Automated Severity Assessment of Software Defect Reports

Tim Menzies
Lane Department of Computer Science,
West Virginia University
PO Box 6109, Morgantown, WV, 26506
304 293 0405
tim@menzies.us

Andrian Marcus
Department of Computer Science
Wayne State University
Detroit, MI 48202
313 577 5408
amarcus@wayne.edu

Abstract

In mission critical systems, such as those developed by NASA, it is very important that the test engineers properly recognize the severity of each issue they identify during testing. Proper severity assessment is essential for appropriate resource allocation and planning for fixing activities and additional testing. Severity assessment is strongly influenced by the experience of the test engineers and by the time they spend on each issue.

The paper presents a new and automated method named SEVERIS (SEVERity ISsue assessment), which assists the test engineer in assigning severity levels to defect reports. SEVERIS is based on standard text mining and machine learning techniques applied to existing sets of defect reports. A case study on using SEVERIS with data from NASA's Project and Issue Tracking System (PITS) is presented in the paper. The case study results indicate that SEVERIS is a good predictor for issue severity levels, while it is easy to use and efficient.

1. Introduction

NASA's software Independent Verification and Validation (IV&V) Program captures all of its findings in a database called the Project and Issue Tracking System (PITS). The data in PITS has been collected for more than 10 years and includes issues on robotic satellite missions and human-rated systems. Nowadays, similar defect tracking systems, such as Bugzilla[1], have become very popular, largely due to the spread of open source software development. These systems help to track bugs and changes in the code, to submit and review patches, to manage quality assurance, to support communication between developers, etc.

As compared to newer systems, the problem with PITS is that there is a lack of consistency in how each of the projects collected issue data. In most instances, the specific configuration of the information captured about an issue was tailored by the IV&V project to meet its needs. This has created consistency problems when metrics data is pulled across projects. While there was a set of required data fields, the majorities of those fields do not provide information in regards to the quality of the issue and are not very suitable for comparing projects.

A common issue among defect tracking systems is that they are useful for storing day-to-day information and generating small-scale tactical reports (e.g., "list the bugs we found last Tuesday"), but difficult to use for high-end business strategic analysis (e.g., "in the past, what methods have proved most cost effective in finding bugs?"). Another issue common to these systems is that most of the data is unstructured (i.e., free text). Specific to PITS is that the database fields in PITS keep changing, yet the nature of the unstructured text remains constant. In consequence, one logical choice in the analysis of defect reports is a combination of text mining and machine learning.

In this paper we present a new approach for extracting general conclusions from PITS data based on text mining and machine learning methods, which are low cost, automatic, and rapid. We designed and built a tool named SEVERIS (SEVERity ISsue assessment) to automatically review issue reports and alert when a proposed severity is anomalous. The way SEVRIS is built provides the probabilities that the assessment is correct. These probabilities can be used to guide decision making in this process. Assigning the correct severity levels to issue reports is extremely important in the process employed at NASA, as it directly impacts resource allocation and planning of subsequent defect fixing activities.

NASA uses a five-point scale to score issue severity. The scale ranges one to five, worst to dullest, respectively. A different scale is used for robotic and human-rated missions (see Table 1).

[1] http://www.bugzilla.org/

Table 1. NASA's severity scores

Severities for robotic missions
Severity 1: Prevent the accomplishment of an essential capability; or jeopardize safety, security, or other requirement designated critical.
Severity 2: Adversely affect the accomplishment of an essential capability and no work-around solution is known; or adversely affect technical, cost or schedule risks to the project or life cycle support of the system, and no work-around solution is known.
Severity 3: Adversely affect the accomplishment of an essential capability but a work-around solution is known; or adversely affect technical, cost, or schedule risks to the project or life cycle support of the system, but a work-around solution is known.
Severity 4: Results in user/operator inconvenience but does not affect a required operational or mission essential capability; or results in inconvenience for development or maintenance personnel, but does not affect the accomplishment of these responsibilities.
Severity 5: Any other issues.
Severities for human-rated missions
Severity 1: A failure which could result in the loss of the human rated system, the loss of flight or ground personnel, or a permanently disabling personnel injury
Severity 1N: A failure which would otherwise be Severity 1 but where an established mission procedure precludes any operational scenario in which the problem might occur, or the number of detectable failures necessary to result in the problem exceeds requirements.
Severity 2: A failure which could result in loss of critical mission support capability
Severity 2N: A failure which would otherwise be Severity 2 but where an established mission procedure precludes any operational scenario in which the problem might occur or the number of detectable failures necessary to result in the problem exceeds requirements.
Severity 3: A failure which is perceivable by an operator and is neither Severity 1 nor 2.
Severity 4: A failure which is not perceivable by an operator and is neither Severity 1 nor 2, nor 3.
Severity 5: A problem which is not a failure but needs to be corrected such as standards violations or maintenance issues

SEVERIS is particularly useful in the following scenarios:

- When a less-experienced test engineer has assigned the wrong severity levels.
- When experienced test engineers are operating under urgent time pressure demands, they could use SEVERIS to automatically and quickly audit their conclusions.
- For agents that can detect severity one and two-level errors with high probability, SEVERIS could check for the rare, but extremely dangerous case, that an IV&V team has missed a high-severity problem.

The paper presents a case study on using SEVERIS to assess the severity of reported issues from five NASA robotic missions. One conclusion from this case study is that the unstructured text might be a better candidate for generating severity assessments than the structured data base fields.

The main contribution of our work is that it successfully addresses an important problem, which has been largely ignored by the research community (i.e., automated defect severity assessment from very loosely structured text). Much prior work has used standard data miners, to learn software defect recognizers from historical records of static code features [11]. Those data mining methods assume that the input data is highly structured, which is rarely the case. In this research, we generate severity predictors from a data source that is so unstructured that it would defeat the previously explored data mining methods.

The main finding of this work is the success and efficiency of the solution stemming from the simplicity of its components (i.e., the combination of standard text mining and rule learning methods), which also make it easy to use and adaptable to other data sets. In addition, SEVERIS provides good estimates by analyzing only textual data extracted from defect reports. This is an important issue, as the severity scores are assigned based on human judgment (see Table 1), which reflects how the test engineer interprets a domain independent scoring guideline in the context of each project.

2. SEVERIS

SEVERIS is based on the automated extraction and analysis of textual descriptions from issue reports in PITS. Text mining techniques are used to extract the relevant features of each report, while machine learning techniques are used to assign these features with proper severity levels, based on the classifications of existing reports.

While, in its current form is specifically tailored to work with PITS reports, with little modifications, SEVRIS can be used with other defect reporting systems, such as Bugzilla.

Figure 1 depicts how SEVERIS interoperates with the human analyst or his supervisor. SEVERIS checks the validity of the severity levels assigned to issues in the following way:

- After seeing an issue in some artifact, a human analyst generates some text notes and assigns a severity level *severityX.*
- SEVERIS learns a predictor for issue severity level from logs of {*notes, severityX*}. A training module does the followings:
    1. Updates the SEVERIS beliefs and

    2. Determines how much self confidence a supervisor might have in the SEVERIS' conclusions.

- Using the learned knowledge, SEVERIS reviews the analyst's text and generates its own *severityY* level.
- If SEVERIS' proposed *severityY* differs from the *severityX* level of the human analyst, then a human supervisor can decide to review the human analyst's *severityX*. To help in that process, the supervisor can review the self confidence information to decide if they trust the SEVERIS' recommendations.

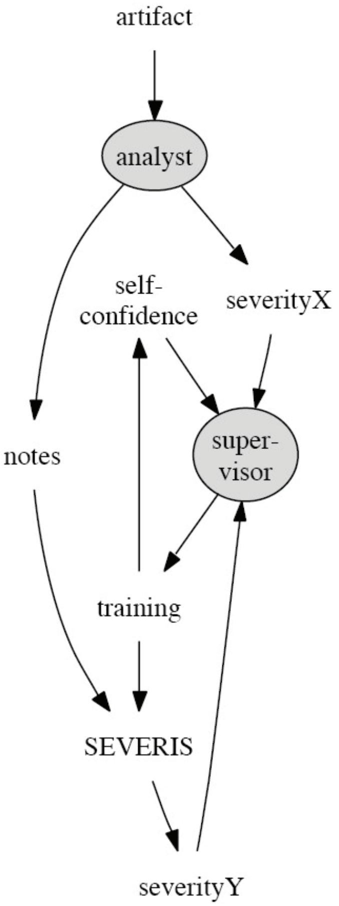

Figure 1. The workflow of SEVERIS. Gray nodes denote humans.

One issue that made the design of SEVERIS challenging is that standard machine learners work well for instances that are nearly all fully described using dozens (or fewer) attributes. On the other hand, text mining applications (e.g., analyzing PITS detect reports) must process thousands of unique words, and any particular paragraph may only mention a few of them. Therefore, before we can apply machine learning to the results of text mining, we have to reduce the number of dimensions (i.e., attributes) in the problem. To this end, we applied several methods for dimensionality reduction used in text mining, in the following order: tokenization, stop word removal, stemming, *tf*idf*, and *InfoGain*.

2.1. Tokenization

A token is a block of text which is considered as a useful part of the unstructured text. In most of the cases it is mapped to a word, but a token could be also represented by a paragraph, a sentence, a syllable or a phoneme. Tokenization represents the process of converting a stream of characters into a sequence of tokens. Tokenization is done by removing punctuation, brackets, capitals, etc. Given the nature of the PITS reports, in SEVERIS we replace punctuation with blank spaces, we remove the non-printable escape characters, and convert all words to lowercases. Additional rules can be implemented if data from other bug reporting systems is used.

2.2. Stop word removal

Stop words are commonly used words that do not carry relevant information to a specific context. A list of English stop words, i.e., prepositions, conjunctions, articles, common verbs, nouns, pronouns, adverbs and adjectives can be found at:
www.dcs.gla.ac.uk/idom/ir_resources/linguistic_utils/stop_words

Figure 2 shows a sample of the stop list used in this study. IV&V's chief engineer, Ken Costello, reviewed this list and removed "counting words" such as "one", "every", etc., arguing that "reasoning about number of events could be an important requirement". SEVERIS supports the use of a keep list of words we want to retain (but, in this study, the keep list was empty).

```
a          about    across   again     against
almost     alone    along    already   also
although   always   am       among     amongst
amongst   amount   an       and       another
any        anyhow   anyone   anything  anyway
anywhere   are      around   as        at
...        ...      ...      ...       ...
```

Figure 2. 24 of the 262 stop words used in the study

2.3. Stemming

Stemming is the process for reducing inflected (or sometimes derived) words to their stem, base or root form – generally a written word form. For example, "run", "runs", "ran", and "running", are all forms of the same root, conventionally written as "run" and the role of a stemmer is to attribute all the derived forms to the root of the lexeme. The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root.

In SEVERIS we use an implementation of the Porter stemmer [12], introduced in 1980 by Martin Porter. While not perfect, this stemmer is very widely used and became the defacto standard algorithm used for English stemming.

2.4. Tf*idf

*Tf*idf* is shorthand for "term frequency times inverse document frequency". The *tf*idf* weight defined for a word is often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. The term frequency in the given document is simply the number of times a given term appears in that document. This count is usually normalized to prevent a bias towards longer documents (which may have a higher term frequency regardless of the actual importance of that term in the document) to give a measure of the importance of the term within a particular document.

The inverse document frequency is a measure of the general importance of the term (obtained by dividing the number of all documents by the number of documents containing the term, and then taking the logarithm of that quotient).

If there be *Words* number of document and each word I appears *Word[I]* number of times inside a set of *Documents* and if *Document[I]* be the documents containing *I*, then:

*Tf*idf = Word[i]/Words*log(Documents/Document[i])*

The standard way to use this measure is to cull all but the *k* top *tf*idf* ranked stopped, stemmed tokens. The case study presented later in the paper used k = 100. The idea is that these are the most important terms for each document, whereas the rest of them can be ignored in the analysis. This is a simple form of noise reduction mechanism.

2.5. InfoGain

According to the *InfoGain* measure, the *best* words are those that *most simplifies* the target concept (in our case, the distribution of severities). We use information theory to measure concept "simplicity" in the case of SEVERIS. Statistical approaches could also be used to determine concept simplicity.

Suppose a data set has 80% severity=5 issues and 20% severity=1 issues. Then that data set has a class distribution C_0 with classes c(1) = *severity5* and c(2) = *severity1* with frequencies *n*(1) = 0.8 and *n*(2) = 0.2. The number of bits required to encode an arbitrary class distribution C_0 is $H(C_0)$ defined as follows:

$$\left.\begin{array}{r}N = \sum_{c\in C} n(c)\\ p(c) = n(c)/N\\ H(C) = -\sum_{c\in C} p(c)log_2 p(c)\end{array}\right\}$$

If *A* is a set of attributes, then the number of bits required to encode a class after observing an attribute is:

$$H(C|A) = -\sum_{a\in A} p(a) \sum_{c\in C} p(c|a) log_2(p(c|a)$$

The highest ranked attribute *Ai* is the one with the largest information gain; i.e., the one that most reduces the encoding required for the data after using that attribute; i.e.,

$$InfoGain(A_i) = H(C) - H(C|A_i)$$

where *H*(*C*) comes from Equation 1. For example, in the case study, we used *InfoGain* to find the top N = {100, 50, 25, 12, 6, 3} most informative tokens.

Once again, the assumption was that the most informative terms are enough to fully describe their corresponding documents from our application point of view. SEVERIS uses *InfoGain* to re-order the top *k* words selected from each issue report based on *tf*idf.*

By using InfoGain to rank all the terms in the data set, a significant dimensionality reduction is achieved. Another popular approach to dimensionality reduction in text retrieval is based on Singular Value Decomposition (SVD) [17] of the *tf*idf* matrix (i.e., word x document matrix) and its most popular implementation is known as Latent Semantic Indexing (LSI) [9]. Since LSI is based on statistical dimensionality reduction, it requires large data set to have good results and SVD is computationally expensive. This motivates our choice of using InfoGain over LSI, which is fairly popular in Software Engineering text retrieval tasks.

2.6. Rule learning

We used a data miner in SEVERIS to learn rules that predict for the severity attribute using the terms found above. The learner used here was a JAVA version of Cohen's RIPPER rule learner [7]. RIPPER is useful for generating very small rule sets. The generated rules are of the form *if → then*:

$$\underbrace{Feature_1 = Value_1 \wedge Feature_2 = Value_2 \wedge \ldots}_{condition} \longrightarrow \underbrace{Class}_{conclusion}$$

RIPPER, is a *covering* algorithm that runs over the data in multiple passes. Rule covering algorithms learns one rule at each pass for the *majority class*. All the examples that satisfy the conditions are marked as *covered* and removed from the data set. The algorithm then recurses on the remaining data. The output of a rule covering algorithm is an ordered decision list of rules where *rule*$_j$ is only tested if all conditions in *rule*$_{i<j}$ fail.

One way to visualize a covering algorithm is to imagine the data as a table on a piece of paper. If there exists a clear pattern between the features and the class, define that pattern as a rule and cross out all the rows covered by that rule. As covering recursively explores the remaining data, it keeps splitting the data into:

- what is easiest to explain, and
- any remaining ambiguity that requires a more detailed analysis.

In the case of SEVERIS the rules inferred from the reports connect the most informative tokens (which form the condition) of each report with its severity level (which forms the conclusion) – see Figure 4.

An alternative approach would be to employ an unsupervised clustering mechanism and cluster the document space (i.e., the reports) into five groups corresponding to each severity level. Such an approach is popular when used for topic discovery in large corpora, but its main problem relies on the use of textual similarity measures between documents as distance metrics for clustering. Our approach eliminates the need to compute such measures between all possible pairs of documents.

3. Case Study

We conducted a case study on real NASA PITS data to evaluate SEVERIS. The goal of the study was to see how well SEVERIS approximates the human evaluation captured by existing reports.

PITS is an extensible issue tracking system and users are free to add fields as needed for their own applications. From a generic standpoint, PITS allow for several different field types. These include pre-defined fields, free-form fields, attachments and context fields (dependencies). Pre-defined fields are those that captured information supplied by the database itself or that has a limited list of input such as the severity ratings shown in Table 1.

Free-form fields are usually used to capture information about the issue itself along with describing the potential impact of the issue and suggested issue resolution. Other free-form fields include the title of the issue and a field for capturing a chronology of the resolution of the issue. PITS also provides for the inclusion of attachments to the issue records. This includes copies of emails, snippets of code or other text documents, or even application specific documents.

Context fields become available depending upon the current state or condition of other fields. For example, a nominal PITS records captures the test engineering task being performed when the defect was found. These tasks are defined according to a NASA test engineering work breakdown structure that is broken down into 3 levels: Process, Task and Activity. There are two processes, which have a different set of tasks and each task generally has a different set of activities. So for example, if the task being performed was “Validate Safety Requirements”, the user would first have to select “Validate” in the process field in order to make the task field active. The task field would then only allow for the selection of tasks that fall under the “Validate” process, not tasks under the “Verification” process. The user would then select “Validate Requirements” as the task followed by the selection of “Validate Safety Requirements” in the activity field.

3.1. Objects of the case study

SEVERIS was applied to {pitsA, pitsB, pitsC, pitsD, pitsE}, five anonymous PITS projects supplied by NASA's Independent Verification and Validation Facility (see Table 2). The data is available at http://promisedata.org/. All these systems were robotic. Note that this data has no severity one issues (these are quite rare) and few severity five issues (these often not reported since they have such a low priority).

Table 2. The five data sets used in the case study. Each cell indicates the number of reports with each severity level in the corresponding data set.

	Sev. 1	Sev. 2	Sev. 3	Sev. 4	Sev. 5
pitsA	0	311	356	208	26
pitsB	0	23	523	382	59
pitsC	0	0	132	180	7
pitsD	0	1	167	13	1
pitsE	0	24	517	243	41

The data sets are quite rich, as they contain in average 775 reports with about 79,000 words. Table 3 shows the size of each data set in terms of number of reports and total word count.

Table 3. The size of the data sets in number of reports and word count

	pitsA	pitsB	pitsC	pitsD	pitsE
Reports	901	1,650	319	182	825
Words	155,165	104,052	23,799	15,517	93,750

3.2. Evaluation

It is a methodological error to assess the rules learned from a data miner using the data used in training. Such a *self-test* can lead to an over-estimate of the value of that model.

Cross-validation, on the other hand, assesses a learned model using data not used to generate it. The data is divided into, say, 10 buckets. Each bucket is set aside as a test set and a model is learned from the remaining data. This learned model is then assessed using the test set. Such cross-validation studies are the preferred evaluation method when the goal is to produce predictors intended to predict future events [13].

Mean results from a 10-way cross-validation can be assessed via a *confusion matrix* (see Table 4). In Table 4, some rule learner has generated predictions for classes {*a*, *b*, *c*, *d*}, which denote, for example, issues of severity {1, 2, 3, 4} (respectively). As shown top left of this matrix, the rules correctly classified issue reports of severity=1 as severity=1 321 times (mean results in 10-way cross-validation). However, some severity=1 issues were incorrectly classified as severity=2 and severity=3 in 12 and 21 cases (respectively).

Table 4. Sample 10-way classification results.

Classified as	tn	fn	fp	tp
a, severity=1	321	12	21	0
b, severity=2	157	41	8	0
c, severity=3	49	3	259	0
d, severity=4	21	1	2	2

Confusion matrices can be summarized as follows. Let {*tn*, *fn*, *fp*, *tp*} denote the true negatives, false negatives, false positives, and true positives (respectively). When predicting for class "*a*", then for Table 4:

- *tn* - are all the examples where issues of severity=1 were classified as severity=1; i.e., *tn*=321.
- *fn* - are all the examples where lower severity issues were classified as severity=1; i.e., *fn*=157+49+21;
- *fp* - are all the examples where severity=1 issues were classified as something else; i.e., *fp*=21+12;
- *tp* - are the remaining examples; i.e., *tp*=41+8+0+3+259+0+1+2+2.

{*tn*, *fn*, *fp*, *tp*} can be combined in many ways. Two of the most common measures used in Information Retrieval (IR) and statistical classification are *recall* and *precision.* We use these two measures here in a manner akin to the one in statistical classification. The precision for a class is the number of *true positives* (i.e., the number of items correctly labeled as belonging to the class) divided by the total number of elements labeled as belonging to the class (i.e., the sum of true positives and false positives, which are items incorrectly labeled as belonging to the class). Recall in this context is defined as the number of *true positives* divided by the total number of elements that actually belong to the class (i.e., the sum of true positives and false negatives, which are items that were not labeled as belonging to that class but should have been).

In our example, recall (or *rec*) comments on how much of the target was found.

$$rec = recall = tp/(fn+tp)$$

Precision (or *pre*) comments on how many of the instances that triggered the detector actually containing the target concept.

$$pre = precision = tp/(tp+fp)$$

In a classification task, a precision score of 1.0 for a class C means that every item labeled as belonging to class C does indeed belong to class C (but says nothing about the number of items from class C that were not labeled correctly) whereas a recall of 1.0 means that every item from class C was labeled as belonging to class C (but says nothing about how many other items were incorrectly also labeled as belonging to class C).

Often, there is an inverse relationship between precision and recall, where it is possible to increase one at the cost of reducing the other. Depending on the application, recall may be favored over precision, or vice versa. A classification system for deciding whether or not, say, a report has severity1, can achieve high precision by only classifying reports with the exact right words as severity1, but at the cost of low recall due to the number of false negatives from reports that did not quite match the specification.

Usually, precision and recall scores are not discussed in isolation. Instead, either values for one measure are compared for a fixed level at the other measure (e.g., precision at a recall level of 0.75) or both are combined into a single measure, such as the *F-measure*, which is the weighted harmonic mean of precision and recall. It has the property that if either precision or recall is low, then the f-measure is decreased. We use in this case study the F_1 measure, with recall and precision are evenly weighted:

$$f\text{-}measure = 2*pre*rec/(pre+rec)$$

The larger these values, the better the model. Table 5 shows the precision, recall, and f-measure values for the data in Table 4.

Table 5. Precision, recall, and f-measures for the data in Table 4.

severity	*precision*	*recall*	*f-measure*
1	0.91	0.59	0.71
2	0.20	0.72	0.31
3	0.83	0.89	0.86
4	0.08	1.00	0.14

3.3. Results and discussion

After stemming and stopping, the top 100 terms, based on *tf*idf*, were selected for each of the five data sets. The top 25 terms are shown in Figure 3, ranked by *InfoGain*..

rank	data set				
	a	b	c	d	e
1	rvm	fsw	softwar	switch	convent
2	sr	declar	fsw	default	the
3	script	requir	specifi	statement	capabl
4	engcntrl	arrai	command	contain	state
5	set	sr	parent	case	interfac
6	differ	parent	sc	trace	control
7	cdh	comment	trace	code	word
8	l4	us	ground	line	declar
9	indic	verifi	perform	violat	variabl
10	verifi	step	section	comment	line
11	section	gce	spec	detail	fsw
12	link	ac	matrix	avion	inst5
13	flight	scenario	cdh	spacecraft	document
14	paramet	defin	initi	appear	conduct
15	state	valid	who/what	would	septemb
16	obc	base	pip	data	set
17	system	command	child	downward	hardwar
18	onli	valu	tim	fsw	artifact
19	spacecraft	els	s919-er2342	defin	stp
20	all	test	icd	presum	condit
21	trace	includ	mu	pixel	compon
22	check	onli	spacecraft	mask	icd
23	vm	complet	verif	spec	sr
24	softwar	state	glori	process	version
25	bootload	control	comm	fpa	check

Figure 3. The top 25 terms in each data set, sorted by *InfoGain*

We have shown these lists to domain experts but, to date, we have not found any particular domain insights from these words. For example, one would expect that words related to the "softwar" stem would be ubiquitous through these reports. Yet the term appears in top 100 for each data set, even on the top position for data set PITSC. We expected that the top ranked terms would provide a good (i.e., human readable and comprehensible) conceptual approximation for each data set. While this was not achieved at data set level, as shown below, these terms can be used very effectively for the task of predicting issue severity.

The issue reports for each data set were then rewritten as frequency counts for those top 100 tokens (with the severity value for each record written to the end of line). This rewriting is similar to the vector space model (VSM) representation of documents in IR [15]. Each report becomes a vector with 100 values corresponding to the *tf*idf* scores of each terms that appears in the report. Even so, the resulting data sets are quite sparse; 10% of the cells have a frequency count of one, and frequency counts higher than 10 occur in only 1/100% of cells, or less. Obviously, choosing less than 100 terms would yield a less sparse data set, but the approximation would be less precise. An acceptable level of approximation can be determined and adjusted in each case depending on how rich the data set is.

Table 6. Confusion matrices with precision, recall, and f-measures scores for the data sets pitsA, pitsB, and pitsC, using the top 100 and the top 3 terms respectively for learning.

pitsA using top 100 terms for learning							
	tn	**fn**	**fp**	**tp**	**rec**	**pre**	**f**
sev=3	321	12	21	0	0.91	0.59	0.71
sev=4	157	41	8	0	0.20	0.72	0.31
sev=2	49	3	259	0	0.83	0.89	0.86
Sev=5	21	1	2	2	0.08	1.00	0.14
pitsA using top 3 terms for learning							
	tn	**fn**	**fp**	**tp**	**rec**	**pre**	**f**
sev=3	314	13	27	0	0.89	0.55	0.68
sev=4	158	25	24	0	0.12	0.52	0.20
sev=2	69	10	232	0	0.75	0.82	0.78
sev=5	25	0	1	0	0.00		
pitsB using top 100 terms for learning							
	tn	**fn**	**fp**	**tp**	**rec**	**pre**	**f**
sev=4	120	254	0	4	0.32	0.63	0.42
sev=3	69	445	0	7	0.85	0.62	0.72
sev=5	0	11	47	0	0.81	1.00	0.90
sev=2	2	9	0	11	0.50	0.50	0.50
pitsB using top 3 terms for learning							
	tn	**fn**	**fp**	**tp**	**rec**	**pre**	**f**
sev=4	60	317	0	0	0.16	0.64	0.25
sev=3	20	501	0	0	0.96	0.57	0.71
sev=5	3	55	0	0	0.00		
sev=2	11	11	0	0	0.00		
pitsC using top 100 terms for learning							
	tn	**fn**	**fp**	**tp**	**rec**	**pre**	**f**
sev=4	162	14	4	n/a	0.90	0.95	0.92
sev=3	7	123	0	n/a	0.95	0.89	0.92
sev=5	2	1	4	n/a	0.57	0.50	0.53
pitsC using top 3 terms for learning							
	tn	**fn**	**fp**	**tp**	**rec**	**pre**	**f**
sev=4	169	11	0	n/a	0.94	0.80	0.86
sev=3	37	93	0	n/a	0.72	0.89	0.79
sev=5	6	1	0	n/a	0.00		

Table 6 and Table 7 show the confusion matrices when learning from the top 100 tokens and top 3 tokens of each data set. In each case a set of rules was learned by SEVERIS from 90% of the data and tested on the remaining 10% to compute mean precision, mean recall, and the mean f-measures. For each data set, we repeated tested the rules 10 times, rotating the 10% testing part of the data.

Table 7. Confusion matrices with precision, recall, and f-measures scores for the data sets pitsD and pitsE, using the top 100 and the top 3 terms respectively for learning.

pitsD using top 100 terms for learning							
	tn	**fn**	**fp**	**tp**	**rec**	**pre**	**f**
sev=2	0	0	0	0	0		
sev=4	0	10	2	0	0.83	0.91	0.87
sev=3	0	1	163	0	0.99	0.98	0.99
sev=5	0	0	1	0	0.00		
pitsD using top 3 terms for learning							
	tn	**fn**	**fp**	**tp**	**rec**	**pre**	**f**
sev=2	0	0	1	0	0.00		
sev=4	0	10	2	0	0.83	0.91	0.87
sev=3	0	1	163	0	0.99	0.98	0.98
sev=5	0	0	1	0	0.00		
pitsE using top 100 terms for learning							
	tn	**fn**	**fp**	**tp**	**rec**	**pre**	**f**
sev=2	1	20	0	0	0.05	0.25	0.08
sev=3	3	490	3	20	0.95	0.70	0.80
sev=5	0	26	9	6	0.22	0.69	0.33
sev=4	0	167	1	74	0.31	0.74	0.43
pitsE using top 3 terms for learning							
	tn	**fn**	**fp**	**tp**	**rec**	**pre**	**f**
sev=2	0	21	0	0	0.00		
sev=3	0	515	0	1	1.00	0.65	0.79
sev=5	0	34	0	7	0.00		
sev=4	0	222	0	20	0.08	0.71	0.15

We chose 100 terms as this number of attributes is larger than most of the data sets used to certify the standard data miners (most of the data sets in the UCI repository of machine learning data has less than 100 attributes [3]).

As an experiment, we tried halving the attribute set size. To our surprise, there was little difference in the halved results as in the whole. This process was repeated until the number of attributes was equal to three. As shown below, even with this very small attribute set, the performance of SEVERIS did not decrease significantly.

Figure 4 shows the rules learned for dataset PITSA considering the top 100 terms. Note that the rules of Figure 4 use only a subset of the 100 terms in the data set. That is, for data set PITSA, there are a handful of terms that are most relevant to predict issue severity. Similar results hold for the other data sets. That is, even when learning from all 100 tokens, most of the rules use a few dozens terms or less.

The rules are not easy to understand as the terms are stemmed. For example, in those rules “sr” is a stemmed version of “srs”, which stands for “systems requirements specification”, a common abbreviation used in the PITS reports.

Even though few tokens were used, in many cases, the f-measures are quite large (see Table 6 and Table 7). The best results are:

- pitsA, for issues of severity=2, f = 78-86%;
- pitsA, for issues of severity=3, f = 68-71%;
- pitsB, for issues of severity=3, f = 71-71%;
- pitsC, for issues of severity=3, f = 79-92%;
- pitsC, for issues of severity=4, f = 86-92%;
- pitsD, for issues of severity=3, f = 98-98%;
- pitsD, for issues of severity=4, f = 91-91%;
- pitsE, for issues of severity=3, f = 65-70%;
- pitsE, for issues of severity=3, f = 71-75%;

These results are better than they might first appear. The first f-measure (in f = X-Y%) corresponds to learning with three tokens and the second one corresponds to learning with 100 tokens. Note how using just a vanishingly small number of tokens (i.e., three) performed nearly as well as using a much larger number of tokens. Also, recall that these are all results from a 10-way cross-validation, which usually over-estimates model error [13]. That is, the real performance values are higher than the values shown above. The f-measure is used in SEVERIS as confidence level in the prediction for new issues.

For other severities, the results are not as positive.

```
     if  CONDITION                                                then  SEVERITY  USED / INCORRECT
     if (script <= 0) and (section >= 2) and (14 >= 1) and (cdh >= 1) then    4      35.0 /   0.0
else if (sr <= 1) and (issu >= 1) and (code >= 3)                    then    4      12.0 /   1.0
else if (sr >= 2) and (rvm >= 1)                                      then    2     183.0 /   9.0
else if (sr >= 2) and (14 >= 1)                                       then    2      55.0 /   1.0
else if (within >= 2) and (state <= 0) and (system <= 0)              then    2      22.0 /   2.0
else if (verifi >= 1) and (fsw >= 1)                                  then    2      10.0 /   1.0
else if (control >= 1) and (code >= 1) and (attitud >= 4)             then    2       5.0 /   0.0
else if (13 >= 2) and (obc <= 0) and (perform <= 0)                   then    2      19.0 /   7.0
else if (script >= 1) and (trace >= 1)                                then    2       3.0 /   0.0
else if true                                                          then    3     554.0 / 219.0
```

Figure 4. Rules learned from the data set PITSA considering the top 100 terms.

Recalling Table 2, none of our data sets had severity=1 errors so the absence of severity=1 results in the above list is not a concern. However, not all datasets resulted in good predictors for severity=2 errors. In all cases where this was observed, the data set had very few examples of such issues:

- pitsB only has 22 records of severity=2;
- pitsC has zero records of severity=2;
- pitsD only has 1 record of severity=2;
- pitsE only has 21 record of severity=2;

Part of our proposed future work is to investigate what is the (data specific) minimum number of classified reports needed to learn in order to predict with higher confidence.

3.4. Threats to validity

Like any empirical data mining work, our conclusions are biased according to what data was used to generate them. Issues of *sampling bias* threaten any data mining experiment; i.e., what matters *there* may not be true *here*. For example, the sample used here comes from NASA, which works in a unique market niche. Nevertheless, we argue that results from NASA are relevant to the general software engineering industry. NASA makes extensive use of contractors who are contractually obliged (ISO-9O01) to demonstrate their understanding and usage of current industrial best practices. These contractors service many other industries; for example, Rockwell-Collins builds systems for many government and commercial organizations. For these reasons, other noted researchers such as Basili, Zelkowitz, et al. [2] have argued that conclusions from NASA data are relevant to the general software. We plan to run similar future experiments on open source date from Bugzilla repositories to reduce this bias.

We must also point out the fact that the learning rules can not be generalized across data sets, as each rule is only relevant to the data it was learnt from. Not having data with issues of severity=1 influenced the results, but we argue that adding additional categories to the learning algorithm would not change the results or the way SEVERIS works.

4. Related Work

Given the nature of defect descriptions in most bug tracking systems (i.e., textual), other researchers applied text mining techniques to analyze bug descriptions in support of various tasks.

One of the addressed problems is the detection of duplicate bug reports. In the most recent work, Wang et al. [18] use an Information Retrieval (IR) technique, vector space model (VSM) [16], to index the titles and descriptions of bug reports in Bugzilla. Textual similarities are computed between the bug descriptions, based on the VSM representation. Additional information is considered from the execution steps to reproduce the bug, described in the reports. Earlier work on the same problem was done by Runeson et al. [14], who proposed the recall rate as evaluation for their approach, also used in [18].

Closer to our work, in as much as some prediction is being made based on observed rules, are the approaches that use the textual similarity between bug descriptions to assign bugs to developers [1, 4, 8, 10]. These approaches work by indexing the bug titles and their descriptions (extracted from Bugzilla) and computing textual similarities between them, using some IR method. Based on these similarities and learning who fixed earlier bugs, these systems recommend the best developer to fix a newly reported bug. The main differences between the approaches are in their choice of IR method and in the machine learning algorithm they use, such as: support vector machines, Naïve Bayes and C4.5 are used in [1]; Probabilistic IR, VSM, support vector machines, classification and regression trees, and k-nearest neighbor are used in [10]; Naïve Bayes is used in [8].

A probabilistic IR model is used in [5, 6] to predict source code files that will change in response to a new bug, based on textual similarities between its description and other similar bug reports, which were previously fixed.

In a similar fashion, Lucene (http://lucene.apache.org) a VSM based IR tool, is used in [19] to predict the time and effort required for fixing a bug.

None of these efforts are specifically targeted at the assessment of the severity level of the bug descriptions. However, what makes this earlier work relevant to ours is that researchers recognized the importance of finding textual similarities between bug reports and correlating them with additional bug related issues. Our work on SEVERIS is based on the same principles, yet it uses different text mining and machine learning techniques, while addressing a new problem.

5. Conclusions and Future Work

Over the years, the Project Issue Tracking System has been extensively and repeatedly modified. Prior attempts at generating generalized conclusions from PITS have required significant levels of manual, hence error-prone, processing.

Here, we show that conclusions can be reached from PITS without heroic effort. Using text mining and machine learning methods, we have shown that it is possible to automatically generate predictors for severity levels from the free text entered into PITS.

Better yet, our rules are self-certifying. Our data mining generation methods builds the rules and prints performance statistics (the confusion matrix with the f-measure). For data sets with more than 30 examples of high severity issues, SEVERIS always found good issue predictors (with high f-measures). Further, SEVERIS does so using surprisingly little domain knowledge. In all cases where large f-measures were seen using the top 100 terms, similar f-measures were seen when using as few as 3 terms. It is a very exciting result since it speaks to the usability of this work.

Future work will be aimed at performing experiments on other defect data (from open source domain) and at assessing what is the optimum number of term to be used in learning and the minimum number of data points in each category.

6. Acknowledgements

This research was conducted at West Virginia University under NASA sub-contract project 100005549, task 5e, award 1002193r.

Andrian Marcus was supported in part by grants from the US National Science Foundation (CCF-0438970 and CCF-0820133).

Special thanks to Ken Costello for collecting, sanitizing, and releasing the data used in this study (note that all project identifiers were removed before the data was passed outside of NASA).

We are grateful to Denys Poshyvanyk and Sonia Haiduc from Wayne State University for their help.

Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or of NASA. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.

7. References

[1] Anvik, J., Hiew, L., and Murphy, G. C., "Who should fix this bug?" in *Proceedings 28th International Conference on Software Engineering (ICSE'06)*, 2006, pp. 361-370.

[2] Basili, V. R., McGarry, F. E., Pajerski, R., and Zelkowitz, M. V., "Lessons learned from 25 years of process improvement: the rise and fall of the NASA software engineering laboratory", in *Proceedings 24th IEEE/ACM International Conference on Software Engineering*, 2002, pp. 69 - 79.

[3] Blake, C. L. and Merz, C. J., "{UCI} Repository of machine learning databases", 1998.

[4] Canfora, G. and Cerulo, L., "How Software Repositories can Help in Resolving a New Change Request", in *Proceedings Workshop on Empirical Studies in Reverse Engineering*, 2005, pp.

[5] Canfora, G. and Cerulo, L., "Impact Analysis by Mining Software and Change Request Repositories", in *Proceedings 11th IEEE International Symposium on Software Metrics (METRICS'05)*, September 19-22 2005, pp. 20-29.

[6] Canfora, G. and Cerulo, L., "Fine Grained Indexing of Software Repositories to Support Impact Analysis", in *Proceedings International Workshop on Mining Software Repositories (MSR'06)*, 2006, pp. 105 - 111.

[7] Cohen, W. W., "Fast Effective Rule Induction", in *Proceedings 12th International Conference on Machine Learning*, 1995, pp. 115-123.

[8] Cubranic, D. and Murphy, G. C., "Automatic Bug Triage Using Text Categorization", in *Proceedings 6th International Conference on Software Engineering & Knowledge Engineering (SEKE'04)*, 2004, pp. 92–97.

[9] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R., "Indexing by Latent Semantic Analysis", *Journal of the American Society for Information Science*, 41, 1990, pp. 391-407.

[10] Di Lucca, G. A., Di Penta, M., and Gradara, S., "An Approach to Classify Software Maintenance Requests", in *Proceedings IEEE International Conference on Software Maintenance*, Montréal, Québec, Canada, 2002, pp. 93-102.

[11] Menzies, T., Greenwald, J., and Frank, A., "Data Mining Static Code Attributes to Learn Defect Predictors", *IEEE Transactions on Software Engineering*, 33, 1, 2007, pp. 2-13.

[12] Porter, M., "An Algorithm for Suffix Stripping", *Program*, 14, 3, July 1980, pp. 130-137.

[13] Quinlan, R., *C4.5: Programs for Machine Learning*, Morgan Kaufman, 1992.

[14] Runeson, P., Alexandersson, M., and Nyholm, O., "Detection of Duplicate Defect Reports Using Natural Language Processing", in *Proceedings 29th IEEE/ACM International Conference on Software Engineering (ICSE'07)*, Minneapolis, MN, 2007, pp. 499-510.

[15] Salton, G., *Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer*, Addison-Wesley, 1989.

[16] Salton, G. and McGill, M., *Introduction to Modern Information Retrieval*, McGraw-Hill, 1983.

[17] Strang, G., *Linear Algebra and its Applications*, 2nd ed., Academic Press, 1980.

[18] Wang, X., Zhang, L., Xie, T., Anvik, J., and Sun, J., "An Approach to Detecting Duplicate Bug Reports using Natural Language and Execution Information", in *Proceedings 30th International Conference on Software Engineering (ICSE'08)*, Leipzig, Germany, 10 - 18 May 2008

[19] Weiß, C., Premraj, R., Zimmermann, T., and Zeller, A., "How Long Will It Take to Fix This Bug?" in *Proceedings 4th Working Conference on Mining Software Repositories (MSR'07)*, 2007