life cycle of a defect in the organization which is presented in Section 2 and 3.

Given the organizations data set, the possibilities of using machine learning for predicting the asset attribute, to justify the development of a classification model, was evaluated. To get familiar with the data set and it's attributes data exploration and mining techniques were used. The data exploration showed that the historic defect reports were labeled with the asset from which the defect was removed and could therefore be used to train a classification model.

After the previous steps, the development and evaluation of different algorithms, together with different subsets of features started.

4.2.1 Data set

The data provided by the organization contained defect reports, submitted by both customers and employees, starting from 2010, grouped by two different products. The two products, denoted as Product 1 and Product 2 in this thesis, are mature products built by several million lines of code with a few hundred active developers each. Furthermore, the products are deployed internationally and each have more than 10,000 submitted historic defect reports.

The data set containing the historic defect reports was imported into a table to act as a snapshot. Only attributes that were mandatory to record when creating a defect report were included in the data set to avoid missing data. The mandatory attributes were: *Description*, *Severity*, *Detection Activity*, *Artifact* and *Asset*. These attributes were a subset of the attributes presented in Table 2.1. Furthermore, the description of the mandatory attributes with their respective variable types can be seen in Table 4.1.

Attribute	**Type**	**Description**
Description	Textual	Description of what is missing, wrong, or unnecessary.
Severity	Ordinal	The highest failure impact that the defect could (or did) cause, as determined by (from the perspective of) the organization responsible for software engineering.
Detection Activity	Nominal	The activity during which the defect was detected (i.e., inspection or testing).
Artifact	Nominal	The specific software work product containing the defect.
Asset	Nominal	The software asset (product, component, module, etc.) containing the defect.

Table 4.1: The mandatory attributes of the defect reports that were provided by the telecom company.

Defect reports which were not addressed or resolved were removed since they could not have been used during training or validation. This was because the asset attribute of the defect reports could be reassigned during the resolution process. Therefore, the assigned value could be incorrect since the value was not final. Furthermore the defect reports that described failures which were not caused by a defect were removed. These failures could, for instance, be caused by not following the documentation when configuring the system. Since the reported failure was not caused by a defect, it would not result in a corrective measure in an asset. Therefore, the assigned value of the asset attribute could be considered incorrect. A lower bound for the number of possible assets for each product can be seen in Table 4.2.

Product	**#Assets**
Product 1	>40
Product 2	>100

Table 4.2: The number of possible assets for each product.

4.2.2 Development Setup

The development and evaluation was performed on a laptop running Windows 7. The limited performance affected the time required to run the experiments. If, for instance, the experiments were conducted on a mainframe computer, each iteration of the 10-Fold Cross-Validation could have been run in parallel. This would have been useful when evaluating algorithms with a slow training phase, such as SVMs, as the computer is unusable during training. Furthermore, this limited the possibilities of tuning each algorithm since each tuning task requires exhaustive search over the parameter values. The optimal value is decided by comparing the score of each parameter value using cross-validation on the training set.

The framework that was used for feature engineering and machine learning tasks is *Scikit-learn* [47]. Scikit-learn offers a vast library of machine learning algorithms and tools for both feature engineering and model evaluation. The selection of the framework was made based on the ease of use, community support and familiarity with Python.

4.2.3 Algorithms

The classification algorithms evaluated for constructing a classification model can be seen in Table 4.3. The set of algorithms was decided based on the usage in related research. As stated by the No Free Lunch Theorem [11] there is no single algorithm that works best for every problem and therefore the most suitable algorithm were decided through the conducted tests. Furthermore, a neural network implementation was excluded due to the long training phase and the large number of parameters needed to be tuned.

Classification Algorithm	**Description**
MultinomialNB [48]	Naive Bayes implementation for multinomially distributed data.
DecisionTreeClassifier [49]	A decision tree implementation for classification that uses a optimized version of CART [50].
LogisticRegression [51]	A Logistic Regression implementation for classification.
KNeighborsClassifier [52]	A K-Nearest Neighbors implementation that uses five neighbors as default.
LinearSVC [53]	A SVM for classification that uses a linear kernel with the cost parameter set to 1 as default.

Table 4.3: Descriptions of the evaluated classification algorithms.

4.2.4 Feature engineering

From the mandatory defect attributes seen in Table 4.1 a set of features were constructed. To represent the textual attribute *Description* as features, the attribute was transformed into a feature vector by using *TF-IDF*. The TF-IDF implementation of Scikit-learn [54] also offered removal of stop words and tokenizations of words using regular expressions. This was utilized to remove all the stop words defined by Scikit-learn [55] and tokenize the words so that they only contained at least one alphabetical character using the following pattern: $(?ui)\\b\\w*[a-z]+\\w*\b$. The stop words and numerical sequences were removed since they were irrelevant for describing the attribute. The 1000 words which had the highest TF-IDF score were selected since a higher number of words would have increased the training time of the model. However, the number of words that are selected from a TF-IDF vector is a parameter that needs to be tuned which is not something that is considered in this thesis. The categorical attributes of the defect reports were transformed into features by using Scikit-learn's algorithm for One-Hot encoding [56].

4.2.5 Iteration 1

The first iteration evaluated the classification scheme seen in Figure 4.2. Using this scheme, the reporter would not assign the asset attribute a value. Instead, the trained classification model would decide the asset, based on the attributes recorded by the reporter. To decide which attributes the model should use when classifying the asset three tests were conducted. The three tests were:

1. Learning with features constructed from the categorical attributes
2. Learning with features constructed from the textual attributes
3. Learning with a combination of the features used for the previous tests

Furthermore, to evaluate which algorithm provided the best result, each test was conducted with the same set of classification algorithms, described in Table 4.3.

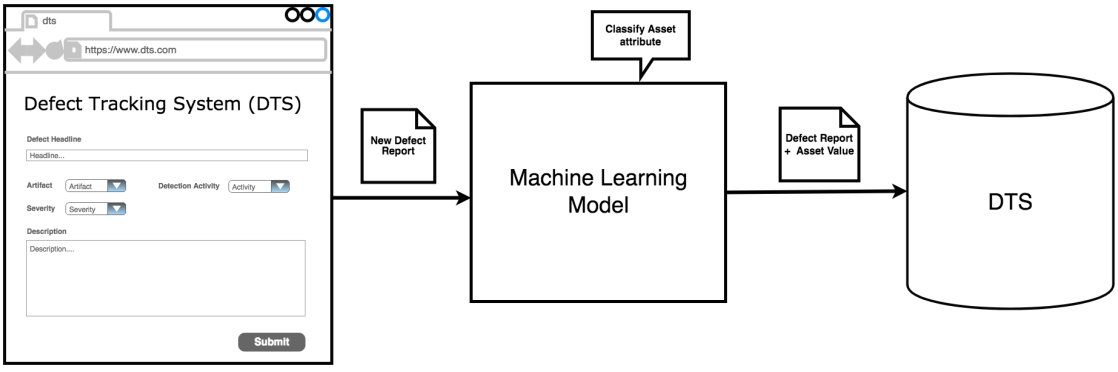


Figure 4.2: Classification scheme for Iteration 1

The metrics used for evaluation during the tests were *Accuracy*, *Precision*, *Recall*, *F1-Score* and *Matthews Correlation Coefficient*. These metrics were used to measure how the classifier performs across all classes, which the accuracy measure fails to

do in unbalanced data sets. However, while these measures show the significance of the classifier the organization has determined that accuracy is the most important metric.

To validate the measures of each classification model, *10-fold cross-validation* with stratified folds was used which were described in Chapter 2. This method of validation has shown to provide a better estimation of performance than lower values of K [26] while higher values of K come at higher computational cost and bias. Furthermore, stratified fold were used to reduce the variance between the different cross-validation iterations and to maintain the class distribution upon validation.

Three benchmarks were created using three different setups, described in Table 4.4, of Scikit-learn's *DummyClassifier*. These setups represent three different classifiers, that do not learn anything about the underlying relationships between the features and the target variable but uses the class distribution of the training data to classify the asset of given defect report. The benchmarks values, seen in Table 4.5 for Product 1 and Table 4.6 for Product 2, were used as lower bound for evaluating the possibilities of using machine learning to classify the asset from which a defect originates. If the performance of the classification model did not exceed these benchmarks then it had not gained any knowledge about the functional relationship between the features and the asset and was therefore not considered to be useful by the telecom company.

Setup	**Description**
stratified	Classifies entries based on the distribution of the label class in the training set.
most_frequent	Classifies entries based on the most frequent label in the training set.
uniform	Classifies entries uniformly at random.

Table 4.4: Descriptions of the different setups of the DummyClassifier used in this thesis.

The benchmark values, presented in Table 4.5 and Table 4.6, shows that there is no correlation between the models classifications and the actual assets since the MCC is equal to or less than zero. The highest accuracy is achieved by always classifying the asset as the most frequently occurring asset in the training data for a given defect report. The models using this strategy, correctly classifies the asset of 13.1% of the defect reports for Product 1 and 9.4% of the defect reports for Product 2. However, it results in a macro-averaged precision and recall close to zero since the models fail to classify any other asset other than the most frequent one. Using any other strategy results in a lower accuracy which in most cases is close to zero.

Setup	Accuracy	Precision (Macro)	Recall (Macro)	F1-score (Macro)	MCC
stratified	8.01%	2.2%	2.44%	2.13%	-0.001
most_frequent	13.1%	0.3%	2.33%	0.54%	0.0
uniform	2.37%	2.26%	2.54%	1.65%	-0.001

Table 4.5: The benchmark values by setup for Product 1.

Setup	Accuracy	Precision (Macro)	Recall (Macro)	F1-score (Macro)	MCC
stratified	2.85%	0.8%	0.81%	0.78%	-0.002
most_frequent	9.4%	0.08%	0.85%	0.15%	0.0
uniform	0.79%	0.83%	0.93%	0.63%	-0.001

Table 4.6: The benchmark values by setup for Product 2.

4.2.6 Iteration 2

The second iteration evaluated the classification scheme seen in Figure 4.3. The first iteration did not make use of the reporter's expertise. Instead of deterministically classifying the source asset of a given defect report, the model in the second iteration would provide the reporter with a list of likely assets for the recorded attributes of the defect report. This would leave the final selection of the source asset to the reporter who would use their expertise to select an asset from the list of recommended assets. Furthermore, the recommendation system could also aid the reporters who have little to no expertise since the list of possible assets would be shortened.

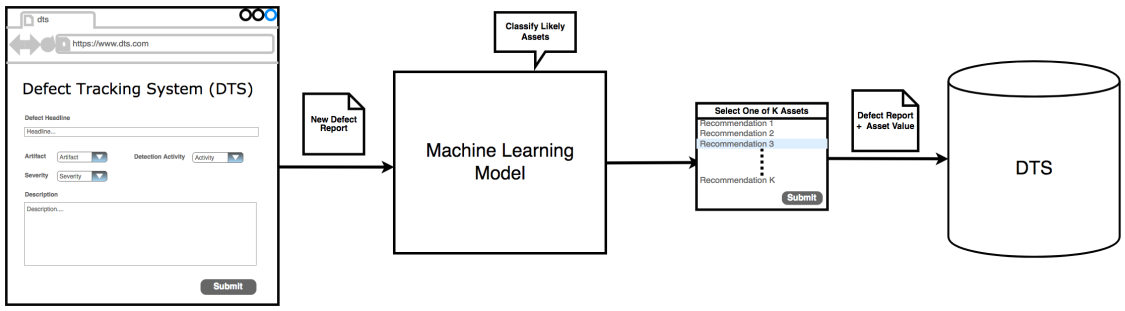


Figure 4.3: Classification scheme for Iteration 2

Similar to the first iteration, three tests were conducted to decide which attributes the model should use when classifying the recommendations. The three tests were:

1. Learning with features constructed from the categorical attributes
2. Learning with features constructed from the textual attributes
3. Learning with a combination of the features used for the previous tests

Furthermore, to evaluate which algorithm provided the best result, each test was conducted with the same set of classification algorithms, described in Table 4.3. As the performance metrics for recommendation systems differ from metrics used for classification models, different metrics than the metrics used for Iteration 1 were used. The metrics used for evaluation during the tests were *Recall@n* and *Precision@n*. Recall@n is the previously presented recall metric extended to recommen-

dation systems and is calculated by using the formula seen in Formula 4.1 where n denotes the number of recommended items.

$$recall@n = \frac{|\text{relevant recommended items } @n|}{|\text{relevant items}|} \tag{4.1}$$

Calculating the Recall@3 for the recommendations seen in Table 4.7 results in a value of approximately $0.667 = 66.7\%$. This is because the number of relevant recommended assets is 2 and the number of relevant assets is 3 across all recommendations.

Id	Recommended Assets	Correct Asset	|Relevant Recommended Assets|	|Relevant Assets|
1	[Login, Validator, Load Balancer]	Login	1	1
2	[Decoder, Login, Load Balancer]	Validator	0	1
3	[Router, Validator, Decoder]	Router	1	1

Table 4.7: Example of recommendations provided by a system that recommends three assets for a given defect report.

The precision@n metric is the precision metric extended to recommendation systems and is calculated by using the formula seen in Formula 4.2.

$$precision@n = \frac{|\text{recommended relevant item } @n|}{|\text{recommended items}|} \tag{4.2}$$

Since each defect report only has one correct asset the accuracy metric equals the recall@n metric. Furthermore, the number of recommended items, used by the precision@n metric, can be simplified to $n \times |\text{relevant items}|$. Therefore the precision@n measure can be simplified to the formula seen in Formula 4.3.

$$precision@n = \frac{recall@n}{n} \tag{4.3}$$

Using the Recall@3 calculated for the recommendations seen in Table 4.7, results in the Precision@3 which can be seen in Formula 4.4.

$$precision@3 = \frac{recall@3}{3} \approx 0.222 = 22.2\% \tag{4.4}$$

Similar to the first iteration, 10-fold cross validation with stratified folds was used to validate the measures of each classification model. Furthermore a benchmark was developed to emulate a recommendation system that uses the class distribution of the training data to provide recommendations. The benchmark values for each product can be seen in 4.8.

Product	Recall@1	Recall@3	Recall@5	Recall@10
Product 1	13.1%	36.75%	52.15%	78.71%
Product 2	9.4%	20.16%	26.66%	40.16%

Table 4.8: The benchmark values of the recommendation system for each product.

5
Results

In this chapter the results of each iteration is presented. Section 5.1 presents the results of Iteration 1 which describes the performance of classification models trained with three different feature sets. Section 5.2 presents the results of Iteration 2 which describes the performance of recommendation models trained with three different feature sets. Furthermore, in Section 5.2 the feature set with the best performing models was used to create charts which shows the balance between precision and recall for each classification model.

5.1 Iteration 1

This section presents the results of the first iteration. The first iteration aimed to evaluate which set of features and classification algorithm provided the best performance when constructing a classification model for classifying the asset from which a defect originates. For each feature set the classification model with the maximum accuracy is bolded. To complement the accuracy metric, metrics such as macro-averages of *Precision*, *Recall* and *F1-Score* are provided. Furthermore, to show the correlation between the predictions and the results the *MCC* metric has been provided.

Table 5.1 and 5.2 shows the performance of the classification models trained with features constructed from the categorical attributes of defect reports from Product 1 and Product 2 respectively. For Product 1 the maximum accuracy of **26.82%** is achieved by using **LogisticRegression**. For Product 2 the maximum accuracy of **29.56%** is achieved by using **LinearSVC**.

Classifier	Accuracy	Precision (Macro)	Recall (Macro)	F1-score (Macro)	MCC
MultinomialNB	26.28%	7.47%	8.17%	7.29%	0.186
DecisionTreeClassifier	25.93%	8.62%	8.36%	7.88%	0.183
LogisticRegression	**26.82%**	**7.87%**	**8.23%**	**7.31%**	**0.191**
KNeighborsClassifier	21.81%	8.45%	7.52%	7.15%	0.147
LinearSVC	26.43%	7.48%	8.27%	7.24%	0.185

Table 5.1: Results for models trained with features constructed from the categorical attributes of defect reports from Product 1

Classifier	Accuracy	Precision (Macro)	Recall (Macro)	F1-score (Macro)	MCC
MultinomialNB	28.72%	11.33%	13.77%	10.61%	0.267
DecisionTreeClassifier	29.3%	17.16%	18.4%	15.5%	0.273
LogisticRegression	29.47%	14.34%	16.82%	13.35%	0.275
KNeighborsClassifier	22.76%	14.66%	15.84%	13.62%	0.205
LinearSVC	**29.56%**	**14.95%**	**17.98%**	**14.33%**	**0.276**

Table 5.2: Results for models trained with features constructed from the categorical attributes of defect reports from Product 2

Table 5.3 and 5.4 shows the performance of the classification models trained with features constructed from the textual attributes of defect reports from Product 1 and Product 2 respectively. For Product 1 the maximum accuracy of **57.36%** is achieved by using **LinearSVC**. For Product 2 the maximum accuracy of **37.17%** is achieved by using **LinearSVC**.

Classifier	Accuracy	Precision (Macro)	Recall (Macro)	F1-score (Macro)	MCC
MultinomialNB	47.17%	15.96%	13.35%	12.98%	0.42
DecisionTreeClassifier	39.67%	17.75%	16.86%	16.72%	0.347
LogisticRegression	55.36%	25.04%	18.56%	19.05%	0.512
KNeighborsClassifier	38.5%	17.99%	15.65%	15.57%	0.331
LinearSVC	**57.36%**	**31.61%**	**24.72%**	**26.02%**	**0.536**

Table 5.3: Results for models trained with features constructed from the textual attributes of defect reports from Product 1

Classifier	Accuracy	Precision (Macro)	Recall (Macro)	F1-score (Macro)	MCC
MultinomialNB	27.12%	11.47%	8.31%	7.33%	0.245
DecisionTreeClassifier	24.67%	13.97%	13.44%	13.11%	0.226
LogisticRegression	35.12%	17.44%	13.0%	12.96%	0.328
KNeighborsClassifier	20.77%	15.75%	11.32%	11.22%	0.188
LinearSVC	**37.17%**	**23.41%**	**19.88%**	**20.07%**	**0.352**

Table 5.4: Results for models trained with features constructed from the textual attributes of defect reports from Product 2

Table 5.5 and 5.6 shows the performance of the classification models trained with features constructed from both the categorical and textual attributes of defect reports from Product 1 and Product 2 respectively. For Product 1 the maximum accuracy of **58.82%** is achieved by using **LinearSVC**. For Product 2 the maximum accuracy of **48.64%** is achieved by using **LinearSVC**.

Classifier	Accuracy	Precision (Macro)	Recall (Macro)	F1-score (Macro)	MCC
MultinomialNB	48.92%	18.81%	15.83%	15.55%	0.441
DecisionTreeClassifier	39.91%	18.57%	17.75%	17.72%	0.35
LogisticRegression	56.93%	28.65%	21.12%	21.79%	0.529
KNeighborsClassifier	34.44%	16.16%	14.72%	14.32%	0.286
LinearSVC	**58.52%**	**33.93%**	**27.89%**	**28.91%**	**0.549**

Table 5.5: Results for models trained with features constructed from both the textual and categorical attributes of defect reports from Product 1

Classifier	Accuracy	Precision (Macro)	Recall (Macro)	F1-score (Macro)	MCC
MultinomialNB	38.35%	22.55%	18.42%	16.8%	0.366
DecisionTreeClassifier	32.66%	23.59%	23.29%	22.61%	0.308
LogisticRegression	47.75%	34.1%	29.26%	28.68%	0.462
KNeighborsClassifier	34.25%	25.11%	25.01%	23.17%	0.324
LinearSVC	**48.64%**	**38.06%**	**35.76%**	**35.14%**	**0.472**

Table 5.6: Results for models trained with features constructed from both the textual and categorical attributes of defect reports from Product 2

5.2 Iteration 2

This section presents the results of the second iteration. The second iteration aimed to evaluate which set of features and classification algorithm provided the best performance when constructing a classification model for providing the reporter with recommendations of which asset a defect originates. For each feature set the classification model with the maximum *Recall@5* is bolded. The metrics *Recall@1*,*Recall@3* and *Recall@10* are also provided to show the relationship between Recall and the number of recommendations. Furthermore, the results of the feature set which provided the highest providing classifier for each product are complemented with a chart which shows the relationship between *Recall* and *Precision* for different number of recommendations.

Table 5.7 and 5.8 shows the performance of the classification models trained with features constructed from the categorical attributes of defect reports from Product 1 and Product 2 respectively. For Product 1 the maximum Recall@5 of **62.72%** is achieved by using **LogisticRegression**. For Product 2 the maximum accuracy of **62.35%** is achieved by using **LinearSVC**.

Classifier	Recall@1	Recall@3	Recall@5	Recall@10
MultinomialNB	26.29%	48.6%	62.54%	81.39%
DecisionTreeClassifier	25.94%	45.98%	59.37%	75.89%
LogisticRegression	**26.83%**	**48.6%**	**62.72%**	**81.66%**
KNeighborsClassifier	21.68%	37.11%	44.42%	48.56%
LinearSVC	26.36%	48.54%	62.65%	81.72%

Table 5.7: Results for recommendation model trained with features constructed from categorical attributes of defect reports from Product 1

Classifier	Recall@1	Recall@3	Recall@5	Recall@10
MultinomialNB	28.74%	48.12%	60.58%	79.58%
DecisionTreeClassifier	29.32%	48.58%	59.3%	75.86%
LogisticRegression	29.49%	49.91%	61.92%	80.95%
KNeighborsClassifier	23.02%	38.11%	45.03%	47.69%
LinearSVC	**29.39%**	**49.43%**	**62.35%**	**81.55%**

Table 5.8: Results for recommendation model trained with features constructed from categorical attributes of defect reports from Product 2

Table 5.9 and 5.10 shows the performance of the classification models trained with features constructed from the textual attributes of defect reports from Product 1 and Product 2 respectively. For Product 1 the maximum Recall@5 of **86.74%** is achieved by using **LinearSVC**. For Product 2 the maximum accuracy of **70.06%** is achieved by using **LinearSVC**.

Classifier	Recall@1	Recall@3	Recall@5	Recall@10
MultinomialNB	47.16%	70.08%	79.28%	89.23%
DecisionTreeClassifier	39.86%	40.59%	41.08%	52.43%
LogisticRegression	55.35%	76.89%	84.33%	92.13%
KNeighborsClassifier	38.49%	57.08%	64.67%	68.94%
LinearSVC	**56.35%**	**78.73%**	**86.74%**	**94.03%**

Table 5.9: Results for recommendation model trained with features constructed from textual attributes of defect reports from Product 1

Classifier	Recall@1	Recall@3	Recall@5	Recall@10
MultinomialNB	27.12%	47.51%	58.01%	73.04%
DecisionTreeClassifier	24.83%	27.16%	28.43%	31.38%
LogisticRegression	35.11%	56.53%	66.43%	79.67%
KNeighborsClassifier	20.76%	35.31%	43.29%	45.98%
LinearSVC	**37.02%**	**60.19%**	**70.06%**	**82.9%**

Table 5.10: Results for recommendation model trained with features constructed from textual attributes of defect reports from Product 2

Table 5.9 and 5.10 shows the performance of the classification models trained with features constructed from both the categorical and textual attributes of defect reports from Product 1 and Product 2 respectively. For Product 1 the maximum Recall@5 of **86.59%** is achieved by using **LinearSVC**. For Product 2 the maximum accuracy of **81.9%** is achieved by using **LinearSVC**.

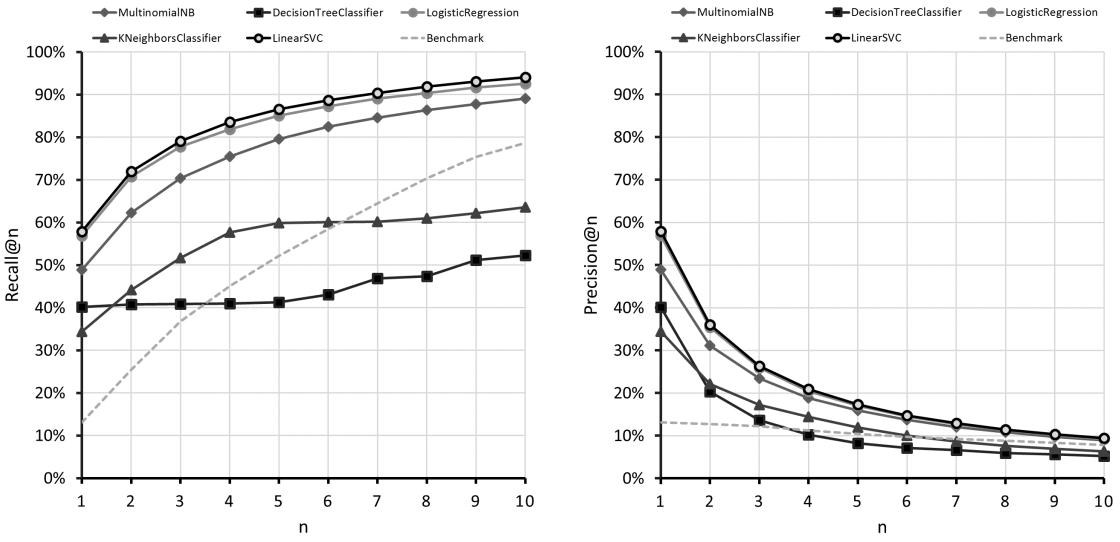
Classifier	Recall@1	Recall@3	Recall@5	Recall@10
MultinomialNB	48.91%	70.44%	79.59%	89.06%
DecisionTreeClassifier	40.17%	40.87%	41.32%	52.31%
LogisticRegression	56.92%	77.75%	85.09%	92.63%
KNeighborsClassifier	34.43%	51.7%	59.86%	63.59%
LinearSVC	**57.91%**	**79.08%**	**86.59%**	**94.1%**

Table 5.11: Results for recommendation model trained with features constructed from both the textual and categorical attributes of defect reports from Product 1

Classifier	Recall@1	Recall@3	Recall@5	Recall@10
MultinomialNB	38.36%	59.19%	70.21%	84.65%
DecisionTreeClassifier	32.67%	34.95%	35.99%	38.45%
LogisticRegression	47.76%	70.54%	80.26%	91.15%
KNeighborsClassifier	34.27%	52.71%	60.04%	61.84%
LinearSVC	**48.8%**	**71.75%**	**81.9%**	**92.34%**

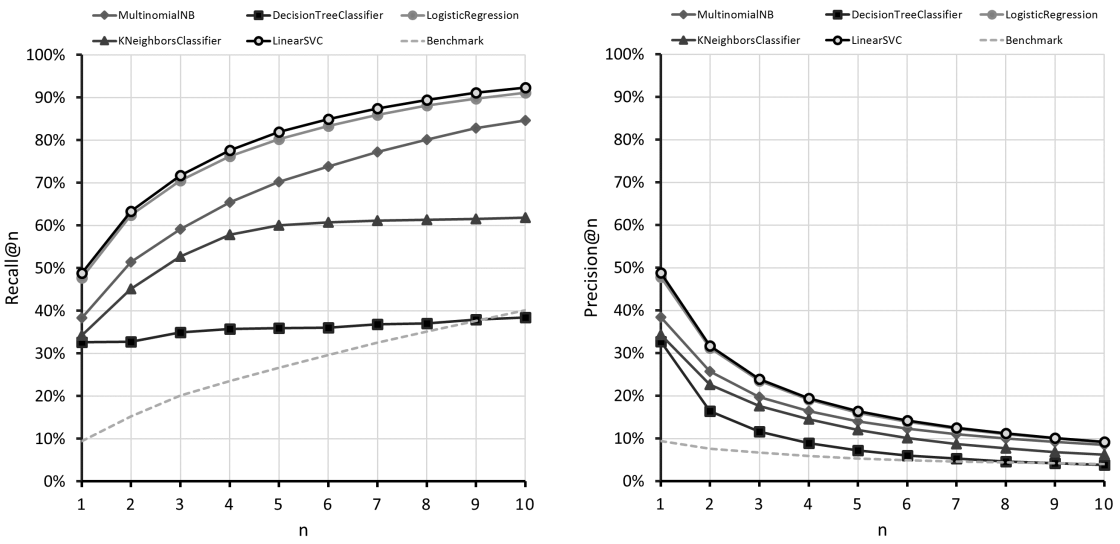
Table 5.12: Results for recommendation model trained with features constructed from both the textual and categorical attributes of defect reports from Product 2

Figure 5.1 and 5.2 shows the relationship between Recall and Precision for different numbers of recommendations of the classification models trained with features constructed from both the categorical and textual attributes of defect reports from Product 1 and Product 2 respectively. Furthermore, the figures include the values of the baseline model as a point of reference.



(a) recall@n **(b)** precision@n

Figure 5.1: Resulting (a) recall@n and (b) precision@n for recommendation model trained with features constructed from both textual and categorical attributes of defect reports from Product 1



(a) recall@n **(b)** precision@n

Figure 5.2: Resulting (a) recall@n and (b) precision@n for recommendation model trained with features constructed from both textual and categorical attributes of defect reports from Product 2

6
Discussion

In this chapter the results of each iteration is discussed. Furthermore, the possible future work and the threats to validity are discussed.

6.1 Iteration 1

The first iteration evaluated which set of features and classification algorithm provided the best performance when constructing a classification model for classifying the asset from which a defect originates. All classification models that were trained during the first iteration, achieved higher scores on all measures, than the benchmark values seen in Table 6.1 and 6.2.

Setup	Accuracy	Precision (Macro)	Recall (Macro)	F1-score (Macro)	MCC
stratified	8.01%	2.2%	2.44%	2.13%	-0.001
most_frequent	13.1%	0.3%	2.33%	0.54%	0.0
uniform	2.37%	2.26%	2.54%	1.65%	-0.001

Table 6.1: The benchmark values by setup for Product 1.

Setup	Accuracy	Precision (Macro)	Recall (Macro)	F1-score (Macro)	MCC
stratified	2.85%	0.8%	0.81%	0.78%	-0.002
most_frequent	9.4%	0.08%	0.85%	0.15%	0.0
uniform	0.79%	0.83%	0.93%	0.63%	-0.001

Table 6.2: The benchmark values by setup for Product 2.

Across all features sets used to train classification models, the models using LogisticRegression and LinearSVC provided the highest accuracies. However, the choice of algorithm did not make a significant difference on the F1-Score and MCC when only using features constructed from the categorical attributes. Comparing the measures of the best performing models using either features constructed from the textual attributes or the categorical attributes shows that the textual attributes provides more information for classifying the asset. The best performing classification model for each product were trained with features constructed from both the categorical and textual attributes using LinearSVC seen in Table 6.3.

Product	Classifier	Accuracy	Precision (Macro)	Recall (Macro)	F1-score (Macro)	MCC
Product 1	LinearSVC	58.52%	33.93%	27.89%	28.91%	0.549
Product 2	LinearSVC	48.64%	38.06%	35.76%	35.14%	0.472

Table 6.3: The best performing classification model for each product.

The best performing classification model developed with the defect reports of Product 1 achieves an accuracy of 58.52%. However, the model only achieves a macro-averaged precision 33.93% and a macro-averaged recall of 27.89%. This shows that when biasing the measures towards the least populated classes the probability of the classifier correctly labeling an unseen defect report is 33.93 % and the probability of correctly labeling a class given an instance of that class is 27.89%. The classifier achieves a MCC of 0.549 which indicates a positive correlation between the predictions and the correct labels. This implies that the classifier is making significant predictions that are determined randomly. Similar observations can be made from the results of the best performing classification model developed with the defect reports of Product 2. However, the model has a lower accuracy of 48.64% but a higher precision and recall of 38.06% and 35.75% respectively.

6.2 Iteration 2

The second iteration evaluated which set of features and classification algorithm provided the best performance when constructing a classification model for providing the reporter with recommendations of which asset a defect originates. For both products, the feature set that provided the best classification model was constructed from both the categorical and textual attributes of the defect reports. The benchmark values for evaluating the developed classification models can be seen in Table 6.4. Comparing the classification models using features constructed from both the categorical and textual attributes shows that the only models performing better than the benchmarks, for both products, were using LinearSVC, LogisticRegression or MultinomialNB.

Product	Recall@1	Recall@3	Recall@5	Recall@10
Product 1	13.1%	36.75%	52.15%	78.71%
Product 2	9.4%	20.16%	26.66%	40.16%

Table 6.4: The benchmark values of the recommendation system for each product.

The best performing classification model for providing the reporter with a list of recommendations of assets, for both products, was constructed using LinearSVC. The Recall@k and Precision@k measures of this model can be seen in Table 6.5 and Table 6.6 respectively. For both products the Recall@n increases as the number of recommendations increases while the Precision@n decreases as the number of recommendations increases.

Product	Classifier	Recall@1	Recall@3	Recall@5	Recall@10
Product 1	LinearSVC	57.91%	79.08%	86.59%	94.1%
Product 2	LinearSVC	48.8%	71.75%	81.9%	92.34%

Table 6.5: The Recall@n of the best performing classifier of each product.

Product	Classifier	Precision@1	Precision@3	Precision@5	Precision@10
Product 1	LinearSVC	57.91%	26.36%	17.32%	9.41%
Product 2	LinearSVC	48.8%	23.92%	16.38%	9.23%

Table 6.6: The Precision@n of the best performing classifier of each product.

To compare the performance of the developed recommendation models with the current list of assets, the Recall@n and Precision@n is aggregated to the F1-Score which is the harmonic mean of the two measures. Since the balance between recall and precision is something that needs to be studied the recall and precision are weighted equally. The F1-Score for the classification models can be seen in Table 6.7. Furthermore, the Recall@n, Precision@n and F1@n measures for the currently provided lists of assets can be seen in Table 6.8.

Product	Classifier	F1@1	F1@3	F1@5	F1@10
Product 1	LinearSVC	57.91%	39.54%	28.87%	17.11%
Product 2	LinearSVC	48.8%	35.88%	27.3%	16.78%

Table 6.7: The F1@n of the best performing classifier of each product.

Comparing the F1-Score for the currently provided lists of assets and the best performing classification models shows that the recommendation systems perform better regardless of the number of recommendations when weighting Recall@n and Precision@n equally.

Product	n	Recall@n	Precision@n	F1@n
Product 1	> 40	100%	$< 2.5\%$	$< 4.88\%$
Product 2	> 100	100%	$< 1\%$	$< 1.98\%$

Table 6.8: The benchmark values of the recommendation system for each product.

For instance, by providing a list of 10 recommendations, the Recall decreased by 5.9% for Product 1 and 7.66% for Product 2 while the Precision increased by more than 276% for Product 1 and 823% for Product 2.

6.3 Future Work

To further the research with classifying the asset from which a defect originates using machine learning there are a few approaches that can be taken.

6.3.1 Feature Engineering

One possible approach is to focus on the features used for classification. In this thesis, two sets of features were constructed and evaluated. However, the degree of which the classification model learns from each feature can be evaluated. For instance, the severity attribute might not correlate with the asset from which the defect originates and could therefore be excluded. This would reduce the number of features which would reduce the classification and training time and also increase the quality of the classifications.

Other than reducing the number of studied features, the existing features can be tuned. For instance, the number of words selected from the feature vector constructed by the TF-IDF algorithm was 1,000. The number of selected words can be tuned by performing cross validation on the training set with different values of the parameter. An increase of the number of selected words might increase the performance of the classification model since significant words that distinguish each defect report might have been excluded when only selecting 1,000 words.

6.3.2 Tuning Classification Algorithms

In this thesis the default values for the parameters of each classification algorithm were used. These parameters are also known as hyperparameters and can be tuned. For instance, by tuning the cost parameter of a SVM using the linear kernel which is used for selecting a hyperplane, the performance of resulting classification model might increase. If the cost parameter is too low then the model might not generalize well to unseen data since the hyperplane might not partition the classes in a way that also correctly separates the classes of unseen data.

6.4 Threats to Validity

To evaluate the validity of the results of this thesis, four types of validity are reviewed. The four types are: conclusion validity, internal validity, construct validity and external validity [57].

6.4.1 Conclusion Validity

The conclusion that using features constructed from textual attributes provide more information for classifying the asset than the features constructed from categorical attributes is valid for the studied defect reports from the telecom company. However, the quality of the description attribute differs between companies and therefore the conclusion might not apply to other companies.

6.4.2 Internal Validity

In this study the defect reports that were not resolved were removed from the data set. The reason for removing these defect reports were that the attributes of a

non-resolved defect report might change during the resolution process. To evaluate features sets and classification algorithms, the data set need to contain the correct values for each given entry. A defect report might be reopened and result in alterations to it's attributes. This might affect the performance of the constructed classification results.

6.4.3 Construct Validity

This study evaluated constructed classification models for classifying or providing recommendations from which asset a defect originates from. The models were constructed using historic defect reports under the assumption that at least one defect has been removed from every asset of the product. Therefore, the performance evaluation of the classification models do not consider assets from which defects have not been removed.

6.4.4 External Validity

The data set used in this study contained defect reports grouped by two different products from a telecom company. To generalize the results of this study to other domains, replicating this study with data sets from other domains is required. Furthermore, in other domains where the end-users submit most of the defect reports the natural language used for the describing the defects might differ. This might affect the performance of the models using features constructed from the description attribute.

7
Conclusion

This thesis aimed to improve defect localization at a telecom company by developing a classification model for classifying the asset from which a defect originates given a defect report. To develop a classification model, qualitative features needed to be constructed. Therefore it was decided that the initial set of features would be constructed from the attributes of the defect report which were mandatory to provide when reporting a defect. The attributes which were mandatory to record were the description, the severity, the detection activity and the artifact. The feature set was divided into two sets: features constructed from categorical attributes and features constructed from textual attributes.

In order to evaluate which set of features provided the most accurate classifications, five classification algorithms were selected, trained and evaluated with each set of features. The tests were conducted with defect reports from two different products and showed that combining features constructed from categorical attributes and features constructed from textual attributes provided the most accurate classifications. Furthermore, the algorithm that provided the most accurate classifications was a SVM using a linear kernel.

The initial classification models deterministically selected the asset for a given defect report and did not utilize the reporter's knowledge about the product. Therefore, a second iteration was performed where the previously developed classification models were extended to provide the reporter with recommendations of likely assets for a given defect report. The results showed that developing a SVM using a linear kernel with features constructed from both categorical and textual attributes of a given defect report provided the best recommendations. When the existing list of assets used when recording the asset attribute and the developed recommendation models were compared it showed a slight decrease in recall but a significant increase in precision.

Although the results showed that machine learning can be used for both classifying the asset and providing recommendations of likely assets, the performance of the models can be increased. This can be done by using one classification algorithm and tuning the hyperparameters used by the algorithm. Furthermore, research that only focuses on the features used for this classification task could also increase the performance. By either increasing or decreasing the feature set a more accurate classification model could be developed.

Bibliography

[1] R. D. Banker, S. M. Datar, C. F. Kemerer, and D. Zweig, "Software complexity and maintenance costs", *Commun. ACM*, vol. 36, no. 11, pp. 81–94, Nov. 1993, ISSN: 0001-0782. DOI: 10.1145/163359.163375. [Online]. Available: http://doi.acm.org.proxy.lib.chalmers.se/10.1145/163359.163375.

[2] R. L. Glass, "Frequently forgotten fundamental facts about software engineering", *IEEE Software*, vol. 18, no. 3, pp. 112–111, May 2001, ISSN: 0740-7459. DOI: 10.1109/MS.2001.922739.

[3] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization", *IEEE Trans. Softw. Eng.*, vol. 42, no. 8, pp. 707–740, Aug. 2016, ISSN: 0098-5589. DOI: 10.1109/TSE.2016.2521368. [Online]. Available: http://dx.doi.org/10.1109/TSE.2016.2521368.

[4] N. Bettenburg, S. Just, A. Schröter, C. Weiß, R. Premraj, and T. Zimmermann, "Quality of bug reports in eclipse", in *Proceedings of the 2007 OOPSLA Workshop on Eclipse Technology eXchange*, ser. eclipse '07, Montreal, Quebec, Canada: ACM, 2007, pp. 21–25, ISBN: 978-1-60558-015-9. DOI: 10.1145/1328279.1328284. [Online]. Available: http://doi.acm.org.proxy.lib.chalmers.se/10.1145/1328279.1328284.

[5] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?", in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. SIGSOFT '08/FSE-16, Atlanta, Georgia: ACM, 2008, pp. 308–318, ISBN: 978-1-59593-995-1. DOI: 10.1145/1453101.1453146. [Online]. Available: http://doi.acm.org/10.1145/1453101.1453146.

[6] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "\"not my bug!\" and other reasons for software bug report reassignments", in *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work*, ser. CSCW '11, Hangzhou, China: ACM, 2011, pp. 395–404, ISBN: 978-1-4503-0556-3. DOI: 10.1145/1958824.1958887. [Online]. Available: http://doi.acm.org/10.1145/1958824.1958887.

[7] L. Jonsson, "Increasing anomaly handling efficiency in large organizations using applied machine learning", in *2013 35th International Conference on Software Engineering (ICSE)*, May 2013, pp. 1361–1364. DOI: 10.1109/ICSE.2013.6606717.

[8] "Ieee standard classification for software anomalies - redline", *IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993) - Redline*, pp. 1–25, Jan. 2010. DOI: 10.1109/IEEESTD.2010.5953441.

[9] K. C. Youm, J. Ahn, and E. Lee, "Improved bug localization based on code change histories and bug reports", English, *Information and Software Technology*, vol. 82, no. Complete, pp. 177–192, 2017. DOI: 10.1016/j.infsof.2016.11.002.

[10] T. C. Silva and L. Zhao, *Machine Learning in Complex Networks*, 1st. Springer Publishing Company, Incorporated, 2016, ISBN: 3319172891, 9783319172897.

[11] C. Sammut and G. I. Webb, *Encyclopedia of Machine Learning*, 1st. Springer Publishing Company, Incorporated, 2011, ISBN: 0387307680, 9780387307688.

[12] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification (2Nd Edition)*. Wiley-Interscience, 2000, ISBN: 0471056693.

[13] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[14] P. Louridas and C. Ebert, "Machine learning", *IEEE Software*, vol. 33, no. 5, pp. 110–115, Sep. 2016, ISSN: 0740-7459. DOI: 10.1109/MS.2016.114.

[15] B. Lantz, *Machine Learning with R*, 2nd. Packt Publishing, 2015, ISBN: 1784393908, 9781784393908.

[16] S. B. Kotsiantis, "Supervised machine learning: A review of classification techniques", in *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, Amsterdam, The Netherlands, The Netherlands: IOS Press, 2007, pp. 3–24, ISBN: 978-1-58603-780-2. [Online]. Available: http://dl.acm.org/citation.cfm?id=1566770.1566773.

[17] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014, ISBN: 1461471370, 9781461471370.

[18] M. Kubat, *An Introduction to Machine Learning*. Springer International Publishing, 2017, ISBN: 9783319639130. [Online]. Available: http://www.springer.com/us/book/9783319348865.

[19] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. The MIT Press, 2012, ISBN: 026201825X, 9780262018258.

[20] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006, ISBN: 0387310738.

[21] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Pearson Education, 2003, ISBN: 0137903952.

[22] C. Li and H. Li, "A survey of distance metrics for nominal attributes.", 2010.

[23] J. Gorodkin, "Comparing two k-category assignments by a k-category correlation coefficient", *Computational Biology and Chemistry*, vol. 28, no. 5, pp. 367–374, 2004, ISSN: 1476-9271. DOI: https://doi.org/10.1016/j.compbiolchem.2004.09.006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1476927104000799.

[24] D. Chicco, "Ten quick tips for machine learning in computational biology", *BioData Min*, vol. 10, p. 35, Dec. 2017, 29234465[pmid], ISSN: 1756-0381. DOI: 10.1186/s13040-017-0155-3. [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC5721660/.

[25] P. Domingos, "A few useful things to know about machine learning", *Commun. ACM*, vol. 55, no. 10, pp. 78–87, Oct. 2012, ISSN: 0001-0782. DOI: 10.1145/2347736.2347755. [Online]. Available: http://doi.acm.org/10.1145/2347736.2347755.

[26] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection", in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'95, Montreal, Quebec, Canada: Morgan Kaufmann Publishers Inc., 1995, pp. 1137–1143, ISBN: 1-55860-363-8. [Online]. Available: http://dl.acm.org/citation.cfm?id=1643031.1643047.

[27] O. Kramer, *Machine Learning in Evolution Strategies*, 1st. Springer Publishing Company, Incorporated, 2016, ISBN: 331933381X, 9783319333816.

[28] Y. C. Cavalcanti, P. A. Mota Silveira Neto, I. d. C. Machado, T. F. Vale, E. S. Almeida, and S. R. d. L. Meira, "Challenges and opportunities for software change request repositories: A systematic mapping study", *J. Softw. Evol. Process*, vol. 26, no. 7, pp. 620–653, Jul. 2014, ISSN: 2047-7473. DOI: 10.1002/smr.1639. [Online]. Available: http://dx.doi.org/10.1002/smr.1639.

[29] A. Goyal and N. Sardana, "Machine learning or information retrieval techniques for bug triaging: Which is better?", *e-Informatica Software Engineering Journal*, vol. Vol. 11, nr 1, pp. 117–141, 2017.

[30] P. Bhattacharya, I. Neamtiu, and C. R. Shelton, "Automated, highly-accurate, bug assignment using machine learning and tossing graphs", *Journal of Systems and Software*, vol. 85, no. 10, pp. 2275–2292, 2012, Automated Software Evolution, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2012.04.053. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121212001240.

[31] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?", in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE '06, Shanghai, China: ACM, 2006, pp. 361–370, ISBN: 1-59593-375-1. DOI: 10.1145/1134285.1134336. [Online]. Available: http://doi.acm.org/10.1145/1134285.1134336.

[32] X. Xia, D. Lo, X. Wang, and B. Zhou, "Accurate developer recommendation for bug resolution", in *2013 20th Working Conference on Reverse Engineering (WCRE)*, Oct. 2013, pp. 72–81. DOI: 10.1109/WCRE.2013.6671282.

[33] S. Banitaan and M. Alenezi, "Tram: An approach for assigning bug reports using their metadata", in *2013 Third International Conference on Communications and Information Technology (ICCIT)*, Jun. 2013, pp. 215–219. DOI: 10.1109/ICCITechnology.2013.6579552.

[34] Z. Lin, F. Shu, Y. Yang, C. Hu, and Q. Wang, "An empirical study on bug assignment automation using chinese bug data", in *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, Oct. 2009, pp. 451–455. DOI: 10.1109/ESEM.2009.5315994.

[35] 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, no. 4, pp. 1533–1578, Aug. 2016, ISSN: 1573-7616. DOI: 10.1007/s10664-015-9401-9. [Online]. Available: https://doi.org/10.1007/s10664-015-9401-9.

[36] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing", in *Proceedings of the 29th International Conference on Software Engineering*, ser. ICSE '07, Washington, DC, USA: IEEE Computer Society, 2007, pp. 499–510, ISBN: 0-7695-2828-7. DOI: 10.1109/ICSE.2007.32. [Online]. Available: http://dx.doi.org.proxy.lib.chalmers.se/10.1109/ICSE.2007.32.

[37] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems", in *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, Jun. 2008, pp. 52–61. DOI: 10.1109/DSN.2008.4630070.

[38] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval", in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10, Cape Town, South Africa: ACM, 2010, pp. 45–54, ISBN: 978-1-60558-719-6. DOI: 10.1145/1806799.1806811. [Online]. Available: http://doi.acm.org/10.1145/1806799.1806811.

[39] Y. Tian, C. Sun, and D. Lo, "Improved duplicate bug report identification", in *2012 16th European Conference on Software Maintenance and Reengineering*, Mar. 2012, pp. 385–390. DOI: 10.1109/CSMR.2012.48.

[40] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features", in *Machine Learning: ECML-98*, Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 137–142, ISBN: 978-3-540-69781-7.

[41] R. Feldman and J. Sanger, *Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. New York, NY, USA: Cambridge University Press, 2006, ISBN: 0521836573, 9780521836579.

[42] M. Staron and W. Meding, "Predicting short-term defect inflow in large software projects: An initial evaluation", in *Proceedings of the 11th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE'07, UK: British Computer Society, 2007, pp. 33–42. [Online]. Available: http://dl.acm.org/citation.cfm?id=2227134.2227138.

[43] ——, "Predicting weekly defect inflow in large software projects based on project planning and test status", *Inf. Softw. Technol.*, vol. 50, no. 7-8, pp. 782–796, Jun. 2008, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2007.10.001. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2007.10.001.

[44] R. Rana, M. Staron, J. Hansson, M. Nilsson, and W. Meding, "A framework for adoption of machine learning in industry for software defect prediction", in *2014 9th International Conference on Software Engineering and Applications (ICSOFT-EA)*, Aug. 2014, pp. 383–392.

[45] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings", *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, Jul. 2008, ISSN: 0098-5589. DOI: 10.1109/TSE.2008.35.

[46] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research", *MIS Q.*, vol. 28, no. 1, pp. 75–105, Mar. 2004, ISSN: 0276-7783. [Online]. Available: http://dl.acm.org/citation.cfm?id=2017212.2017217.

[47] Scikit-learn. (2017). Scikit-learn Framework, [Online]. Available: http://scikit-learn.org/stable/ (visited on 05/23/2018).

[48] ——, (2017). MultinomialNB Scikit-learn, [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html (visited on 05/23/2018).

[49] ——, (2017). DecisionTreeClassifier Scikit-learn, [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier (visited on 05/23/2018).

[50] ——, (2017). Decision Tree implementation Scikit-learn, [Online]. Available: http://scikit-learn.org/stable/modules/tree.html#tree-algorithms-id3-c4-5-c5-0-and-cart (visited on 05/23/2018).

[51] ——, (2017). LogisticRegression Scikit-learn, [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (visited on 05/23/2018).

[52] ——, (2017). KNeighborsClassifier Scikit-learn, [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier (visited on 05/23/2018).

[53] ——, (2017). LinearSVC Scikit-learn, [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC (visited on 05/23/2018).

[54] ——, (2017). TfidfVectorizer Scikit-learn, [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html (visited on 05/23/2018).

[55] ——, (2017). Stop Words Scikit-learn, [Online]. Available: http://ir.dcs.gla.ac.uk/resources/linguistic_utils/stop_words (visited on 05/23/2018).

[56] ——, (2017). OneHotEncoder Scikit-learn, [Online]. Available: `http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html` (visited on 05/23/2018).

[57] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: An Introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000, ISBN: 0-7923-8682-5.