Improving IV&V Techniques Through the Analysis of Project Anomalies: Text Mining PITS issue reports - preliminary 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 first 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.
- In early October, 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.

This report presents a preliminary report on our use of the PITS issues tracking database. It will be shown that, 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. This research was conducted at West Virginia University under NASA subcontract project 100005549, task 5e, award 1002193r.

Code: All software discussed here is available from http://unbox.org/wisp/tmine under the GNU Public License (version 2: see www.gnu.org/copyleft/gpl.html).

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.

- 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.

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.

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. For example, with our system, the following dialogue is possible:

> *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.*

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.

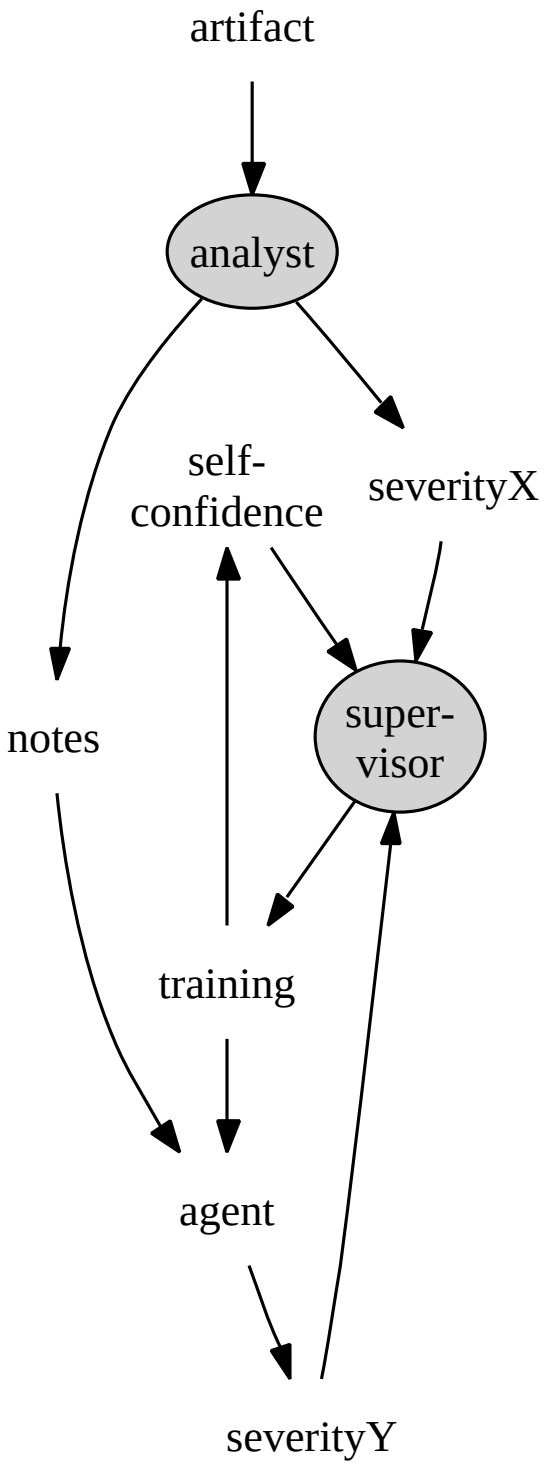

Fig. 4 An agent for reviewing issue severity levels. Gray nodes denote humans.

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 [6]. 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 [1, 5]. 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. Words are reduced to simple tokens via (e.g.) removing all punctuation remarks, then sending all upper case to lower.

This code assumes that the input is some tab separated columns, one record per line. Two special columns are recognized: the *Wanted* column (i.e. the column with the text) and a *Klass* column (i.e. the column with the goal value- in our case, the severity level). The *Wanted* column is extracted, tokenized, and printed one record per line with the associated *Klass* (severity) value.

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.

```
#### config
Want=4   # analyst's notes are in column 4
Klass=5  # severity is in column 5

#### workers
tokenize()  { column $1 | lowerCase | alphasNotLasts; }
lowerCase() { tr A-Z a-z $1 ; }
column() {
    gawk -F"\t" ' NR > 1 {if($Klass) Class=$Klass;
                          print $Goal " "  Class}
                ' Klass=${Klass} Goal=$Want $1 ; }
alphasNotLast() {
    # don't tokenize last column- it holds the class
    gawk '{ for(I=1;I<NF; I++)
                gsub(/[^abcdefghijklmnopqrstuvwxyz]/," ",$I);
                print $0
          }'  $1
}

#### main
cat report | tokenize  > words
```

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.

3.3 Stemming

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

- CONNECT
- CONNECTED
- CONNECTING
- CONNECTION
- CONNECTIONS

Porter's stemming algorithm [3] 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).

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

Fig. 9 Using a downloaded stemmer.

```
#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. 10 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. 11 Finding the 100 highest Tf*Idf words using the $tfidf.awk$ code of Figure 10.

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).

*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:

- Let there be $Words$ number of documents;
- Let some word I appear $Word[I]$ number of times inside a set of $Documents$;
- Let $Document[I]$ 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 10 and Figure 11).

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

data set	Domain	number	% of total
l	FSW	104	67%
	Ground	17	11%
	Instruments	33	21%
p	Payload	584	75%
	Spacecraft	188	24%
s	AIA	62	4%
	EVE	68	4%
	SDO	1	0%
	Ground	17	1%
	Spacecraft	1353	90%

Fig. 12 Domains in the this study.

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:

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

After discretizing numeric data[3] then 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 $N = 10$ most informative tokens.

[3] E.g. given an attribute's minimum and maximum values, replace a particular value n with $(n - min)/((max - min)/10)$. For more on discretization, see [2].

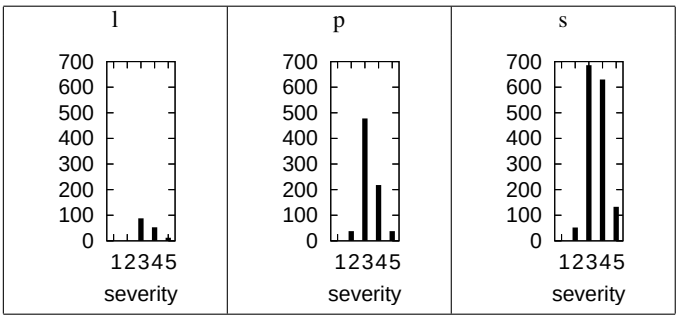

Fig. 13 Severities.

4 Results

4.1 Data

The above methods where applied to “l”, “p”, “s”; i.e. three anonymous PITS projects supplied by Ken Costello. The domains mentioned in these data sets are shown in Figure 12. This data contained 155, 773, 4661 issue reports for projects “l,p,s” (respectively); i.e. “l” was quite small and “s” was quite large. The issues were created and found in various phases (see Figure 14- note that there is no created/found data for project “s” since that project lacked phase information).

As shown in Figure 13, the severities in this data were not evenly distributed:

- Most of the severities where 3 or 4
- The data contained only one severity 1 record and 91 severity 2 records.

4.2 Stopping and Stemming

Figure 15 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: Figure 16 shows that in all three data sets, there exist a very small number of words with high Tf*Idf scores. The 100 top Tf*Idf tokens from each data set were extracted and the issue reports were rewritten as frequency counts for those top 100 tokens, with the severity value for each record written to the end of line (see Figure 17). A review of the top 100 tokens showed little similarity in the top-ranked tokens; i.e. this study of three data sets found no general pattern in the terminology associated with severity.

project=l

		Found in		
		Requirements	Design	Implementation
Created in	Requirements	28	8	79
	Design		23	
	Implementation			15

project=p

		Found in				
		Design	SW requirements design	SW Preliminary design	Implementation	SW implementation
Created in	Design	362				
	SW requirements design			50		87
	Subsystems requirements design					19
	Implementation				69	
	SW implementation					165

Fig. 14 Leakage matrices for project=l (above) and project=b (below).

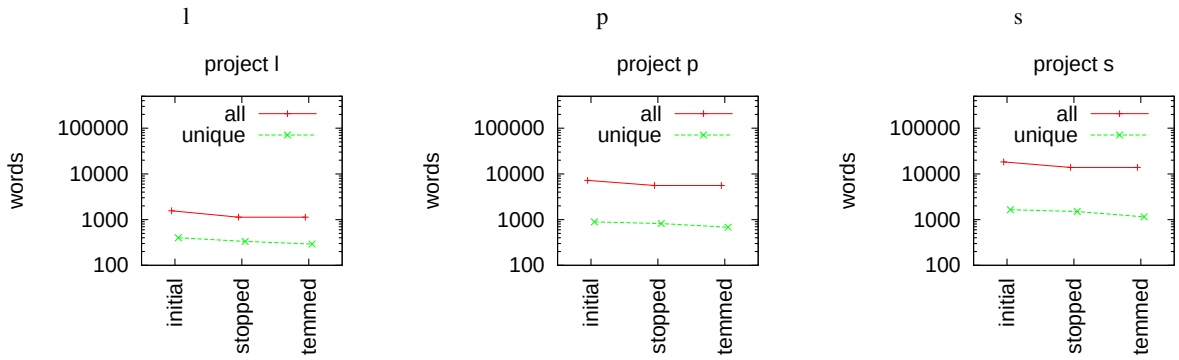

Fig. 15 Effects of stopping and stemming.

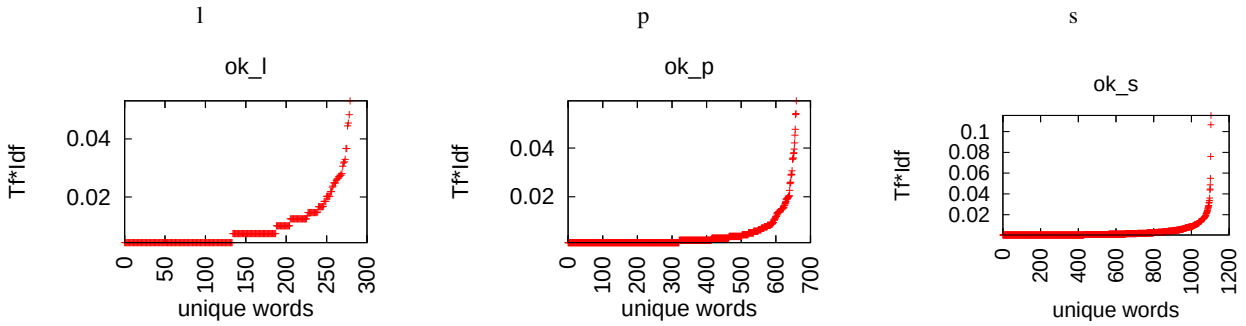

Fig. 16 Tf*Idf scoring for the stopped, stemmed tokens. Note that most tokens can be ignored since they have very low Tf*Idf scores.

```
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. 17 Re-writing issue reports as frequency counts.

4.4 Learning

A classier was then called to learn a predictor for the severity attribute using the other 100 attributes. The classifier used here was a JAVA version of Quinlan;s C4.5 decision tree learner [4, 6]. C4.5 applies InfoGain to find the best root of a tree. Data is then split according to the values of the attribute and the algorithm recurses into every split.

For each data set, the classifier was called twice:

- Once using 100 independent attributes;
- Once using the 10 independent attributes ranked as "top ten" by InfoGain.

Figure 18 and Figure 19 show the decision trees learned from 100 and 10 attributes for project "l". As might be expected, the tree learned from 100 attributes is larger than the one learned from 10 attributes.

At the bottom of Figure 18 and Figure 19 are *confusion matrices* that report how many times records of each severity were classified as severity 1,2,3,4, or 5. This matrix shows average performance results of decision trees learned from ten 90% samples of project "l", then tested on the remaining 10% of the data. These confusion matrices offer the *self-confidence* measures discussed in §2:

- For example, in Figure 18, severity 3 issues were classified as severity 3 in $\frac{76}{76+11} = 87\%$ of cases. That is, if the agent reports that the issue is severity 3, then it is highly likely that the agent is right and other severity allocations are wrong.
- However, in the same figure, the results for severity 4 classification is not as impressive: these severities were only correctly classified in $\frac{21}{21+32} = 39\%$ of the tests. That is, if the agent reports that the issue is severity 4, then it is unlikely that the agent is right.

decision tree:

```
child <= 0
|   buffer <= 0
|   |   sr <= 0
|   |   |   after <= 0
|   |   |   |   gnac <= 0
|   |   |   |   |   statement <= 0
|   |   |   |   |   |   potenti <= 0
|   |   |   |   |   |   |   fswr <= 0
|   |   |   |   |   |   |   |   sb <= 0
|   |   |   |   |   |   |   |   |   variabl <= 0: _3
|   |   |   |   |   |   |   |   |   variabl > 0: _4
|   |   |   |   |   |   |   |   sb > 0: _4
|   |   |   |   |   |   |   fswr > 0: _4
|   |   |   |   |   |   potenti > 0: _3
|   |   |   |   |   statement > 0: _4
|   |   |   |   gnac > 0: _4
|   |   |   after > 0: _4
|   |   sr > 0: _4
|   buffer > 0
|   |   cfe <= 1: _3
|   |   cfe > 1: _4
child > 0: _4
```

confusion matrix:

```
  a  b  c   <-- severity classified as
 76 11  0 |  a = 3
 32 21  0 |  b = 4
  1  0  0 |  c = 2
```

Fig. 18 Project "l"'s results, using 100 attributes.

decision tree:

```
child <= 0
|   after <= 0
|   |   gnac <= 0: _3
|   |   gnac > 0: _4
|   after > 0: _4
child > 0: _4
```

confusion matrix:

```
  a  b  c   <-- severity classified as
 87  0  0 |  a = 3
 48  5  0 |  b = 4
  1  0  0 |  c = 2
```

Fig. 19 Project "l"'s results, using 10 attributes.

decision tree:

```
convent <= 0
|   unus <= 0: _3
|   unus > 0: _4
convent > 0: _4
```

confusion matrix:

```
   a   b   c   <-- classified as
  46 171   0 |   a = _4
   3 473   0 |   b = _3
   0  38   0 |   c = _2
```

Fig. 20 Project "p"'s results, using 10 attributes.

(Note that that the classifiers of Figure 18 and Figure 19 never correctly classified any issue as severity 1 or 2- thought one severity 2 issue was incorrectly classified as severity 3. This is not a fault of our method; rather it is a quirk of the training data in Figure 13 that has no severity 1 issues and only one severity two issues. In the future, we will repeat this method for data sets with more severe issues.)

One problem with decision tree learning is that the tree can be unreadable. Project "l" is our smallest data set and its decision tree (in Figure 18) would be hard to explain to most users. Worse, other larger data sets yield far larger and far more confusing decision trees. For example, the decision tree from project "s" is ten times larger than Figure 18.

The trees learned from the ten attributes selected by InfoGain are much more readable (see Figure 19, Figure 20, Figure 21). For example, Figure 19 has only six nodes and the tree learned from ten project "s" attributes has only 40 nodes (see Figure 21). Such smaller trees have interesting performance properties:

- Smaller trees can't handle as many special cases as the larger trees. Hence, the performance of the smaller theory can be worse than the more elaborate theory. For example, the first and last lines of Figure 22 shows that the probabilities of detecting severity 2 and 4 errors is always decreased by InfoGain.
- InfoGain can also discard attributes that tend to confuse a decision tree learner. Hence, the performance of the smaller theory can be better than the more elaborate theory. For example, the middle row of Figure 22 shows that InfoGain improved the ability of our agent to detect severity 3 errors in projects "l" and "p" (but not in "s"). Some of the improvements were remarkable- InfoGain+decision tree learning for projects "l" and "p" yielded detectors with a 99% to 100% probability of detecting severity 3 errors.

decision tree:

```
arrai <= 0
|   verifi <= 0
|   |   s <= 0
|   |   |   test <= 1
|   |   |   |   requir <= 0: _4
|   |   |   |   requir > 0
|   |   |   |   |   doe <= 0
|   |   |   |   |   |   requir <= 1: _3
|   |   |   |   |   |   requir > 1: _4
|   |   |   |   |   doe > 0
|   |   |   |   |   |   requir <= 2
|   |   |   |   |   |   |   requir <= 1: _3
|   |   |   |   |   |   |   requir > 1: _4
|   |   |   |   |   |   requir > 2: _3
|   |   |   test > 1
|   |   |   |   scenario <= 1
|   |   |   |   |   line <= 0: _3
|   |   |   |   |   line > 0
|   |   |   |   |   |   scenario <= 0: _4
|   |   |   |   |   |   scenario > 0: _3
|   |   |   |   scenario > 1: _4
|   |   s > 0
|   |   |   doe <= 0
|   |   |   |   test <= 0: _4
|   |   |   |   test > 0: _3
|   |   |   doe > 0: _3
|   verifi > 0
|   |   doe <= 0
|   |   |   line <= 0
|   |   |   |   s <= 0
|   |   |   |   |   requir <= 1: _3
|   |   |   |   |   requir > 1: _4
|   |   |   |   s > 0: _4
|   |   |   line > 0: _4
|   |   doe > 0: _3
arrai > 0
|   comm <= 0: _3
|   comm > 0: _2
```

confusion matrix:

```
   a   b   c   d   <-- classified as
 411 273   0   0 |   a = _3
 212 411   0   0 |   b = _4
  18  18  16   0 |   c = _2
   0   1   0   0 |   d = _1
```

Fig. 21 Project "s"'s results, using 10 attributes.

	l			p			s		
severity	100	10	$\frac{100}{10}$	100	10	$\frac{100}{10}$	100	10	$\frac{100}{10}$
2	0	0	0.00	52	0	0.0	38	31	0.82
3	83	100	1.20	90	99	1.1	73	60	0.82
4	39	9	0.23	52	21	0.4	71	65	0.91

Fig. 22 Perecent robabilities of detection using 100 attributes or just the 10 selected by InfoGain for projects "l", "p", "s". The difference in performance is shown in the columns marked with $\frac{100}{10}$ (and numbers greater than one mean that InfoGain has made the performance *better*).

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, the detectors learned via the methods described in this report come with a set of self-confidence measures. In some cases, that self-confidence was very high (e.g. a 100% confidence in our ability to recognize severity 3 errors in project "l" using the tiny decision tree of Figure 19). That is, the user of our proposed agent would know when to trust, and when to ignore, the agent's conclusions.

The current implementation of the agent has its limits. For example:

- The training data used in the study had very few level 1 and level 2 severity problems. Hence, there was not enough data for our agent to learn high probability detectors for high severity issues.
- In each of the three data sets studied here, useful predictors were learned for each data set but the data sets were so different that the predictors are not general outside of their training domain.
- Some of the predictions have low probability, particularly for severity 1 and severity 2 errors.

Hence, for the next version of this report, we will:

- Repeat this study using different data sets containing higher severities.
- Study more data sets looking for patterns that are general to multiple training domains.
- Explore a wider range of text mining techniques to improve our probability of detecting the severity. For example:
  - This study used various "magic numbers"; e.g. the 100 Tf*Idf token and the top 10 InfoGain-scored tokens. Subsequent studies should check how changing those magic numbers effects the outcome.
  - Stemming and stopping seemed to be of little value in this study. Perhaps our performance changes if we ignore one or both of them?
  - Work with di-grams/ tri-grams of tokens instead of one-grams;
  - Further filter the tokens by removing tokens that score less or more than some minimum or maximum counts;
  - Using domain knowledge, add in special "keep words" representing core domain concepts;
  - Augment the syntactic analysis used in this study with a more semantic analysis (e.g. more background knowledge of error types);
  - Normalize the Tf*Idf counts using different filters (e.g. logarithms).

References

1. R. A. Baeza-Yates and B. Ribeiro-Neto, editors. *Modern Information Retrieval*. Addison-Wesley, 1999.
2. J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *International Conference on Machine Learning*, pages 194–202, 1995.
3. M. Porter. An algorithm for suffix stripping. In K. S. Jones and P. Willet, editors, *Readings in Information Retrieval, San Francisco: Morgan Kaufmann*. 1997.
4. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufman, 1992. ISBN: 1558602380.
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.