Improving IV&V Techniques Through the Analysis of Project Anomalies: Text Mining PITS issue reports - final report

Tim Menzies

Lane Department of Computer Science and Electrical Engineering, West Virginia University, USA
tim@menzies.us
http://menzies.us

Abstract This project is in two parts. The *second part* will try to combine two (or more) of the IV&V data sources into an active monitoring framework where data collected during an active IV&V project will trigger an alert if a project becomes unusual" (and defining "unusual" is one of the goals of this project).

But before we can generalize between sources, we need to study each source in isolation to determine its strengths and weaknesses. Hence, the *first part* of this project aims to gain experience with the various IV&V data sources available to researchers like myself; i.e.

– SILAP, from the IV&V planning and scoping team;
– James Dabney's Bayes networks that describe the IV&V business practices of the L3 IV&V contractor;
– The PITS issue tracking data;
– The LINKER database project that intends to join PITS to other data sources;
– Balanced score card strategy maps from NASA Langley.
– and the COCOMO data sets from JPL.

This is the second year of a three year project that started in June 2006. The project is data-rich project and much progress has already been achieved.

– At SAS'06, a preliminary report described what had been learned from the SILAP data. A ranking was offered on the most common IV&V work-breakdown structure (WBS) activities. This ranking can be used for (e.g.) identifying what WBS tasks would benefit most from optimization.
– This report on SILAP was finalized in the first part of 2007. In summary, there exists a very strong signal in the SILAP data for issue frequency and severity.
– In October'06, a preliminary report was delivered on the Bayes network. On a limited case study, it was shown that Bayes nets and treatment learning could generate parsimonious explanations for project events.
– A preliminary report on text mining from the PITS issues tracking database that generated an expert system which audited a test engineer's proposed severity level.

This document updates the preliminary PITS report. Before, the PITS report studied two projects with a limited range of severities (mostly severity 3 and 4). Here, we explore five projects with a much wider range of severities. The results from the PITS preliminary report is confirmed. Using text mining, PITS can be used to generate an expert system that audits a test engineer's proposed severity level for an issue.

Credits: This work was made possible due to the heroic efforts of Ken Costello (chief engineering at NASA IV&V) who provided the PITS defect reports. The text mining technology used here was inspired by the trace-ability work of Jane Hayes and Alex Dekhtyar. Alex was particularly helpful is mapping out the ABCs of text mining. Jane also offered extensive advice on how to extend the current system. This research was conducted at West Virginia University under NASA sub-contract project 100005549, task 5e, award 1002193r.

Revision history : Much of the introductory and exposition text of this document comes from the preliminary report. However, all the results of this document have been recomputed for the five new projects. Also new in this report are the rule minimization results.

Contents

List of Figures

1 Introduction: We don't need another hero

NASA's software 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.

It is difficult, to say the least, to generate conclusions from a moving target like PITS. Several heroic studies have made significant conclusions using PITS data (see Figure 1). These studies were heroic in the sense that the "heroes" reached their goals after tedious and complex struggling. Worse, the extracted data was only accessible with the help of NASA civil servants- a scarce and expensive resource.

The problem with PITS is that there is a lack of consistency in how each of the projects collected issue data. In virtually all 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.

NASA is very aware of the problems with PITS and is taking active steps to improve it. At the time of this writing, there is an on-going effort to implement a mandatory data set in each IV&V project database to support IV&V effectiveness metrics. This effort has been in development for about a year and is currently being executed by several projects. However, it is too early to make any useful observations from that data.

- Ken Costello (IV&V's chief engineer) compiled statistics for NASA headquarters that showing, in nine IV&V tasks, the majority of issues found by IV&V were found via an analysis of requirements documents.
- Marcus Fisher (IV&V's research lead) applied a "mid-course correction" to one IV&V project after checking the progress of the IV&V against historical records in PITS.
- David Raffo (University of Portland), working with Ken Costello and other civil servants, found enough cost data to partially tune his waterfall-based model of IV&V;
- In a prior report in this project, Melissa Northey (Project Manager) performed some joins across PITS to return costs for different IV&V tasks;

Fig. 1 A partial list of past heroic successes with PITS.

To be fair, PITS is hardly unique. Based on my experience with data mining at other corporations, I assert that PITS is a typical database, 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?"). Like many other databases, it takes heroes to extract information from PITS. Sadly, most of the heroes I know are so busy saving their own part of the world that they have little time to save researchers like me.

Hence, in this report, we try a new approach for extracting general conclusions from PITS data. Unlike previous heroic efforts, our text mining and machine learning methods are low cost, automatic, and rapid. We find we can build an agent to automatically review issue reports and alert when a proposed severity is anomalous. Better, the way we generated the agent means that we have probabilities that the agent is correct. These probabilities can be used to intelligently guide decision making.

An extremely surprising conclusion from this report is that the unstructured text might be a better candidate for generating lessons learned than the structured data base fields.

- While the database fields in PITS keep changing, the nature of the unstructured text remains constant.
- In other words, the reason it is so hard in the past to reason about PITS is that we have been looking at the wrong data.

If we could properly understand unstructured text, this would be a result of tremendous practical importance. A recent study[1] concluded that

- 80 percent of business is conducted on unstructured information;
- 85 percent of all data stored is held in an unstructured format (e.g. the unstructured text descriptions of issues found in PITS);
- Unstructured data doubles every three months;

That is, if we can tame the text mining problem, it would be possible to reason and learn from a much wider range of NASA data than ever before.

[1] `http://www.b-eye-network.com/view/2098`

2 Concept of Operations

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 Figure 2 and Figure 3). The data used in this report comes from robotic missions.

Using text mining and machine learning methods, this report shows that it is possible to automatically generate a *review agent* from PITS issue reports via the process of Figure 4. This agent can check the validity of the severity levels assigned to issues:

- After seeing an issue in some *artifact*, a human *analyst* generates some text *notes* and assigned a severity level *severityX*.
- An agent learns a predictor for issue severity level from logs of $\{notes, severityX\}$. A *training* module (a) updates the agent beliefs and (b) determines how much *self-confidence* a *supervisor* might have in the *agent*'s conclusions.
- Using the learned knowledge, the *agent* reviews the *analysts*'s text and generates its own *severityY* level.
- If the agent's 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 agent's recommendations.

This agent would be of useful under the following circumstances:

- 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 the agent to automatically and quickly audit their conclusions.
- For agents that can detect severity one and two-level errors with high probability, the agent could check for the rare, but extremely dangerous case, that an IV&V team has missed a high-severity problem.

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.

Fig. 2 Severities for robotic 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.

Severity 5: A problem which is not a failure but needs to be corrected such as standards violations or maintenance issues.

Fig. 3 Severities for human-rated missions.

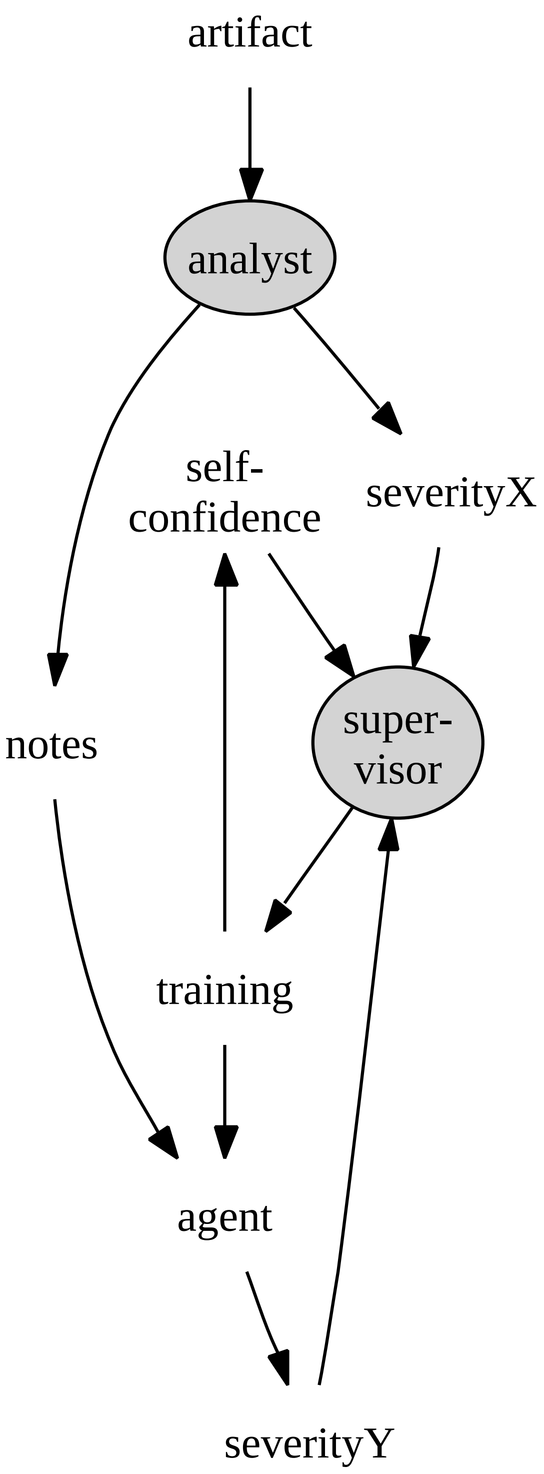

Fig. 4 An agent for reviewing issue severity levels. Gray nodes denote humans.

3 How it Works

The essential problem of text mining is *dimensionality reduction*. Standard machine learners work well for instances that are nearly all fully described using dozens (or fewer) attributes [**?**]. But 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 text mining, we have to reduce the number of dimensions (i.e. attributes) in the problem.

There are several standard methods for dimensionality reduction such as *tokenization, stop lists, stemming, Tf*Idf* and *InfoGain*. All these methods are discussed below.

3.1 Tokenization

Figure 5 shows the tokenizer used in this study:

- `Clean` replaces certain punctuation with blanks spaces.
- The file `bad.txt` contains some non-printable escape characters which are removed from the issue reports.
- Finally, `lowerCase` sends all text to lower case.

3.2 Stop lists

Another way to reduce dimensionality is to remove "dull" words via a *stop list* of "dull" words. Figure 6 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". Figure 7 shows code for a stop-list function. Note that our code supports use a *keep list* of words we want to retain (but, in this study, the keep list was empty).

3.3 Stemming

Terms with a common stem will usually have similar meanings. For example, all these words relate to the same concept.

```
clean() {
      gawk '{gsub(Bad,""); print $0}' \
         Bad=`cat $Here/bad.txt` $1 |
      sed 's/[\"\.\*\/\\\(\)\{\}\[\]><\(\)]/ /g'
}
lowerCase() {
      tr A-Z a-z $1 ;
}
```

Fig. 5 Tokenization

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
...	...	...	...	...

Fig. 6 24 of the 262 stop words used in this study.

```
stops()     { gawk '
   NR==1 {
       while (getline < Stops)  Stop[$0] = 1;
       while (getline < Keeps)  Keep[$0] = 1;
    }
    { for(I=1;I<=NF;I++)
       if (Stop[$I] && ! Keep[$I])
           $I=" "
      print $0
    }'  \
      Stops="$Here/stop_words.txt" \
      Keeps="$Here/keep_words.txt"  \
     $1
}
```

Fig. 7 Applying a stop-list.

RULE			EXAMPLE		
----			-------		
ATIONAL	->	ATE	relational	->	relate
TIONAL	->	TION	conditional	->	condition
			rational	->	rational
ENCI	->	ENCE	valenci	->	valence
ANCI	->	ANCE	hesitanci	->	hesitance
IZER	->	IZE	digitizer	->	digitize
ABLI	->	ABLE	conformabli	->	conformable
ALLI	->	AL	radicalli	->	radical
ENTLI	->	ENT	differentli	->	different
ELI	->	E	vileli	->	vile
OUSLI	->	OUS	analogousli	->	analogous
IZATION	->	IZE	vietnamization	->	vietnamize
ATION	->	ATE	predication	->	predicate
ATOR	->	ATE	operator	->	operate
ALISM	->	AL	feudalism	->	feudal
IVENESS	->	IVE	decisiveness	->	decisive
FULNESS	->	FUL	hopefulness	->	hopeful
OUSNESS	->	OUS	callousness	->	callous
ALITI	->	AL	formaliti	->	formal
IVITI	->	IVE	sensitiviti	->	sensitive
BILITI	->	BLE	sensibiliti	->	sensible

Fig. 8 Some stemming rules.

```
stemming()  { perl $Here/stemming.pl $1 ; }
```

Fig. 9 Using a downloaded stemmer.

- CONNECT
- CONNECTED
- CONNECTING
- CONNECTION
- CONNECTIONS

Porter's stemming algorithm [**?**] is the standard stemming tool. It repeatedly replies a set of pruning rules to the end of words until the surviving words are unchanged. The pruning rules ignore the semantics of a word and just perform syntactic pruning (e.g. Figure 8).

Porter's stemming algorithm has been coded in any number of languages[2] such as the Perl *stemming.pl* used in this study (see Figure 9).

Stemming is the end of our pre-processing (a sequence that began with the `clean` function shown above). Recall that the complete sequence was

$$clean \rightarrow lowerCase \rightarrow stops \rightarrow stems$$

This full sequence is shown in Figure 10.

[2] `http://www.tartarus.org/martin/PorterStemmer`

```
selectColumns() {
    gawk -F, '$3 {OFS=","; print $4 "," $3}' -
}
cleans() {
    for i in $Files; do
        (cd $Dir; clean tab${i}5.csv 2> /dev/null |
        lowerCase |
        stops |
        stemming |
        selectColumns > $Temp/ok_$i
    done
}
Files="a b c d e
Dir=$Root/mine/trunk/doc/07/telling/data/raw
cleans
```

Fig. 10 Preparation.

```
#update counters for all words in the record
function train()  {
    Documents++;
    for(I=1;I<NF;I++) {
        if( ++In[$I,Documents]==1)
            Document[$I]++
        Word[$I]++
        Words++
   }
}
# computer tfidf for one word
function tfidf(i) {
    return Word[i]/Words*log(Documents/Document[i])
}
```

Fig. 11 tfidf.awk.

```
tfidf() {
    gawk  -f tfidf.awk --source '
      { train() }
  END { OFS=","; for(I in Word) print I, tfidf(I) } ' $1 ;
      } ' $1
}
tfidf | sort -t, -n +0 | tail -100
```

Fig. 12 Finding the 100 highest Tf*Idf words using the $tfidf.awk$ code of Figure 11.

3.4 Tf*IDF

Tf*Idf is shorthand for "term frequency times inverse document frequency". This calculation models the intuition that jargon usually contains technical words that appear a lot, but only in a small number of paragraphs. For example, in a document describing a space craft, the terminology relating to the power supply may be appear frequently in the sections relating to power, but nowhere else in the document.

Calculating Tf*Idf is a relatively simple matter. If there be $Words$ number of document and each word I appear $Word[I]$ number of times inside a set of $Documents$ and if $Document[I]$ be the documents containing I, then:

$$Tf{*}Id = Word[i]/Words{*}log(Documents/Document[i])$$

The standard way to use this measure is to cull all but the k top Tf*Idf ranked stopped, stemmed tokens. This study used $k = 100$ (see Figure 11 and Figure 12).

3.5 InfoGain

According to the $InfoGain$ measure, the *best* words are those that *most simplifies* the target concept (in our case, the distribution of severities). Concept "simplicity" is measured using information theory. 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:

$$\left.\begin{array}{r}N = \sum_{c \in C} n(c)\\ p(c) = n(c)/N\\ H(C) = -\sum_{c \in C} p(c) log_2 p(c)\end{array}\right\} \quad (1)$$

If A is a set of attributes, 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 A_i 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.

$$InfoGain(A_i) = H(C) - H(C|A_i) \quad (2)$$

where $H(C)$ comes from Equation 1. In this study, we will use InfoGain to find the top $N \in \{100, 50, 25, 12, 6, 3\}$ most informative tokens.

3.6 Rule Learning

A data miner was then called 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 [**?**, **?**]. RIPPER is useful for generating very small rule sets. The generated rules are of the form $if \rightarrow then$:

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

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.

```
  a    b    c    d    <-- classified as
---  ---  ---  ---
321   12   21    0 |    a = 1
157   41    8    1 |    b = 2
 49    3  259    0 |    c = 3
 21    1    2    2 |    d = 4
```

Fig. 13 Sample 10-way classification results.

$precision$	$pd = recall$	f-measure	severity
0.893	0.833	0.862	1
0.586	0.907	0.712	2
0.719	0.198	0.311	3
0.667	0.077	0.138	4

Fig. 14 Some precision, recall, f-measures from Figure 13.

3.7 Assessing the Results

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 [?].

Mean results from a 10-way cross-validation can be assessed via a *confusion matrix* such as Figure 13. In that figure, some rule learner has generated predictions for classes {a,b,c,d} which denote 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-val). However, some severity=1 issues were incorrectly classified as severity=2 and severity=3 in 12 and 21 cases (respectively).

A confusion matrices can be summarized as follows. Let $\{A, B, C, D\}$ denote the true negatives, false negatives, false positives, and true positives (respectively). When predicting for class "a", then for Figure 13:

– A are all the examples where issues of severity=1 were classified as severity=1; i.e. A=321.
– B are all the examples where lower severity issues were classified as severity=1; i.e. B=157+49+21;
– C are all the examples where severity=1 issues were classified as something else; i.e. C=21+12
– D are the remaining examples; i.e. D=41+8+1=2+259+0+1+2+2.

A, B, C, D can be combined in many ways. Recall (or pd) comments on how much of the target was found.

$$pd = recall = D/(B + D) \quad (3)$$

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

$$prec = precision = D/(D + C) \quad (4)$$

The f-measure is the harmonic mean of precision and recall. It has the property that if *either* precision or recall is low, then the f-measure is decreased. The f measure is useful for dual assessments that include *both* precision and recall.

$$f\text{-}measure = \frac{2 \cdot prec \cdot pd}{prec + pd} \quad (5)$$

Note that all these measures fall in the range

$$0 \leq \{pd, prec, f\} \leq 1$$

Also, the *larger* these values, the better the model. Figure 14 shows the $precision$, $recall$, and f-measure values for Figure 13.

4 Results

4.1 Data

The above methods where applied to {a,b,c,d,e}, five anonymous PITS projects supplied by Ken Costello (see Figure 15). All these systems were robotic. Note that this data has no severity one issues (these are quite rate and few severity five issues (these often not reported since they have such a low priority).

data set	severity	number
a	1	0
	2	311
	3	356
	4	208
	5	26
b	1	0
	2	23
	3	523
	4	382
	5	59
c	1	0
	2	0
	3	132
	4	180
	5	7
d	1	0
	2	1
	3	167
	4	13
	5	1
e	1	0
	2	24
	3	517
	4	243
	5	41

Fig. 15 Data sets in the this study.

4.2 Stopping and Stemming

Figure 16 shows some disappointing results for stopping and stemming. In these data sets, stopping and stemming methods barely reduced the number of tokens.

*4.3 Tf*Idf*

Tf*Idf proved to be more powerful than stopping or stemming. Figure 17 shows that in all data sets, there exist a very small number of words with high Tf*Idf scores. These top 100 terms are shown in Figure 18 and Figure 19. We have shown these lists to domain experts but, to date, we have not found any particular domain insights from these words. However, as shown below, even if we don't understand *why* these terms were chosen, they are can be used very effectively for the task of predicting issue severity.

4.4 Learning

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- see Figure 20). As shown in Figure 21 the resulting data sets are quite sparse. This figure shows the distributions of the frequency counts of the cells in the data sets. Note that most cells have a zero frequency count; 10% of the cells have a frequency count of one, and frequency counts higher than 10 occur in only $\frac{1}{100}\%$ of cells, or less.

Figure 22 shows the rules and confusion matrix see when learning from the top 100 tokens of data set "a". This rule

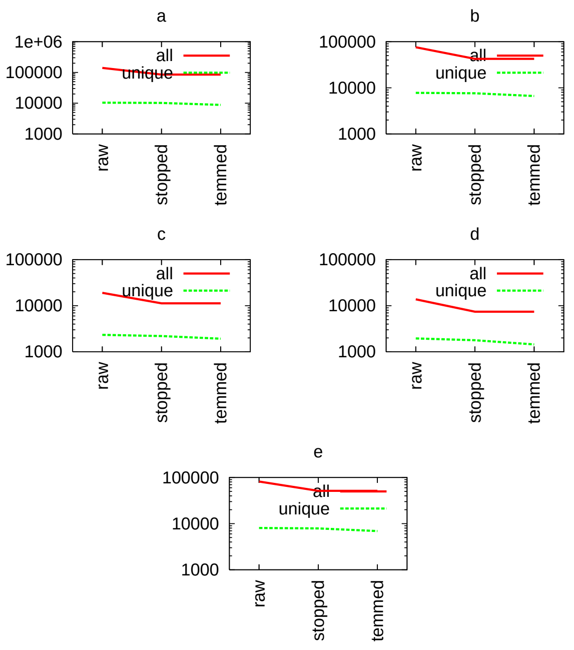

Fig. 16 Effects of stopping and stemming.

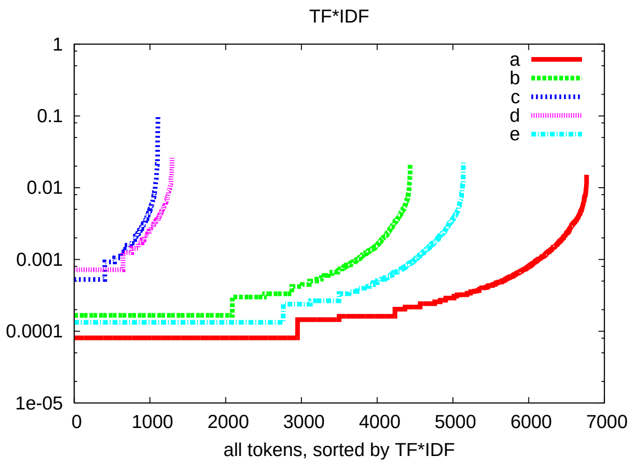

Fig. 17 Tf*Idf scoring for the stopped, stemmed tokens. Note that most tokens can be ignored since they have very low Tf*Idf scores.

	data set				
rank	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
26	capabl	all	traceabl	packet	releas
27	number	page	data	on	flight
28	vml	alloc	configur	collater	statu
29	sequenc	caus	card	scienc	implement
30	l3	verif	channel	sc	current
31	specif	flag	artifact	oper	number
32	issu	inform	downlink	logic	specifi
33	oper	fail	valid	mode	power
34	messag	trace	mechan	tabl	l4
35	support	detail	satellit	document	rqmt
36	defin	read	system	initi	list
37	fp	pse	oper	ffi	macro
38	address	ivv	includ	collect	unsign
39	code	function	instrument	onli	messag
40	uplink	condit	launch	column	assign
41	document	interfac	possibl	black	fault
42	command	on	adac	within	short
43	task	tabl	scienc	zero	test
44	rt	thruster	electr	and/or	rev
45	note	document	tp	point	time
46	monitor	initi	mode	apertur	design
47	ground	true	safehold	support	mode
48	accept	the	note	fault	jpl
49	load	in	_vbuf	exist	clear
50	initi	fprintfsetup	common2/includ/vbufh	/line	data

Fig. 18 Top 1 to 50 terms found by TF*IDF, sorted by infogain.

	data set				
rank	a	b	c	d	e
51	calcul	correct	'common2 hdlclite hdlclitec'	cadenc	oper
52	attitud	rate	struct	dure	int
53	telemetri	two	refer	all	paramet
54	fsw	end	list	store	tefsw
55	within	reset	store	csc	rafsw
56	point	req	packet	momentum	read
57	dure	telemetri	all	fine	all
58	log	packet	telemetri	target	inst6
59	packet	address	point	kav	refer
60	receiv	buffer	ap	protect	inst9
61	rate	counter	syspciinbyt	second	function
62	fault	list	interfac	autonom	reset
63	event	uint32	'common2 gener tlm_cmdc'	capabl	inst4
64	reset	rvtm	'glori cdh comm genportc'	level	command
65	process	error	'common2 gener com_cmdc'	note	configur
66	mode	level	detail	manag	enabl
67	refer	issu	rate	ground	dl
68	memori	note	em_map_pageunsign	command	us
69	engin	can	common2/includ/emsfunch	ccd	code
70	error	data	int	implic	trace
71	data	plan	collect	bin	long
72	second	sdn	'common2 router routerc'	long	flow
74	enabl	paramet	subsystem	appar	discret
75	perform	number	valu	perform	chart
76	ac	algorithm	'common2 vbuf vbufc'	short	c
77	transit	specifi	short	flight	b
78	includ	execut	compar	engin	gener
79	design	exampl	control	field	posit
80	time	void	capabl	set	spacecraft
81	execut	time	hlite_freestruct	hardwar	section
82	arrai	line	us	valu	m
83	specifi	current	soh	softwar	defin
84	control	set	float	implement	case
85	respons	review	char	enter	fail
86	current	indic	vbuf_freestruct	respect	call
87	checksum	case	power	could	initi
88	interrupt	variabl	unsign	nim	l5
89	power	specif	accuraci	smear	actuat
90	case	chang	commun	us	descript
91	tabl	section	asec	fg	tabl
92	singl	code	document	signal	subsystem
93	list[	statement	'common2 gener mm_utilc'	segment	valu
94	dump	instanc	long	fulli	softwar
95	us	updat	specif	error	issu
96	valu	procedur	compon	safe	error
97	scrub	need	orbit	design	switch
98	safe	check	ignor	requir	level
99	procedur	softwar	_hlite_cntl_blk	check	requir
100	word	charact	common2/includ/hl_protoh	period	temperatur

Fig. 19 Top 51 to 100 terms found by TF*IDF, sorted by infogain.

```
NR ==1 {
# grab the words we want to count
while (getline < "top100") Want[$0] = 1;
    # write the header
    for(I in Want)
              printf("%s,",I);
    print "severity"
}
NR > 1 { # rewrite each record as counts of "Want"
        gsub(/ /,"",$2);  counts($1,Want,  $2)
}
function counts(str,want,klass,   sum,out,i,j,n,tmp,got) {
    n=split(str,tmp," ");
    for(i=1;i<=n;i++)
        if (tmp[i] in want)
             got[tmp[i]]++;
    for(j in want) {
        sum += got[j]
        out = out got[j]+0 ",";
    }
    if (sum)
        print out "_" klass
}
```

Fig. 20 Re-writing issue reports as frequency counts.

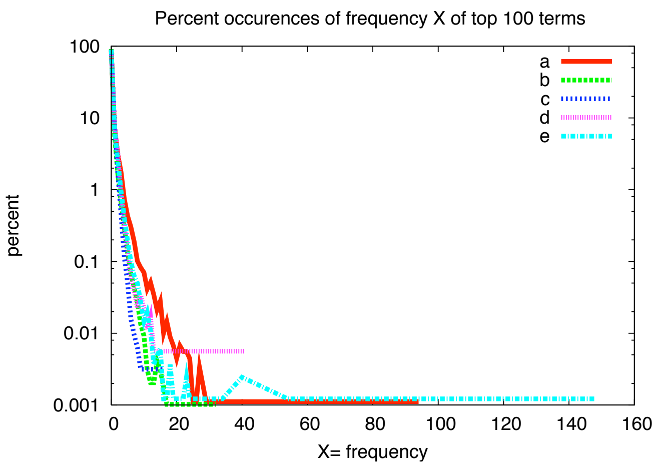

Fig. 21 Frequency counts seen in the cells of our data. Note that our data is mostly sparse: only 10% of the time (or less) were there frequency counts at or over 1. In most data sets, frequency counts of 0 (i.e. empty cell) appeared in 90% of the cells..

set is a little hard to read so Figure 23 shows the same experiment, but only using the top 3 ranked tokens. In those rules `sr` is a stemmed version of `srs`; i.e. systems requirements specification. Note that the rules of Figure 22 use only a subset of the 100 terms in the data set. That is, for data set "a", there exists a handful of terms that most predict for issue severity. Similar results hold for same results repeat for data sets {b,c,d,e} (see Figure 24 to Figure 31). That is, even when learning from all 100 tokens, most of the rules use a few dozens terms or less. Even though few tokens were used, in many cases, the f measures are quite large:

- Data set "a", for issues of severity=2, $f = 78...82\%$;
- Data set "a", for issues of severity=3, $f = 69...71\%$;
- Data set "b", for issues of severity=3, $f = 70...71\%$;
- Data set "c", for issues of severity=3, $f = 80...92\%$;
- Data set "c", for issues of severity=4, $f = 86...92\%$;
- Data set "d", for issues of severity=3, $f = 96...98\%$;
- Data set "d", for issues of severity=4, $f = 87...87\%$;
- Data set "e", for issues of severity=3, $f = 79...80\%$;

These results are better than they might first appear:

- These results are listed in the format, e.g. of $f = 79...80\%$. and show the results from using the $N = 3...N = 100$ tokens. Note how using just a vanishingly small number of tokens performed nearly as well as using a much larger number of tokens.
- Recall that these are all results from a 10-way cross-validation which usually over-estimates model error [?], That is, the real performance values are *higher* than the values shown above.

For other severities, the results are not as positive. Recalling Figure 15, 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:

- Data set “b”, only has 22 records of severity=2;
- Data set “c”, has zero records of severity=2;
- Data set “d”, only has 1 record of severity=2;
- Data set “e”, only has 21 record of severity=2;

5 Discussion

Over the years, the Project Issue Tracking System (PITS) 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 those statistics, these rules support the following dialogue:

> *Tim wrote the problem report and he says this is a severity 5 issue. But the agent says that its a severity 3 issue with probability 83%. Hmmm... the agent seems pretty sure of itself- better get someone else to take a look at the issue.*

When this work began, we thought that we were conducting a baseline text mining experiment that would serve as a (low) baseline against which we could assess more sophisticated methods. However, for data sets with more that 30 examples of high severity issues, we always found good issue predictors (with high f-measures).

Further, we did so using surprisingly little domain knowledge. In call cases where large f-measures were seen using the top 100 terms, similar f measures were seen when using 3 terms. This is a very exciting result since it speaks to the *usability* of this work. It would be a simple to matter to apply these rules. E Given that a few frequency counts are enough to predict for issue severity, even a manual method would suffice.

We end this report with one caution. As seen in Figure 22 to Figure 31, the learned predictors are different for different data sets. We hence recommend adding these text mining tools to PITS and, on a regular basis, generate new rules relevant to just one project.

References

1. R. A. Baeza-Yates and B. Ribeiro-Neto, editors. *Modern Information Retrieval*. Addison-Wesley, 1999.
2. W. Cohen. Fast effective rule induction. In *ICML’95*, pages 115–123, 1995. Available on-line from `http://www.cs.cmu.edu/~wcohen/postscript/ml-95-ripper.ps`.
3. J. Demsar. Statistical comparisons of clasifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006. Avaliable from `http://jmlr.csail.mit.edu/papers/v7/demsar06a.html`.
4. M. Porter. An algorithm for suffix stripping. In K. S. Jones and P. Willet, editors, *Readings in Information Retrieval, San Francisco: Morgan Kaufmann*. 1997.
5. G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw Hill, 1983.
6. I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.
7. I. H. Witten and E. Frank. *Data mining. 2nd edition*. Morgan Kaufmann, Los Altos, US, 2005.

```
     if  CONDITION                                                    then  SEVERITY  USED / INCORRECT
     if (script <= 0) and (section >= 2) and (l4 >= 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 (l4 >= 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 (l3 >= 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

   a   b   c   d   <-- classified as
 321  12  21   0 |   a = 3
 157  41   8   1 |   b = 4
  49   3 259   0 |   c = 2
  21   1   2   2 |   d = 5
```

Fig. 22 Data set “a”; top 100 tokens; learned rules.

```
     if  CONDITION              then  SEVERITY  USED / INCORRECT
     if (rvm <= 0) and (sr = 3) then     4       52.0 /  21.0
else if (sr >= 2)               then     2      289.0 /  54.0
else if true                    then     3      557.0 / 245.0


  a   b   c   d   <-- classified as
 314  13  27   0 |   a = 3
 158  25  24   0 |   b = 4
  69  10 232   0 |   c = 2
  25   0   1   0 |   d = 5
```

Fig. 23 Data set “a”; top 3 tokens; learned rules.

```
     if  CONDITION                                                      then  SEVERITY  USED / INCORRECT
     if (arrai >= 2) and (line >= 1) and (us <= 1)                      then    2       7.0 /   0.0
else if (base >= 4)                                                     then    2       3.0 /   0.0
else if (fsw <= 0) and (declar >= 1)                                    then    4      86.0 /  26.0
else if (fsw <= 0) and (complet >= 1) and (section <= 0)                then    4      27.0 /   4.0
else if (fsw <= 0) and (statement >= 1) and (need <= 0) and (valu <= 0) then    4      36.0 /  10.0
else if true                                                            then    3     819.0 / 332.0

   a   b   c   d   <-- classified as
 120 253   0   4 |   a = 4
  69 445   0   7 |   b = 3
  11  47   0   0 |   c = 5
   2   9   0  11 |   d = 2
```

Fig. 24 Data set “b”; top 100 tokens; learned rules.

```
     if  CONDITION                  then  SEVERITY  USED / INCORRECT
     if (fsw <= 0) and (declar >= 1) then    4      94.0 /  34
else if true                        then    3     884.0 / 383.0

   a   b   c   d   <-- classified as
  60 317   0   0 |   a = _4
  20 501   0   0 |   b = _3
   3  55   0   0 |   c = _5
  11  11   0   0 |   d = _2
```

Fig. 25 Data set “b”; top 3 tokens; learned rules.

```
      if  CONDITION                                        then  SEVERITY  USED / INCORRECT
     if (section >= 2) and (matrix >= 1) and (icd >= 1)    then      5      7.0 /  1.0
else if (softwar >= 1)                                     then      3     95.0 /  3.0
else if (parent <= 0) and (trace <= 0)                     then      3     48.0 / 14.0
else if true                                               then      4    167.0 /  4.0

   a   b   c   <-- classified as
 162  14   4 |   a = _4
   7 123   0 |   b = _3
   2   1   4 |   c = _5
```

Fig. 26 Data set “c”; top 100 tokens; learned rules.

```
        if  CONDITION               then  SEVERITY  USED / INCORRECT
        if  (softwar >= 1)          then      3     95.0 /  3.0
   else if  true                    then      4    222.0 / 44.0

      a   b   c   <-- classified as
    169  11   0 |   a = _4
     37  93   0 |   b = _3
      6   1   0 |   c = _5
```

Fig. 27 Data set “c”; top 3 tokens; learned rules.

```
        if  CONDITION               then  SEVERITY  USED / INCORRECT
        if (switch >= 2)            then     4     11.0 / 1.0
   else if true                     then     3    167.0 / 4.0)

      a   b   c   d   <-- classified as
      0   0   1   0 |   a = _2
      0  10   2   0 |   b = _4
      0   1 163   0 |   c = _3
      0   0   1   0 |   d = _5
```

Fig. 28 Data set “d”; top 100 tokens; learned rules.

```
        if  CONDITION               then  SEVERITY  USED / INCORRECT
        if  (switch >= 2)           then     4      11.0 / 1.0
   else if  true                    then     3     167.0 / 4.0)

      a   b   c   d   <-- classified as
      0   0   1   0 |   a = _2
      0  10   2   0 |   b = _4
      0   1 163   0 |   c = _3
      0   0   1   0 |   d = _5
```

Fig. 29 Data set “d”; top 3 tokens; learned rules.

```
        if  CONDITION                                                      then  SEVERITY  USED / INCORRECT
        if (trace >= 1) and (test >= 1) and (case >= 3)                    then     2     3.0  /   0.0
   else if (error >= 1) and (line >= 2)                                    then     2     3.0  /   0.0
   else if (convent >= 2) and (declar <= 0) and (function <= 0)            then     5     6.0  /   0.0
   else if (case >= 2) and (sr >= 3)                                       then     5     6.0  /   1.0
   else if (refer >= 1) and (section >= 3) and (refer <= 1)                then     5     7.0  /   2.0
   else if (the >= 1) and (fsw >= 2)                                       then     4    47.0  /   5.0
   else if (control <= 0) and (convent >= 3)                               then     4    25.0  /   3.0
   else if true                                                            then     3   723.0  / 214.0

      a   b   c   d   <-- classified as
      1  20   0   0 |   a = _2
      3 490   3  20 |   b = _3
      0  26   9   6 |   c = _5
      0 167   1  74 |   d = _4
```

Fig. 30 Data set “e”; top 100 tokens; learned rules.

```
        if  CONDITION                     then  SEVERITY  USED / INCORRECT
        if  (the >= 1) and (convent >= 3) then    4       30.0 /   8.0
   else if  true                                  3      790.0 / 275.0

      a   b   c   d   <-- classified as
      0  21   0   0 |   a = _2
      0 515   0   1 |   b = _3
      0  34   0   7 |   c = _5
      0 222   0  20 |   d = _4
```

Fig. 31 Data set “e”; top 3 tokens; learned rules.