

Panel: Context-Dependent Evaluation of Tools for NL RE Tasks: Recall vs. Precision, and Beyond

**Daniel Berry, Jane Cleland-Huang,
Alessio Ferrari, Walid Maalej,
John Mylopoulos, Didar Zowghi**

Vocabulary

CBS = Computer-Based System

SE = Software Engineering

RE = Requirements Engineering

RS = Requirements Specification

NL = Natural Language

NLP = Natural Language Processing

IR = Information Retrieval

HD = High Dependability

HT = Hairy Task

NLP for RE?

After Kevin Ryan observed in 1993 that NLP was not likely to ever be powerful enough to do RE, ...

RE researchers began to apply NLP to build tools for a variety of *specific* RE tasks involving NL RSs

NLP for RE!

Since then, NLP has been applied to

- abstraction finding,
- requirements tracing,
- multiple RS consolidation,
- requirement classification,
- app review analysis,
- model synthesis,
- RS ambiguity finding, and its generalization,
- RS defect finding

These and others are collectively NL RE tasks.

Task Vocabulary

A *task* is an instance of one of these or other NL RE tasks.

A task *T* is applied to a collection of documents *D* relevant to one RE effort for the development of a CBS.

A *correct answer* is an instance of what *T* is looking for.

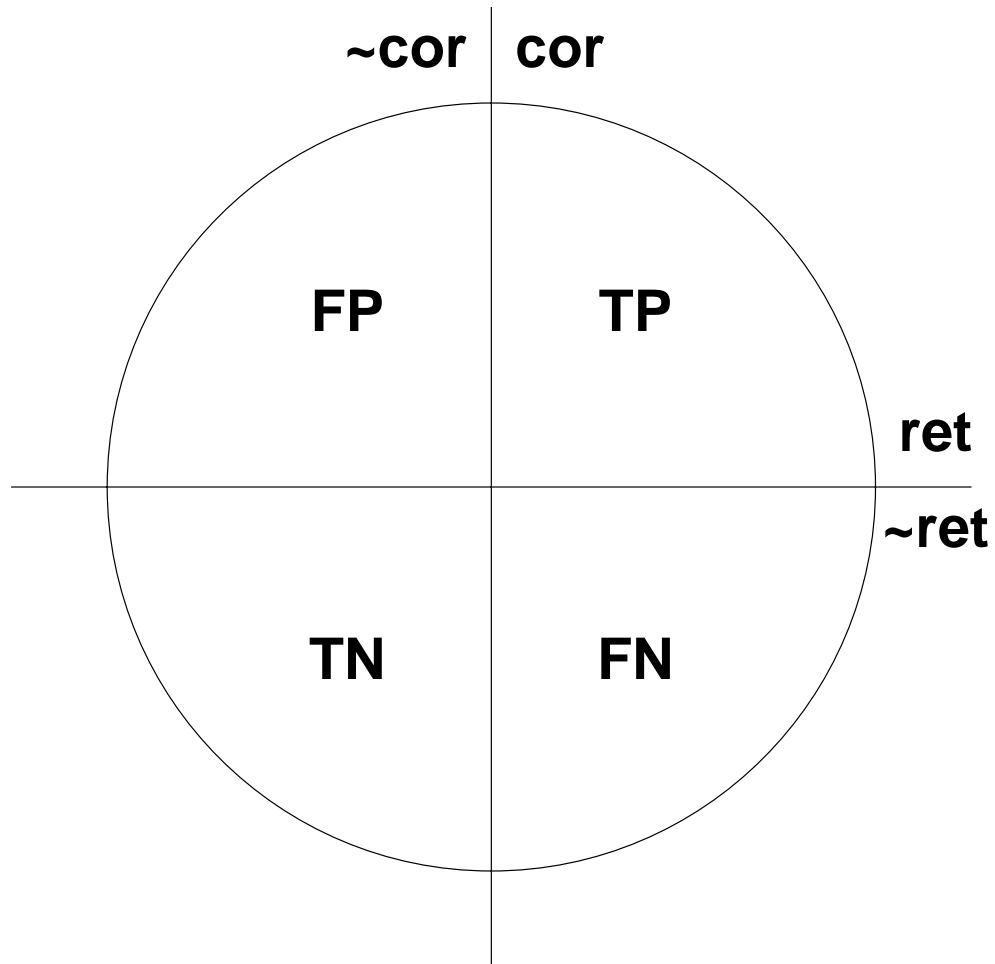
Task Vocabulary, Cont'd

A correct answer is somehow derived from D .

A *tool* for T returns to its users *answers* that it believes to be correct.

The job of a tool for T is to return correct answers and to avoid returning incorrect answers.

Universe of an RE Tool



Adopting IR Methods

RE field has often adopted (and adapted) IR algorithms to develop tools for NL RE tasks.

Quite naturally RE field has adopted also IR's measures:

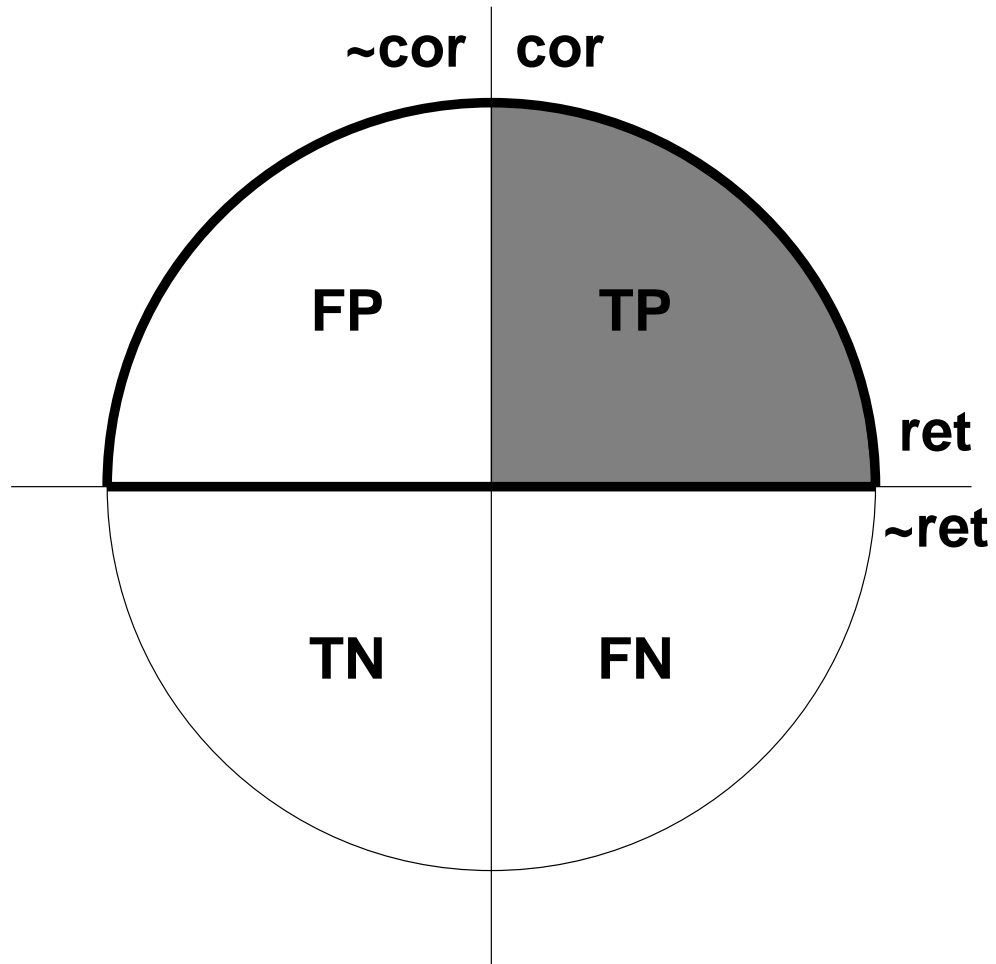
- **precision, P ,**
- **recall, R , and**
- **the F -measure**

Precision

P is the percentage of the tool-returned answers that are correct.

$$\begin{aligned} P &= \frac{| \textit{ret} \cap \textit{cor} |}{| \textit{ret} |} \\ &= \frac{| \textit{TP} |}{| \textit{FP} | + | \textit{TP} |} \end{aligned}$$

Precision

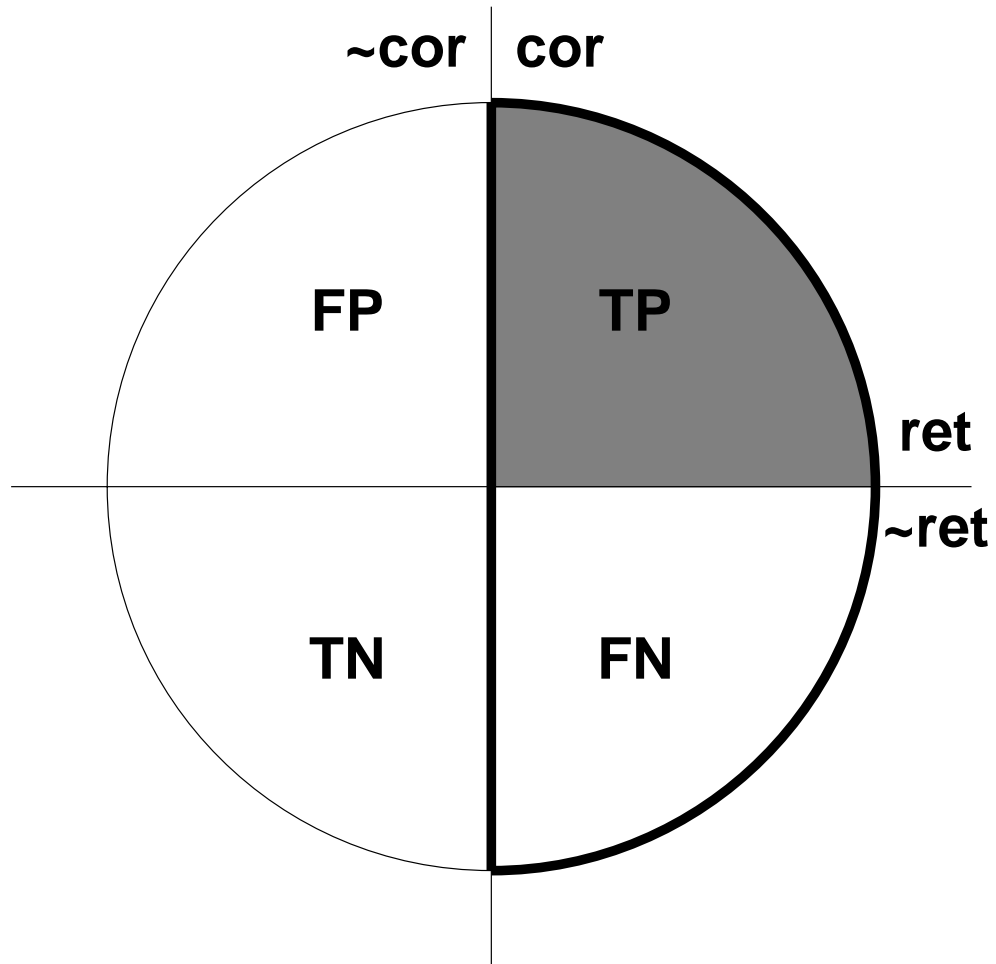


Recall

***R* is the percentage of the correct answers that the tool returns.**

$$\begin{aligned} R &= \frac{| \textit{ret} \cap \textit{cor} |}{| \textit{cor} |} \\ &= \frac{| \textit{TP} |}{| \textit{TP} | + | \textit{FN} |} \end{aligned}$$

Recall



F-Measure

***F*-measure: harmonic mean of *P* and *R*
(harmonic mean is the reciprocal of the
arithmetic mean of the reciprocals)**

Popularly used as a composite measure.

$$F = \frac{1}{\frac{\frac{1}{P} + \frac{1}{R}}{2}} = 2 \cdot \frac{P \cdot R}{P + R}$$

Weighted F -Measure

For situations in which R and P are not equally important, there is a weighted version of the F -measure:

$$F_{\beta} = (1 + \beta^2) \cdot \frac{P \cdot R}{\beta^2 \cdot P + R}$$

Here, β is the ratio by which it is desired to weight R more than P .

Note That

$$F = F_1$$

**As β grows, F_β approaches R
(and P becomes irrelevant).**

If Recall Very Very Important

Now, as $w \rightarrow \infty$,

$$\begin{aligned} F_w &\approx w^2 \cdot \frac{P \cdot R}{w^2 \cdot P} \\ &= \frac{w^2 \cdot P \cdot R}{w^2 \cdot P} = R \end{aligned}$$

As the weight of R goes up, the F-measure begins to approximate simply R !

If Precision Very Very Important

Then, as $w \rightarrow 0$,

$$F_w \approx 1 \cdot \frac{P \cdot R}{R}$$

$$= P$$

which is what we expect.

High-Level Objective

High-level objective of this panel is to

**explore the validity of the tacit
assumptions the RE field made ...**

**in simply adopting IR's tool evaluation
methods to ...**

evaluate tools for NL RE tasks.

Detailed Objectives

The detailed objectives of this panel are:

- **to discuss R , P , and other measures that can be used to evaluate tools for NL RE tasks,**
- **to show how to gather data to decide the measures to evaluate a tool for an NL RE task in a variety of contexts, and**
- **to show how these data can be used in a variety of specific contexts.**

To the Practitioner Here

We believe that you are *compelled* to do many of these kinds of *tedious* tasks in your work.

This panel will help you learn how to decide for any such task ...

if it's worth using any offered tool for for the task instead of buckling down and doing the task manually.

It will tell you the *data* you need to *know*, and to *demand* from the tool builder, in order to make the decision *rationally* in your context!

Plan for Panel

The present slides are an overview of the panel's subject.

After this overview, panelists will describe the evaluation of specific tools for specific NL RE tasks in specific contexts.

Plan, Cont'd

We will invite the audience to join in after that.

In any case, if *anything* is not clear, please ask for clarification immediately!

***But*, please no debating during anyone's presentation.**

Let him or her finish the presentation, and *then* you offer your viewpoint.

R vs. *P* Tradeoff

P and *R* can usually be traded off in an IR algorithm:

- increase *R* at the cost of decreasing *P*, or
- increase *P* at the cost of decreasing *R*

Extremes of Tradeoff

Extremes of this tradeoff are:

1. tool returns all possible answers, correct and incorrect: for

$$R = 100\%, P = C,$$

$$\text{where } C = \frac{\# \text{ correctAnswers}}{\# \text{ answers}}$$

= λ , the frequency
of correct answers

2. tool returns only one answer, a correct one: for

$$P = 100\%, R = \varepsilon,$$

$$\text{where } \varepsilon = \frac{1}{\# \text{ correctAnswers}}$$

Extremes are Useless

**Extremes are useless, because in either case,
...**

the entire task must be done manually on the original document in order to find *exactly* the correct answers.

100% Recall Useless?

Returning everything to get 100% recall doesn't save any real work, because we still have to manually search the entire document.

This is why we are wary of claims of 100% recall ... Maybe it's a case of this phenomenon!

What is missing?

Summarization

Summarization

If we can return a subdocument significantly smaller than the original ...

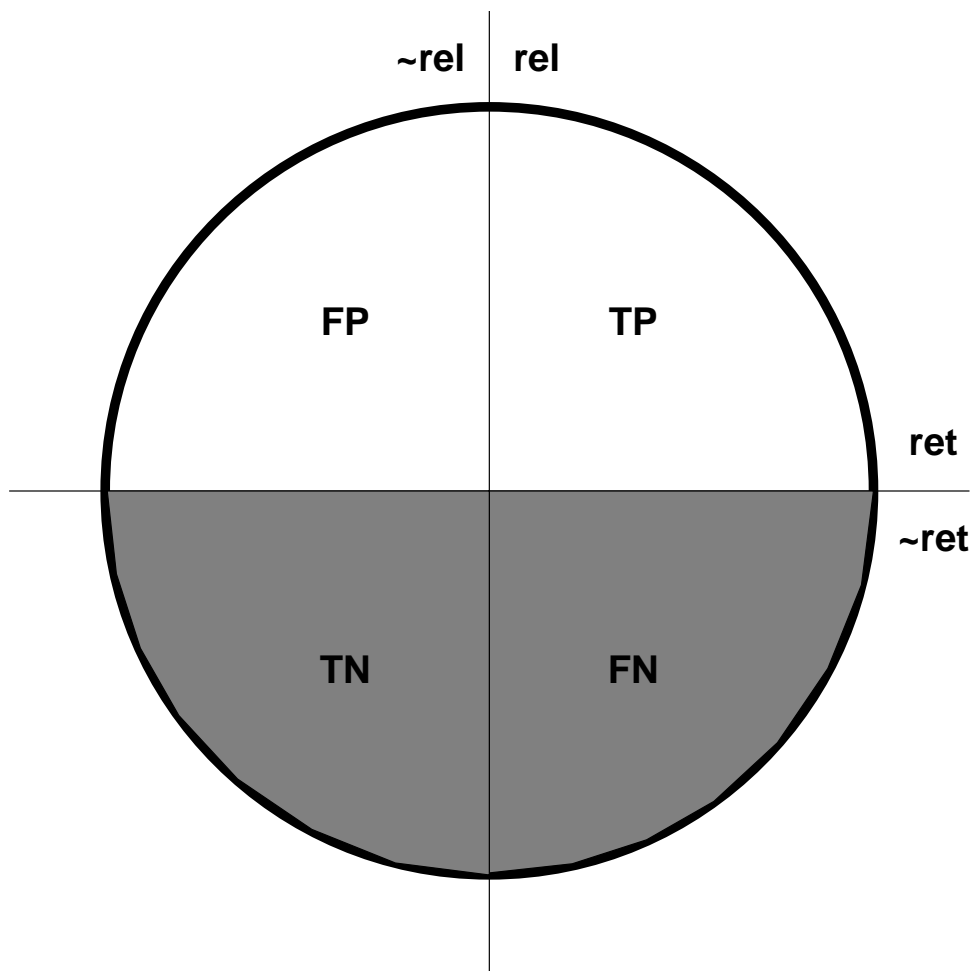
that contains *all* relevant items, ...

then we have saved some real work.

Summarization Measure

Summarization = fraction of the original document that is eliminated from the return

$$\begin{aligned} S &= \frac{|\sim ret|}{|\sim ret \cup ret|} = \frac{|\sim ret|}{|\sim rel \cup rel|} \\ &= \frac{|TN| + |FN|}{|TN| + |FN| + |TP| + |FP|} \end{aligned}$$



How to Use Summarization

We would *love* a tool with 100% recall and 90% summarization.

Then we really do not care about precision.

In Other Words

That is, if we can get rid of 90% of the document with the assurance that ... what is gotten rid of contains *only irrelevant* items and thus ...

what is returned contains *all* the relevant items, ...

then we are *very happy!* 😊

For T=tracing, summarization is not really applicable. However, there is another measure, Selectivity, that IS applicable. See Addendum (3)

Historically, IR Tasks

IR field, e.g., for search engine task, values P higher than R :

Valuing P more than R

Makes sense:

Search for a Portuguese restaurant.

All you need is 1 correct answer:

$$R = \frac{1}{\# \text{ correctAnswers}}$$

But you are very annoyed at having to wade through many FPs to get to the 1 correct answer, i.e.,
with low P

NL RE Task

Very different from IR task:

- task is **hairy**, and
- often **critical to find all correct answers**, for $R = 100\%$, e.g. for a safety- or security-critical CBS.

Hairy Task

On **small scale**, finding a correct answer in a single document, a **hairy NL RE task**, ...

e.g., deciding whether a particular sentence in one RS has a defect, ...

is easy.

Hairy Task, Cont'd

However, in the **context of typical large collection of large NL documents** accompanying the development of a CBS, the **hairy NL RE task, ...**

e.g., finding in all NL RSs for the CBS, all defects, ...

some of which involve multiple sentences in multiple RSs, ...

becomes *unmanageable*.

Hairy Task, Cont'd

It is the problem of **finding *all*** of the ***few*** **matching pairs of needles** distributed throughout **multiple haystack.**

“Hairy Task”?

Theorems, i.e., verification conditions, for proving a program consistent with its formal spec, are not particularly deep, ...

involve high school algebra, ...

but are incredibly messy, even unmanageable, requiring facts from all over the program and the proofs so far ...

and require the help of a theorem proving tool.

We used to call these “hairy theorems”.

“Hairy Task”?, Cont’d

At one place I consulted, its interactive theorem prover was nicknamed “Hairy Reasoner” 😊 (with apologies to the late Harry Reasoner of ABC and CBS News)

Other more conventional words such as “complex” have their own baggage.

Hairiness Needs Tools

The very hairiness of a HT is what motivates us to develop tools to assist in performing the HT, ...

particularly when, e.g. for safety- or security-critical CBS, ...

***all* correct answers, ...**

e.g., ambiguities, defects, or traces ...

***must* be found.**

Hairiness Needs Tools, Cont'd

For such a tool, ...

R is going to be more important than P , and ...

β in F_β will be > 1

Here, divert to talk about how close 100% recall needs to be. See Addendum (1)

What Affects R vs. P Tradeoff?

Three partially competing factors affecting relative importance of R and P are:

- **the value of β as a ratio of two time durations,**
- **the real-life cost of a failure to find a TP, and**
- **the real-life cost of FPs.**

Value of β

**The value of β can be taken as ratio of
the time for a human to find a TP in a
document
over
the time for a human to reject a tool-
presented FP.**

**We will see how to get estimates during gold-
standard construction.**

Replace by Addendum (2)

Some Values of β

The panel paper gives some β values ranging from 1.07 to **73.60** for the tasks:
predicting app ratings, estimating user experiences, & finding feature requests from app reviews;
finding ambiguities; and
finding trace links.

Replace by Addendum (2)

Gold Standard for T

Need a representative same document D for which a group G of humans have done T manually to obtain a list L of correct answers for T on D .

This list L is the gold standard.

L is used to measure R and P for any tool t , by comparing t 's output on D with L .

Replace by Addendum (4)

Gather Data During L 's Construction

During L 's construction, gather following data

- average time for anyone to find any correct answer = β 's numerator,
- average time to decide the correctness of any potential answer = lower upper bound estimate for β 's denominator, independent of any tool's actual value,

Replace by Addendum (4)

During L 's Construction, Con't

- average R of any human in G , relative to final L = estimate for **humanly achievable high recall (HAHR)**.

Replace by Addendum (4)

Real-life cost of not finding a TP

For a safety-critical CBS, this cost can include loss of life.

For a security-critical CBS, this cost can include loss of data.

Real-life cost of FPs

High annoyance with a tool's many FPs can deter the tool's use.

Tool vs. Manual

Should we use a tool for a particular HT T ?

Have to compare tool's R with that of humans manually performing the T on the same documents.

Goal of 100% R ?

For a use of the HT in the development of a safety- or security-critical CBS, we need the tool to achieve R close to 100%.

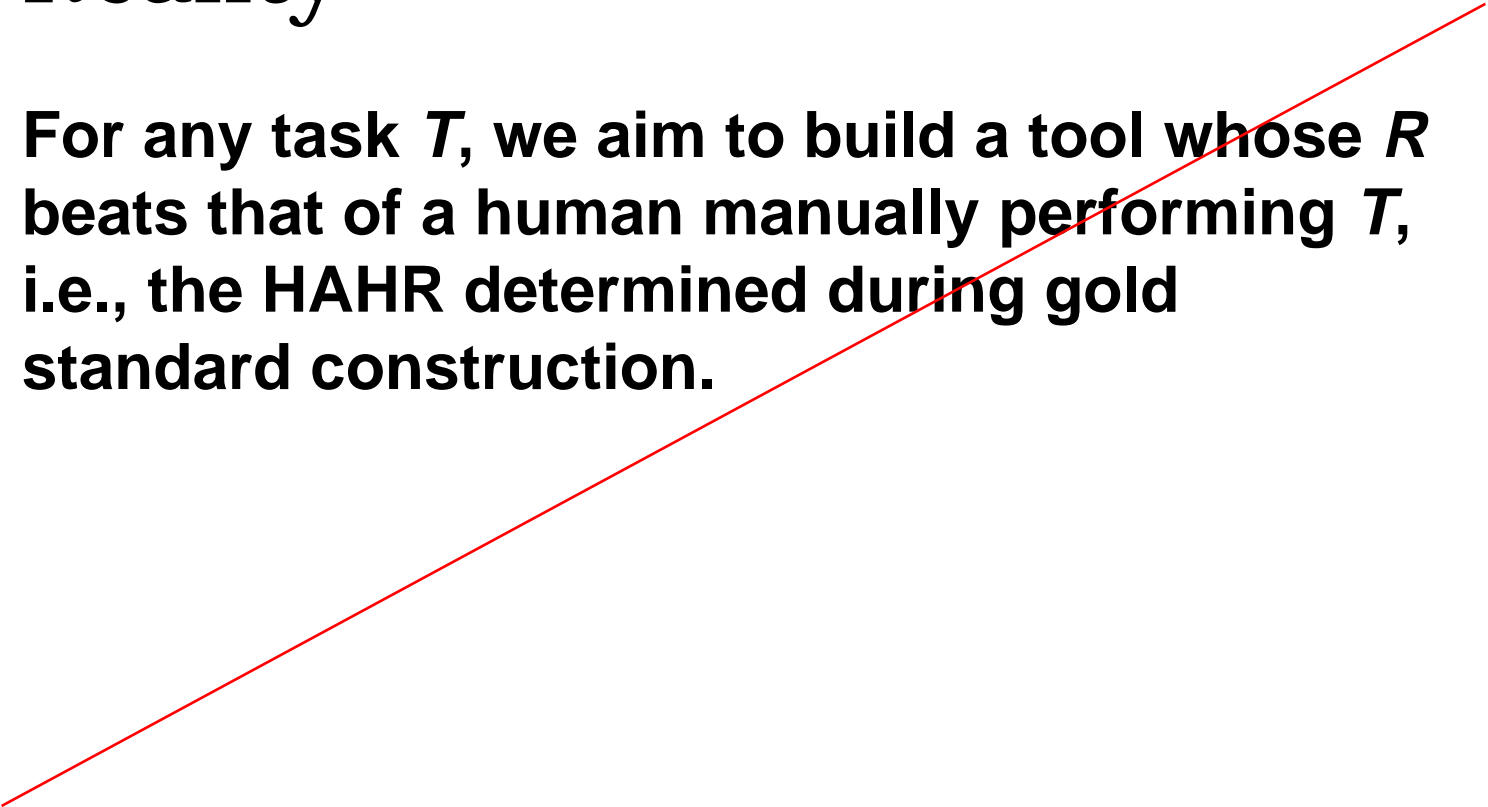
Goal of 100% R , Cont'd

However,

- achieving $R = 100\%$ for T is probably impossible, even for a human!
- there's no way to be sure that a tool or person has achieved $R = 100\%$ because the only way to measure R is to compare the tool or person's output with the set of all correct answers, which is impossible to obtain!

Reality

For any task T , we aim to build a tool whose R beats that of a human manually performing T , i.e., the HAHR determined during gold standard construction.



Summary

To evaluate a tool t for a task T , we need

- to have effective empirical ways to measure tool's and humans' R and P , and times to do T ,
- to take into account the value of β and the real-life costs, and
- to compare tool's R and P and humans' R and P on the same set of documents.

Now Panelists Take Over

The panelists consider the evaluation of tools

...

for a variety of HTs ...

in a variety of contexts.

Evaluating NL RE tools (Mylopoulos)

- Context: We have developed a tool for extracting requirements from regulations, e.g., extract requirements for a meetingscheduler from a privacy law [Zeni-REJ-2015].
- (Dubious) Assumption: Legalese \subseteq NL
- Key considerations in evaluating the tool:
 - ✓ Tool is useful if it reduces manual effort substantially without deterioration in the quality of the answer;
 - ✓ (Hence) Tool must find all/most requirements, otherwise a person has to make a full pass over the regulation to find FNs;
 - ✓ There is no gold standard, even experts make mistakes.

Position statement (Mylopoulos)

- Tools automate NL RE tasks, or assist people in performing NL RE tasks.
- For automation tools, precision is very important in determining the quality of the tool; e.g., for a tool that takes as input a RS and a regulation and tells you whether RS complies with the regulation, a 50% correct response renders the tool useless.
- For assistive tools, reduction of manual effort while maintaining quality is the ultimate evaluation criterion, and then recall is very important.

Example Tool Evaluation

Tracing tool developed and evaluated by
Merten et al [REFSQ16]

$$R = 1.0, P = .02, F_1 = .039, F_2 = .093$$

Do Information Retrieval Algorithms for Automated Traceability Perform Effectively on Issue Tracking System Data?

Thorsten Merten¹(✉), Daniel Krämer¹, Bastian Mager¹, Paul Schell¹,
Simone Bürsner¹, and Barbara Paech²

¹ Department of Computer Science, Bonn-Rhein-Sieg University of Applied Sciences,
Sankt Augustin, Germany

{thorsten.merten,simone.buersner}@h-brs.de,
{daniel.kraemer.2009w,bastian.mager.2010w,
paul.schell.2009w}@informatik.h-brs.de

² Institute of Computer Science, University of Heidelberg, Heidelberg, Germany
paech@informatik.uni-heidelberg.de

Abstract. [Context and motivation] Traces between issues in issue tracking systems connect bug reports to software features, they connect competing implementation ideas for a software feature or they identify duplicate issues. However, the trace quality is usually very low. To improve the trace quality between requirements, features, and bugs, information retrieval algorithms for automated trace retrieval can be employed. Prevailing research focusses on structured and well-formed documents, such as natural language requirement descriptions. In contrast, the information in issue tracking systems is often poorly structured and contains digressing discussions or noise, such as code snippets, stack traces, and links. Since noise has a negative impact on algorithms for automated trace retrieval, this paper asks: [Question/Problem] Do information retrieval algorithms for automated traceability perform effectively on issue tracking system data? [Results] This paper presents an extensive evaluation of the performance of five information retrieval algorithms. Furthermore, it investigates different preprocessing stages (e.g. stemming or differentiating code snippets from natural language) and evaluates how to take advantage of an issue's structure (e.g. title, description, and comments) to improve the results. The results show that algorithms perform poorly without considering the nature of issue tracking data, but can be improved by project-specific preprocessing and term weighting. [Contribution] Our results show how automated trace retrieval on issue tracking system data can be improved. Our manually created gold standard and an open-source implementation based on the OpenTrace platform can be used by other researchers to further pursue this topic.


Keywords: Issue tracking systems · Empirical study · Traceability · Open-source

which algorithm performs best with a certain data set without experimenting, although BM25 is often used as a baseline to evaluate the performance of new algorithms for classic IR applications such as search engines [2, p. 107].

2.2 Measuring IR Algorithm Performance for Trace Retrieval

IR algorithms for trace retrieval are typically evaluated using the recall (R) and precision (P) metrics with respect to a reference trace matrix. R measures the retrieved relevant links and P the correctly retrieved links:

$$R = \frac{CorrectLinks \cap RetrievedLinks}{CorrectLinks}, \quad P = \frac{CorrectLinks \cap RetrievedLinks}{RetrievedLinks} \quad (2)$$

Since P and R are contradicting metrics (R can be maximized by retrieving all links, which results in low precision; P can be maximised by retrieving only one correct link, which results in low recall) the F_β -Measure as their harmonic mean is often employed in the area of traceability. In our experiments, we computed results for the F_1 measure, which balances P and R , as well as F_2 , which emphasizes recall: 

$$F_\beta = \frac{(1 + \beta^2) \times Precision \times Recall}{(\beta^2 \times Precision) + Recall} \quad (3)$$

Huffman Hayes et al. [13] define *acceptable*, *good* and *excellent* P and R ranges. Table 3 extends their definition with according F_1 and F_2 ranges. The results section refers to these ranges.

2.3 Issue Tracking System Data Background

At some point in the software engineering (SE) life cycle, requirements are communicated to multiple roles, like project managers, software developers and, testers. Many software projects utilize an ITS to support this communication and to keep track of the corresponding tasks and changes [28]. Hence, requirement descriptions, development tasks, bug fixing, or refactoring tasks are collected in ITSs. This implies that the data in such systems is often uncategorized and comprises manifold topics [19].

The NL data in a single issue is usually divided in at least two fields: A title (or summary) and a description. Additionally, almost every ITS supports commenting on an issue. Title, description, and comments will be referred to as *ITS data fields* in the remainder of this paper. Issues usually describe new software requirements, bugs, or other development or test related tasks. Figure 1³ shows an excerpt of the title and description data fields of two issues, that both request a new software feature for the Redmine project. It can be inferred from the text, that both issues refer to the same feature and give different solution proposals.

³ Figure 1 intentionally omits other meta-data such as authoring information, date- and time-stamps, or the issue status, since it is not relevant for the remainder of this paper.

or comments represent only hasty notes meant for a developer – often without forming a whole sentence. In contrast, RAs typically do not contain noise and NL is expected to be correct, consistent, and precise. Furthermore, structured RAs are subject to a specific quality assurance⁵ and thus their structure and NL is much better than ITS data.

Since IR algorithms compute the text similarity between two documents, spelling errors and hastily written notes that leave out information, have a negative impact on the performance. In addition, the performance is influenced by source code which often contains the same terms repeatedly. Finally, stack traces often contain a considerable amount of the same terms (e.g. Java package names). Therefore, an algorithm might compute a high similarity between two issues that refer to different topics if they both contain a stack trace.

3 Related Work

Borg et al. conducted a systematic mapping of trace retrieval approaches [3]. Their paper shows that much work has been done in trace retrieval between RA, but only few studies use ITS data. Only one of the reviewed approaches in [3] uses the BM25 algorithm, but VSM and LSA are used extensively. This paper fills both gaps by comparing VSM, LSA, and three variants of BM25 on unstructured ITS data. [3] also reports on preprocessing methods saying that stop word removal and stemming are most often used. Our study focusses on the influence of ITS-specific preprocessing and ITS data field-specific term weighting beyond removing stop words and stemming. Gotel et al. [10] summarize the results of many approaches for automated trace retrieval in their roadmap paper. They recognize that results vary largely: “[some] methods retrieved almost all of the true links (in the 90 % range for recall) and yet also retrieved many false positives (with precision in the low 10–20 % range, with occasional exceptions).” We expect that the results in this paper will be worse, as we investigate in issues and not in structured RAs.

Due to space limitations, we cannot report on related work extensively and refer the reader to [3, 10] for details. The experiments presented in this paper are restricted to standard IR text similarity methods. In the following, extended approaches are summarized that could also be applied to ITS data and/or combined with the contribution in this paper: Nguyen et al. [21] combine multiple properties, like the connection to a version control system to relate issues. Gervasi and Zowghi [8] use additional methods beyond text similarity with requirements and identify another affinity measure. Guo et al. [11] use an expert system to calculate traces automatically. The approach is very promising, but is not fully automated. Sultanov and Hayes [29] use reinforcement learning and improve the results compared to VSM. Niu and Mahmoud [22] use clustering to group links in high-quality and low-quality clusters respectively to improve accuracy. The low-quality clusters are filtered out. Comparing multiple techniques for trace retrieval, Oliveto et al. [23] found that no technique outperformed the others.

⁵ Dag and Gervasi [20] surveyed automated approaches to improve the NL quality.

Table 1. Project characteristics

	c:geo	Lighttpd	Radiant	Redmine
Software Type	Android app	HTTP server	content mgmt. system	ITS
Audience	consumer	technician	consumer / developer	hoster / developer
Main programming lang.	Java	C	Ruby	Ruby
ITS	GitHub	Redmine	GitHub	Redmine
ITS Usage	ad-hoc	structured	ad-hoc	very structured
ITS size (in # of issues)	~ 3850	~ 2900	~ 320	~ 19,000
Open issues	~ 450	~ 500	~ 50	~ 4500
Closed issues	~ 3400	~ 2400	~ 270	~ 14,500
Sample size	100 \approx 3%	100 \approx 3%	100 \approx 30%	100 < 1%
Sampled issues with link	~ 50%	~ 20%	~ 12%	~ 70%
Issues labeled explicitly as Feature or Bug in sample	25F/26B	30F/70B	0F/0B	31F/61B
Project size (in LOC)	~ 130,000	~ 41,000	~ 33,000	~ 150,000

Researched Projects and Project Selection. The data used for the experiments in this paper was taken from the following four projects:

- *c:geo*, an Android application to play a real world treasure hunting game.
- *Lighttpd*, a lightweight web server application.
- *Radiant*, a modular content management system.
- *Redmine*, an ITS.

The projects show different characteristics with respect to the software type, intended audience, programming languages, and ITS. Details of these characteristics are shown in Table 1. *c:geo* and *Radiant* use the GitHub ITS and *Redmine* and *Lighttpd* the Redmine ITS. Therefore, the issues of the first two projects are categorized by tagging, whereas every issue of the other projects is marked as a feature or a bug (see Table 1). *c:geo* was chosen because it is an Android application and the ITS contains more consumer requests than the other projects. *Lighttpd* was chosen because it is a lightweight web server and the ITS contains more code snippets and noise than the other projects. *Radiant* was chosen because its issues are not categorized as feature or bug at all and it contains fewer issues than the other projects. Finally, *Redmine* was chosen because it is a very mature project and ITS usage is very structured compared to the other projects. Some of the researchers were already familiar with these projects, since we reported on ITS NL contents earlier [19].

Gold Standard Trace Matrices. The first, third, and fourth author created the gold standard trace matrices (GSTM). For this task, the title, description, and comments of each issue was manually compared to every other issue. Since 100 issues per project were extracted, this implies $\frac{100 * 100}{2} - 50 = 4950$ manual comparisons. To have semantically similar gold standards for each project, a code of conduct was developed that prescribed e.g. when a generic trace should be created (as defined in Sect. 2.3) or when an issue should be treated as duplicate (the description of both issues describes exactly the same bug or requirement).

and
on to
next
page

Table 2. Extracted traces vs. gold standard

# of relations	Projects			
	c:geo	Lighttpd	Radiant	Redmine
DTM generic	59	11	8	60
GSTM generic	102	18	55	94
GSTM duplicates	2	3	-	5
Overlapping	30	9	5	45

Table 3. Evaluation measures adapted from [13]

Acceptable	Good	Excellent
$0.6 \leq r < 0.7$	$0.7 \leq r < 0.8$	$r \geq 0.8$
$0.2 \leq p < 0.3$	$0.3 \leq p < 0.4$	$p \geq 0.4$
$0.2 \leq F_1 < 0.42$	$0.42 \leq F_1 < 0.53$	$F_1 \geq 0.53$
$0.43 \leq F_2 < 0.55$	$0.55 \leq F_2 < 0.66$	$F_2 \geq 0.66$

Since concentration quickly declines in such monotonous tasks, the comparisons were aided by a tool especially created for this purpose. It supports defining related and unrelated issues by simple keyboard shortcuts as well as saving and resuming the work. At large, a GSTM for one project was created in two and a half business days.



In general the GSTMs contain more traces than the DTMs (see Table 2). A manual analysis revealed that developers often missed (or simply did not want to create) traces or created relations between issues that are actually not related. The following examples indicate why GSTMs and DTMs differ: (1) Eight out of the 100 issues in the c:geo dataset were created automatically by a bot that manages translations for internationalization. Although these issues are related, they were not automatically marked as related. There is also a comment on how internationalization should be handled in issue (#4950). (2) Some traces in the Redmine based projects do not follow the correct syntax and are therefore missed by a parser. (3) Links are often vague and unconfirmed in developer traces. E.g. c:geo #5063 says that the issue “could be related to #4978 [...] but I couldn’t find a clear scenario to reproduce this”. We also could not find evidence to mark these issues as related in the gold standard but a link was already placed by the developers. (4) Issue #5035 in c:geo contains a reference to #3550 to say that a bug occurred before the other bug was reported (the trace semantics in this case is: “occurred likely before”). There is, however, no semantics relation between the bugs, therefore we did not mark these issues as related in the gold standard. (5) The Radiant project simply did not employ many manual traces.

5.2 Tools

The experiments are implemented using the OpenTrace (OT) [1] framework. OT retrieves traces between NL RAs and includes means to evaluate results with respect to a reference matrix.

OT utilizes IR implementations from Apache Lucene⁷ and it is implemented as an extension to the General Architecture for Text Engineering (GATE) framework [6]. GATE’s features are used for basic text processing and pre-processing functionality in OT, e.g. to split text into tokens or for stemming. To make both frameworks deal with ITS data, some changes and enhancements were made to

⁷ <https://lucene.apache.org>.

Table 4. Data fields weights (l), algorithms and preprocessing settings (r)

Weight				Rationale / Hypothesis	Algorithm	Settings
Title	Description	Comments	Code			
1	1	1	1	Unaltered algorithm	BM25	Pure, +, L
1	1	1	0	– without considering code	VSM	TF-IDF
1	1	0	0	– also without comments	LSI	<i>cos</i> measure
2	1	1	1	Title more important	Preprocessing	
2	1	1	0	– without considering code	Settings	
1	2	1	1	Description more important	<i>Standard</i>	
1	1	1	2	Code more important	Stemming	on/off
8	4	2	1	Most important information first	Stop Word Removal	on/off
4	2	1	0	– without considering code	<i>ITS-specific</i>	
2	1	0	0	– also without comments	Noise Removal	on/off
					Code Extraction	on/off

OT: (1) refactoring to make it compatible with the current GATE version (8.1), (2) enhancement to make it process ITS data fields with different term weights, and (3) development of a framework to configure OT automatically and to run experiments for multiple configurations. The changed source code is publicly available for download⁸.

5.3 Algorithms and Settings

For the experiment, multiple term weighting schemes for the ITS data fields and different preprocessing methods are combined with the IR algorithms VSM, LSI, BM25, BM25+, BM25L. Beside stop word removal and stemming, which we will refer to as *standard preprocessing*, we employ *ITS-specific preprocessing*. For the ITS-specific preprocessing, noise (as defined in Sect. 2) was removed and the regions marked as code were extracted and separated from the NL. Therefore, term weights can be applied to each ITS data field and the code. Table 4 gives an overview of all preprocessing methods (right) and term weights as well as rationales for the chosen weighting schemes (left).

6 Results

We compute $trace_t$ with different thresholds t in order to maximize precision, recall, F_1 and F_2 measure. Results are presented as F_2 and F_1 measure in general. However, maximising recall is often desirable in practice, because it is simpler to remove wrong links manually than to find correct links manually. Therefore, R with corresponding precision is also discussed in many cases.

As stated in Sect. 5.1, a comparison with the GSTM results in more authentic and accurate measurements than a comparison with the DTM. It also yields better results: F_1 and F_2 both increase about 9 % in average computed on the

⁸ <http://www2.inf.h-brs.de/~tmerte2m> – In addition to the source code, gold standards, extracted issues, and experiment results are also available for download.

unprocessed data sets. A manual inspection revealed that this increase materializes due to the flaws in the DTM, especially because of missing traces. Therefore, the results in this paper are reported in comparison with the GSTM.

6.1 IR Algorithm Performance on ITS Data

Figure 2 shows an evaluation of all algorithms with respect to the GSTMs for all projects with and without *standard preprocessing*. The differences per project are significant with 30 % for F_1 and 27 % for F_2 . It can be seen that standard preprocessing does not have a clear positive impact on the results. Although, if only slightly, a negative impact on some of the project/algorithm combinations is noticeable. On a side note, our experiment supports the claim of [12], that removing stop-words is not always beneficial on ITS data: We experimented with different stop word lists and found that a small list that essentially removes only pronouns works best.

In terms of algorithms, to our surprise, no variant of BM25 competed for the best results. The best F_2 measures of all BM25 variants varied from 0.09 to 0.19 over all projects, independently of standard preprocessing. When maximizing R to 1, P does not cross a 2 % barrier for any algorithm. Even for $R \geq 0.9$, P is still < 0.05 . All in all, the results are not good according to Table 3, independently of standard preprocessing, and they cannot compete with related work on structured RAs.

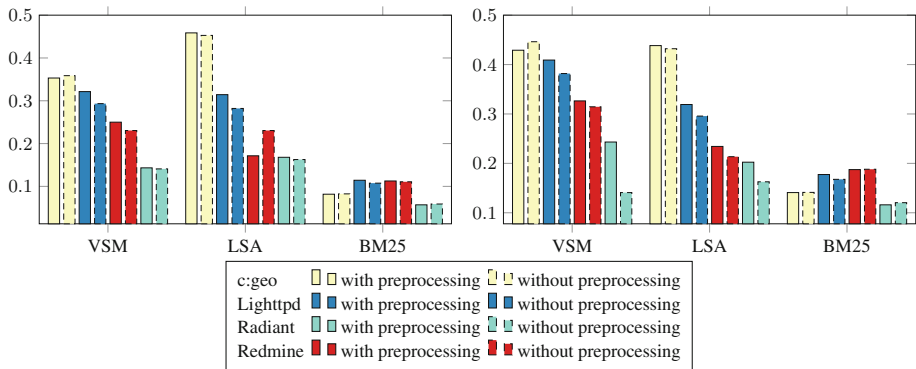


Fig. 2. Best F_1 (left) and F_2 (right) scores for every algorithm

Although results decrease slightly in a few cases, the negative impact is negligible. Therefore, the remaining measurements are reported with the standard preprocessing techniques enabled⁹.

⁹ In addition, removing stop words and stemming is considered IR best practices, e.g. [2, 17].

Mining Some Estimates

In their description of their gold standard construction process, I was able to mine some estimates, which were validated by e-mail with Merten:

- Time to find a correct link: 17.84 person-minutes
- Time to consider a potential link: 14.54 person-seconds (independent of any tool)

$$\therefore \beta = 73.6 \text{ and } F_{\beta} = .990$$

Replace by Addendum (5)

Mining More Estimates

Based on facts:

- **There was an upfront discussion, leading to consensus, on criteria for TP links.**
- **About 5% of the considered links needed a discussion during the construction.**

I estimate that HAHR is [95% – 90%]

Verdict on the Tool

Is the tool worth using?

It depends! 😊

Certainly, the tool's *R* beats HAHR!

So, it gets down to whether the tool makes the remaining manual job easier, i.e., smaller or faster.

Verdict, Cont'd

The problem with such R and P is that they are close to those of the useless' tool that returns every answer.

If the tool's answers summarize the original documents, i.e.,

the tool's answers contain every correct link, *and* are smaller than the original documents, so that the human has less work to do ...

If ..., Cont'd

Or the tool presents the information relevant to vetting a link in a form that makes the vetting time less than 14.54 seconds, ...

then the tool is worth using

else the tool is not worth using

when you need 100% *R*.

On the automatic classification of app reviews

Walid Maalej¹ · Zijad Kurtanović¹ · Hadeer Nabil² · Christoph Stanik¹

Walid: please look at my highlightings and added stickies. Look in particular at the sticky I attached to highlighted text on page 321 (11 or 21).

Dan

Received: 14 November 2015 / Accepted: 26 April 2016 / Published online: 14 May 2016
© Springer-Verlag London 2016

Abstract App stores like Google Play and Apple AppStore have over 3 million apps covering nearly every kind of software and service. Billions of users regularly download, use, and review these apps. Recent studies have shown that reviews written by the users represent a rich source of information for the app vendors and the developers, as they include information about bugs, ideas for new features, or documentation of released features. The majority of the reviews, however, is rather non-informative just praising the app and repeating to the star ratings in words. This paper introduces several probabilistic techniques to classify app reviews into four types: bug reports, feature requests, user experiences, and text ratings. For this, we use review metadata such as the star rating and the tense, as well as, text classification, natural language processing, and sentiment analysis techniques. We conducted a series of experiments to compare the accuracy of the techniques and compared them with simple string matching. We found that metadata alone results in a poor classification accuracy. When combined with simple text classification and natural language preprocessing of the text—particularly with bigrams and lemmatization—the classification precision for all review types got up to 88–92 % and the recall up to 90–99 %. Multiple binary classifiers outperformed single multiclass classifiers. Our results inspired the design of a review analytics tool, which should help app vendors and developers deal with the large amount of reviews, filter critical reviews, and assign them to the appropriate

stakeholders. We describe the tool main features and summarize nine interviews with practitioners on how review analytics tools including ours could be used in practice.

Keywords User feedback · Review analytics · Software analytics · Machine learning · Natural language processing · Data-driven requirements engineering

1 Introduction

Nowadays it is hard to imagine a business or a service that does not have any app support. In July 2014, leading app stores such as Google Play, Apple AppStore, and Windows Phone Store had over 3 million apps.¹ The app download numbers are astronomic with hundreds of billions of downloads over the last 5 years [9]. Smartphone, tablet, and more recently also desktop users can search the store for the apps, download, and install them with a few clicks. Users can also review the app by giving a star rating and a text feedback.

Studies highlighted the importance of the reviews for the app success [22]. Apps with better reviews get a better ranking in the store and with it a better visibility and higher sales and download numbers [6]. The reviews seem to help users navigate the jungle of apps and decide which one to use. Using free text and star rating, the users are able to express their satisfaction, dissatisfaction or ask for missing features. Moreover, recent research has pointed the potential importance of the reviews for the app developers and vendors as well. A significant amount of the reviews

✉ Walid Maalej
maalej@informatik.uni-hamburg.de

¹ Department of Informatics, University of Hamburg, Hamburg, Germany

² German University of Cairo, Cairo, Egypt

¹ <http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/>.

Table 2 Overview of the evaluation data

App(s)	Category	Platform	#Reviews	Sample
1100 apps	All iOS	Apple	1,126,453	1000
Dropbox	Productivity	Apple	2009	400
Evernote	Productivity	Apple	8878	400
TripAdvisor	Travel	Apple	3165	400
80 apps	Top four	Google	146,057	1000
PicsArt	Photography	Google	4438	400
Pinterest	Social	Google	4486	400
Whatsapp	Communication	Google	7696	400
Total			1,303,182	4400

From the collected data, we randomly sampled a subset for the manual labeling as shown in Table 2. We selected 1000 random reviews from the Apple store data and 1000 from the Google store data. To ensure that enough reviews with 1, 2, 3, 4, and 5 stars are sampled, we split the two 1000-review samples into 5 corresponding subsamples each of size 200. Moreover, we selected 3 random Android apps and 3 iOS apps from the top 100 and fetched their reviews between 2012 and 2014. From all reviews of each app, we randomly sampled 400. This led to additional 1200 iOS and 1200 Android app-specific reviews. In total, we had 4400 reviews in our sample.

For the truth set creation, we conducted a *peer, manual content analysis* for all the 4400 reviews. Every review in the sample was assigned randomly to 2 coders from a total of 10 people. The coders were computer science master students, who were paid for this task. Every coder read each review carefully and indicated its types: bug report, feature request, user experience, or rating. We briefed the coders in a meeting, introduced the task, the review types, and discussed several examples. We also developed a coding guide, which describes the coding task, defines precisely what each type is, and lists examples to reduce disagreements and increase the quality of the manual labeling. Finally, the coders were able to use a coding tool (shown on Fig. 1) that helps to concentrate on one review at once and to reduce coding errors. If both coders agreed on a review type, we used that label in our golden standard. A third coder checked each label and solved the disagreements for a review type by either accepting the proposed label for this type or rejecting it. This ensured that the golden set contained only peer-agreed labels.

In the third phase, we used the manually labeled reviews to train and to test the classifiers. A summary of the experiment data is shown in Table 3. We only used reviews, for which both coders *agreed* that they are of a certain type or not. This helped that a review in the corresponding evaluation sample (e.g., bug reports) is labeled correctly. Otherwise training and testing the classifiers on

unclear data will lead to unreliable results. We evaluated the different techniques introduced in Sect. 2, while varying the classification features and the machine learning algorithms.

We evaluated the classification accuracy using the standard metrics precision and recall. Precision_{*i*} is the fraction of reviews that are classified correctly to belong to type *i*. Recall_{*i*} is the fraction of reviews of type *i* which are classified correctly. They were calculated as follows:

$$\text{Precision}_i = \frac{TP_i}{TP_i + FP_i} \quad \text{Recall}_i = \frac{TP_i}{TP_i + FN_i} \quad (1)$$

TP_i is the number of reviews that are classified as type *i* and actually are of type *i*. FP_i is the number of reviews that are classified as type *i* but actually belong to another type *j* where $j \neq i$. FN_i is the number of reviews that are classified to other type *j* where $j \neq i$ but actually belong to type *i*. We also calculated the *F*-measure (*F1*), which is the harmonic mean of precision and recall providing a single accuracy measure. We randomly split the truth set at a ratio of 70:30. That is, we randomly used 70 % of the data for the training set and 30 % for the test set. Based on the size of our truth set, we felt this ratio is a good trade-off for having large-enough training and test sets. Moreover, we experimented with other ratios and with the cross-validation method. We also calculated how informative the classification features are and ran paired *t* tests to check whether the differences of *F1*-scores are statistically significant.

The results reported in Sect. 4 are obtained using the Monte Carlo cross-validation [38] method with 10 runs and random 70:30 split ratio. That is, for each run, 70 % of the truth set (e.g., for true positive bug reports) is randomly selected and used as a training set and the remaining 30 % is used as a test set. Additional experiments data, scripts, and results are available on the project Web site: <http://mast.informatik.uni-hamburg.de/app-review-analysis/>.

4 Research results

We report on the results of our experiments and compare the accuracy (i.e., precision, recall, and *F*-measures) as well as the performance of the various techniques.

4.1 Classification techniques

Table 4 summarizes the results of the classification techniques using Naive Bayes classifier on the whole data of the truth set (from the Apple AppStore and the Google Play Store). The results in Table 4 indicate the mean values obtained by the cross-validation for each single combination of classification techniques and a review type. The

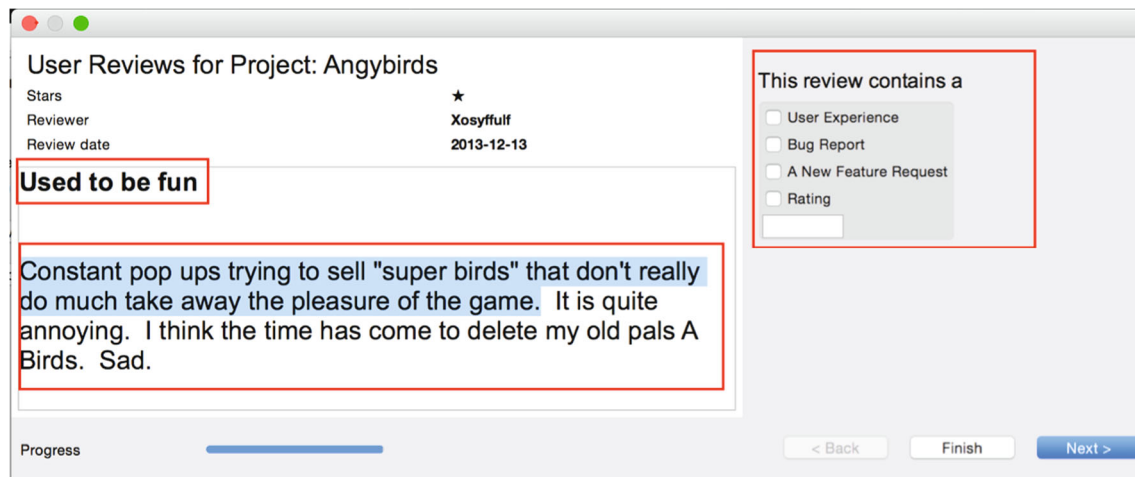


Fig. 1 Tool for manual labeling of the reviews

Table 3 Number of manually analyzed and labeled reviews

Sample	Manually analyzed	Bug reports	Feature requests	User experiences	Ratings
Random apps Apple	1000	109	83	370	856
Selected apps Apple	1200	192	63	274	373
Random apps Google	1000	27	135	16	569
Selected apps Google	1200	50	18	77	923
Total	4400	378	299	737	2721

numbers in bold represent the highest scores for each column, which means the highest accuracy metric (precision, recall, and *F*-measure) for each classifier.

Table 5 shows the *p* values of paired *t* tests on whether the differences between the mean *F1*-scores of the baseline classifier and the various classification techniques are statistically significant. For Example: If one classifier result is 80 % for a specific combination of techniques and another result is 81 % for another combination, those two results could be statistically different or it could be by chance. If the *p* value calculated by the paired *t* test is very small, this means that the difference between the two values is statistically significant. We used Holm’s step-down method [16] to control the family-wise error rate.

Overall, the precisions and recalls of all probabilistic techniques were clearly higher than 50 % except for three cases: the precision and recall of feature request classifiers based on rating only as well as the recall of the same technique (rating only) to predict ratings. Almost all probabilistic approaches outperformed the basic classifiers that use string matching with at least 10 % higher precisions and recalls.

The combination of text classifiers, metadata, NLP, and the sentiments extraction generally resulted in high precision and recall values (in most cases above 70 %). However, the combination of the techniques did not always rank

best. Classifiers only using metadata generally had a rather low precision but a surprisingly high recall except for predicting ratings where we observed the opposite.

Concerning NLP techniques, there was no clear trend like “more language processing leads to better results.” Overall, removing stopwords significantly increased the precision to predict bug reports, feature request, and user experience, while it decreased the precision for ratings. We observed the same when adding lemmatization. On the other hand, combining stop word removal and lemmatization did not had any significant effect on precision and recall.

We did not observe any significant difference between using one or two sentiment scores.

4.2 Review types

We achieved the highest precision for predicting user experience and ratings (92 %), the highest recall, and *F*-measure for user experience (respectively, 99 and 92 %).

For bug reports we found that the highest precision (89 %) was achieved with the bag of words, rating, and one sentiment, while the highest recall (98 %) with using bigrams, rating, and one score sentiment. **For predicting bug reports the recall might be more important than precision. Bug reports are critical reviews, and app vendors would probably need to make sure that a review analytics**

Table 4 Accuracy of the classification techniques using Naive Bayes on app reviews from Apple and Google stores (mean values of the 10 runs, random 70:30 splits for training:evaluation sets)

Classification techniques	10.00 <i>Hairy</i>			9.09 <i>Hairy</i>			2.71 <i>So-So</i>			1.07 <i>Not Hairy</i>		
	Bug reports			Feature requests			User experiences			Ratings		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Basic (string matching)	0.58	0.24	0.33	0.39	0.55	0.46	0.27	0.12	0.17	0.74	0.56	0.64
<i>Document classification (&NLP)</i>												
Bag of words (BOW)	0.79	0.65	0.71	0.76	0.54	0.63	0.82	0.59	0.68	0.67	0.85	0.75
FR Bigram	0.68	0.98	0.80	0.68	0.97	0.80	0.70	0.99	0.82	0.91	0.62	0.73
BOW + bigram	0.85	0.90	0.87	0.86	0.85	0.85	0.87	0.91	0.89	0.85	0.89	0.87
BOW + lemmatization	0.88	0.74	0.80	0.86	0.65	0.74	0.90	0.67	0.77	0.73	0.91	0.81
BOW – stopwords	0.86	0.69	0.76	0.86	0.65	0.74	0.91	0.67	0.77	0.74	0.91	0.81
BOW + lemmatization – stopwords	0.85	0.71	0.77	0.87	0.67	0.76	0.91	0.67	0.77	0.75	0.90	0.82
Rat BOW + bigrams – stopwords + lemmatization	0.85	0.91	0.88	0.86	0.83	0.85	0.89	0.94	0.91	0.85	0.90	0.87
<i>Metadata</i>												
Rating	0.64	0.82	0.72	0.31	0.35	0.31	0.74	0.89	0.81	0.72	0.34	0.46
Rating + length	0.76	0.75	0.75	0.68	0.67	0.67	0.72	0.82	0.77	0.70	0.68	0.69
Rating + length + tense	0.74	0.73	0.74	0.64	0.71	0.67	0.74	0.80	0.77	0.70	0.68	0.69
Rating + length + tense + 1× sentiment	0.69	0.76	0.72	0.66	0.66	0.66	0.71	0.85	0.77	0.71	0.66	0.68
Rating + length + tense + 2× sentiments	0.66	0.78	0.71	0.65	0.72	0.68	0.67	0.88	0.76	0.69	0.67	0.68
<i>Combined (text and metadata)</i>												
BOW + rating + lemmatize	0.85	0.73	0.78	0.89	0.64	0.74	0.90	0.67	0.77	0.73	0.89	0.80
BOW + rating + 1× sentiment	0.89	0.72	0.79	0.89	0.60	0.71	0.92	0.73	0.81	0.75	0.93	0.83
BOW + rating + tense + 1 sentiment	0.87	0.71	0.78	0.87	0.60	0.70	0.92	0.69	0.79	0.74	0.90	0.81
UE BR Bigram + rating + 1× sentiment	0.73	0.98	0.83	0.71	0.96	0.81	0.75	0.99	0.85	0.92	0.69	0.79
Bigram – stopwords + lemmatization + rating + tense + 2× sentiment	0.72	0.97	0.82	0.70	0.94	0.80	0.75	0.98	0.85	0.92	0.72	0.81
BOW + bigram + tense + 1× sentiment	0.87	0.88	0.87	0.85	0.83	0.83	0.88	0.94	0.91	0.83	0.87	0.85
BOW + lemmatize + bigram + rating + tense	0.88	0.88	0.88	0.87	0.84	0.85	0.89	0.94	0.92	0.84	0.90	0.87
BOW – stopwords + bigram + rating + tense + 1× sentiment	0.88	0.89	0.88	0.86	0.84	0.85	0.87	0.93	0.90	0.83	0.89	0.86
BOW – stopwords + lemmatization + rating + 1× sentiment + tense	0.88	0.71	0.79	0.87	0.64	0.74	0.91	0.72	0.80	0.73	0.90	0.80
BOW – stopwords + lemmatization + rating + 2× sentiments + tense	0.87	0.71	0.78	0.86	0.68	0.76	0.91	0.73	0.81	0.75	0.90	0.82

Bold values represent the highest score for the corresponding accuracy metric per review type

Table 5 Results of the paired *t* test between the different techniques (one in each row) and the baseline BoW (using Naive Bayes on app reviews from Apple and Google stores)

Classification techniques	Bug reports		Feature requests		User experiences		Ratings	
	<i>F1</i> -score	<i>p</i> value	<i>F1</i> -score	<i>p</i> value	<i>F1</i> -score	<i>p</i> value	<i>F1</i> -score	<i>p</i> value
<i>Document classification (&NLP)</i>								
Bag of words (BOW)	0.71	Baseline	0.63	Baseline	0.68	Baseline	0.75	Baseline
Bigram	0.80	0.043	0.80	2.5e−06	0.82	0.00026	0.73	0.55
BOW + bigram	0.87	6.9e−05	0.85	2.6e−07	0.89	4.7e−06	0.87	2.9e−05
BOW + lemmatization	0.80	0.031	0.74	0.0022	0.77	0.0028	0.81	0.029
BOW − stopwords	0.76	0.09	0.74	0.0023	0.77	0.0017	0.81	0.0019
BOW − stopwords + lemmatization	0.77	0.051	0.76	0.0008	0.77	0.0021	0.82	0.0005
BOW − stopwords + lemmatization + bigram	0.88	6.6e−05	0.85	2.9e−07	0.91	4.3e−08	0.87	0.0009
<i>Metadata</i>								
Rating	0.72	1.0	0.31	0.04	0.81	7.1e−05	0.46	6.9e−06
Rating + length	0.75	0.09	0.67	0.04	0.77	0.0005	0.69	0.0098
Rating + length + tense	0.74	0.63	0.67	0.083	0.77	0.0029	0.69	0.029
Rating + length + tense + 1× sentiment	0.73	1.0	0.66	0.16	0.77	0.004	0.68	8.9e−05
Rating + length + tense + 2× sentiments	0.71	1.0	0.68	0.0002	0.76	0.028	0.68	0.029
<i>Combined (text and metadata)</i>								
BOW + rating + lemmatize	0.78	0.064	0.74	0.0005	0.77	0.0023	0.80	0.0044
BOW + rating + 1× sentiment	0.79	0.0027	0.71	0.039	0.81	0.0002	0.83	0.001
BOW + rating + 1 sentiment + tense	0.78	0.0097	0.70	0.039	0.79	0.0002	0.81	0.0012
Bigram + rating + 1 sentiment	0.83	0.0039	0.81	9.5e−06	0.85	2e−05	0.79	0.042
Bigram − stopwords + lemmatization + rating + tense + 2× sentiment	0.82	0.0019	0.80	1.7e−06	0.85	2.5e−05	0.81	0.029
BOW + bigram + tense + 1× sentiment	0.87	0.0001	0.83	1.2e−05	0.91	1.9e−07	0.85	0.0002
BOW + lemmatize + bigram + rating + tense	0.88	7.6e−06	0.85	7.6e−07	0.92	1.2e−07	0.87	1.6e−05
BOW − stopwords + bigram + rating + tense + 1× sentiment	0.88	1.6e−06	0.85	7.6e−07	0.90	4.8e−06	0.86	0.0002
BOW − stopwords + lemmatization + rating + tense + 1× sentiment	0.79	0.064	0.74	0.0008	0.80	0.0014	0.80	0.029
BOW − stopwords + lemmatization + rating + tense + 2× sentiments	0.78	0.051	0.76	0.0012	0.81	0.0003	0.82	0.0002

tool does not miss any of them, with the compromise that a few of the reviews predicted as bug reports are actually not (false positives). For a balance between precision and recall combining bag of words, lemmatization, bigram, rating, and tense seems to work best.

Concerning feature requests, using the bag of words, rating, and one sentiment resulted in the highest precision with 89 %. The best *F*-measure was 85 % with bag of words, lemmatization, bigram, rating, and tense as the classification features.

The results for predicting user experiences were surprisingly high. We expect those to be hard to predict as the basic technique for user experiences shows. The best option that balances precision and recall was to combine bag of words with bigrams, lemmatization, the rating, and the tense. This option achieved a balanced precision and recall with a *F*-measure of 92 %.

Predicting ratings with the bigram, rating, and one sentiment score leads to the top precision of 92 %. This

result means that stakeholders can precisely select rating among many reviews. Even if not all ratings are selected (false negatives) due to average recall, those that are selected will be very likely ratings. A common use case would be to filter out reviews that only include ratings or to select another type of reviews with or without ratings.

Table 6 shows the ten most informative features of a combined classification technique for each review type.

4.3 Classification algorithms

Table 7 shows the results of comparing the different machine learning algorithms Naive Bayes, Decision Trees, and MaxEnt. We report on two classification techniques (bag of words and bag of words + metadata) since the other results are consistent and can be downloaded from the project Web site.² In all experiments, we found that binary

² <http://mast.informatik.uni-hamburg.de/app-review-analysis/>.

Table 6 Most informative features for the classification technique bigram – stop words + lemmatization + rating + 2× sentiment scores + tense

Bug report	Feature request	User experience	Rating
Rating (1)	Bigram (way to)	Rating (3)	Bigram (will not)
Rating (2)	Bigram (try to)	Rating (1)	Bigram (to download)
Bigram (every time)	Bigram (would like)	Bigram (use to)	Bigram (use to)
Bigram (last update)	Bigram (5 star)	Bigram (to find)	Bigram (new update)
Bigram (please fix)	Rating (1)	Bigram (easy to)	Bigram (fix this)
Sentiment (−4)	Bigram (new update)	Bigram (go to)	Bigram (can get)
Bigram (new update)	Bigram (back)	Bigram (great to)	Bigram (to go)
Bigram (to load)	Rating (2)	Bigram (app to)	Rating (1)
Bigram (it can)	Present cont. (1)	Bigram (this great)	Bigram (great app)
Bigram (can and)	Bigram (please fix)	Sentiment (−3)	Present simple (1)

Table 7 *F*-measures of the evaluated machine learning algorithms (B = binary classifier, MC = multiclass classifiers) on app reviews from Apple and Google stores

Type	Technique	Bug R.	<i>F</i> req.	<i>U</i> exp.	Rat.	Avg.
<i>Naive Bayes</i>						
B	Bag of words (BOW)	0.71	0.63	0.68	0.75	0.70
MC	Bag of words	0.66	0.31	0.43	0.59	0.50
B	BOW + metadata	0.79	0.71	0.81	0.83	0.79
MC	BOW + metadata	0.62	0.42	0.50	0.58	0.53
<i>Decision Tree</i>						
B	Bag of words	0.81	0.77	0.82	0.79	0.79
MC	Bag of words	0.49	0.32	0.44	0.52	0.44
B	BOW + metadata	0.73	0.68	0.78	0.78	0.72
MC	BOW + metadata	0.62	0.47	0.53	0.54	0.54
<i>MaxEnt</i>						
B	Bag of words	0.66	0.65	0.58	0.67	0.65
MC	Bag of words	0.26	0.00	0.12	0.22	0.15
B	BOW + metadata	0.66	0.65	0.60	0.69	0.65
MC	BOW + metadata	0.14	0.00	0.29	0.04	0.15

classifiers are more accurate for predicting the review types than multiclass classifiers. One possible reason is that each binary classifier uses two training sets: one set where the corresponding type is observed (e.g., bug report) and one set where it is not (e.g., not bug report). Concerning the binary classifiers Naive Bayes outperformed the other algorithms. In Table 7, the numbers in bold represent the highest average scores for the binary (B) and multiclass (MC) case.

4.4 Performance and data

The more data are used to train a classifier the more time the classifier would need to create its prediction model. This is depicted in Fig. 2 where we normalized the *mean time* needed for the four classifiers depending on the size of the training set.

In this case, we used a consistent size for the test set of 50 randomly selected reviews to allow a comparison of the results.

We found that when using more than 200 reviews to train the classifiers the time curve gets much more steep with a

rather exponential than a linear shape. For instance, the time needed for training almost doubles when the training size grows from 200 to 300 reviews. We also found that MaxEnt needed much more time to build its model compared to all other algorithms for binary classification. Using the classification technique BoW and Metadata, MaxEnt took on average ~ 40 times more than Naive Bayes and ~ 1.36 times more than Decision Tree learning.

These numbers exclude the overhead introduced by the sentiment analysis, the lemmatization, and the tense detection (part-of-speech tagging). The performance of these techniques is studied well in the literature [4], and their overhead is rather exponential to the text length. However, the preprocessing can be conducted once on each review and stored separately for later usages by the classifiers. Finally, stopword removal introduces a minimal overhead that is linear to the text length.

Figure 3 shows how the accuracy changes when the classifiers use larger training sets. The precision curves are

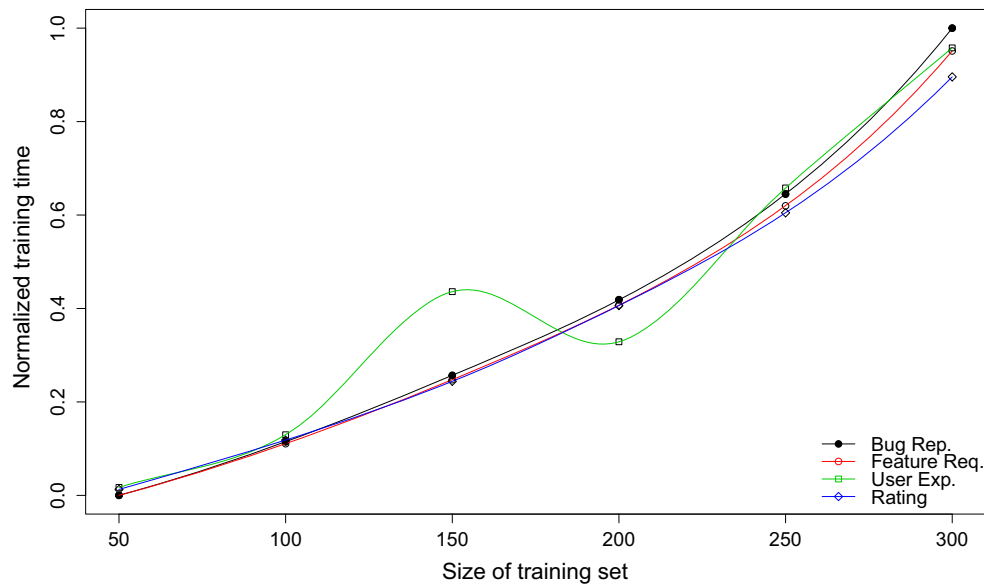


Fig. 2 How the size of the training set influences the time to build the classification model (Naive Bayes using BoW + rating + lemmatization (see Table 4))

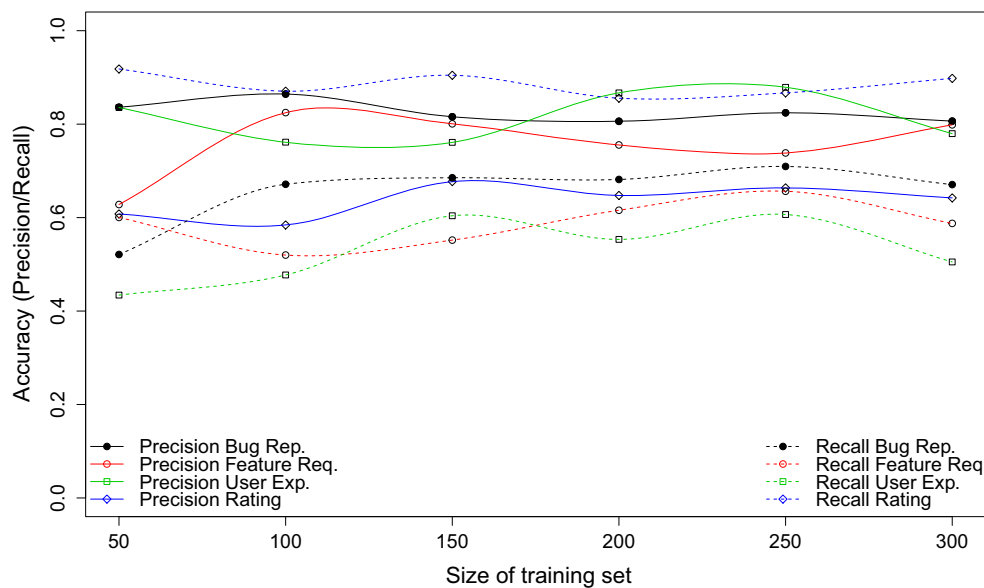


Fig. 3 How the size of the training set influences the classifier accuracy (Naive Bayes using BoW + rating + lemmatization (see Table 4))

represented with continuous lines, while the recall curves are dotted. From Figs. 2 and 3 it seems that 100–150 reviews are a good size of the training sets for each review type, allowing for a high accuracy while saving resources.

With an equal ratio of candidate and non-candidate reviews the expected size of the training set doubles leading to 200–300 reviews per classifier recommended for training.

Finally, we also compared the accuracy of predicting the Apple AppStore reviews with the Google Play Store reviews. We found that there are differences in predicting the review types between both app stores as shown in

Tables 8 and 9. The highest values of a metric are emphasized as bold for each review type. The biggest difference in both stores is in predicting bug reports. While the top value for *F*-measure for predicting bugs in the Apple AppStore is 90 %, the *F*-measure for the Google Play Store is 80 %. A reason for this difference might be that we had less labeled reviews for bug reports in the Google Play Store. On the other hand, feature requests in the Google Play Store have a promising precision of 96 % with a recall of 88 %, while the precision in the Apple AppStore is 88 % with a respective recall of 84 %, by

