Performance of Defect Prediction in Rapidly Evolving Software

Davide G. Cavezza, Roberto Pietrantuono, Stefano Russo

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell’Informazione
Università degli Studi di Napoli Federico II
Motivations (1/2)

♦ Defect prediction gives insight into product quality
  ▪ Useful to make decisions on when to release
♦ Rapidly evolving development paradigms
  ▪ Agile methods
    • Continuous Integration, Continuous Delivery
  ▪ Short release-cycle required
Motivations (2/2)

- Classical “static” defect prediction: choose a model and cross-validate it on all the available data
  - There is no insight on how long the model remains valid
  - This is a key concern in rapidly changing software
- We propose a dynamic prediction model
  - The model is periodically retrained with the most recent data
Commit-level defect prediction

- Relationship between a commit’s features and its defectiveness
- Learning algorithms are used to predict if a commit is defective given its feature values
  - Supervised learning: the training set consists of commits whose defectiveness has been assessed
Dynamic prediction phases

1. Model selection
2. (Re)training
3. Prediction
4. Evaluation
Model selection

You need to choose:
- a performance metric for x-validation
- the time extension of the historical data
(Re)training

Best performing model

Training

Trained model

Most recent labelled data
Performance of Defect Prediction in Rapidly Evolving Software

Prediction

Trained model

Commit request

Warning

Yes

Defective?

No

Accept change
Execution periodically
- Time interval between two evaluations must be chosen

- Predicted defectiveness
- Actual defectiveness

Compare

Performance measure

<threshold?

Yes
Select new model

No
Keep using current model
Experimental setting (1/4)

♦ Eclipse JDT
♦ Commit data extracted from Git repository
♦ SZZ algorithm to distinguish defective and non-defective commits

<table>
<thead>
<tr>
<th>Total commits</th>
<th>Timespan</th>
<th>Defective commits</th>
<th>Non-defective commits</th>
</tr>
</thead>
<tbody>
<tr>
<td>26,009</td>
<td>From 2001-06-05 To 2014-12-13</td>
<td>13,984 (53.77%)</td>
<td>12,025 (46.23%)</td>
</tr>
</tbody>
</table>
## Experimental setting (2/4)

### ♦ Commit-level features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of modified files (NF)</td>
<td>Number of files modified in the commit</td>
</tr>
<tr>
<td>Entropy</td>
<td>Scattering of modifications throughout the modified files</td>
</tr>
<tr>
<td>Lines added (LA)</td>
<td>Number of lines added in the commit</td>
</tr>
<tr>
<td>Lines deleted (LD)</td>
<td>Number of lines deleted in the commit</td>
</tr>
<tr>
<td>FIX</td>
<td>Binary value indicating whether or not the commit is a bug fix</td>
</tr>
<tr>
<td>Number of developers (NDEV)</td>
<td>Number of developers that changed the files touched by the commit before the commit was issued</td>
</tr>
<tr>
<td>AGE</td>
<td>Average time interval between the current and the last change across all the involved files</td>
</tr>
<tr>
<td>Number of unique changes (NUC)</td>
<td>Number of unique commits that last changed the involved files</td>
</tr>
<tr>
<td>Experience (EXP)</td>
<td>Experience of the developer, measured as the number of changes previously committed by him</td>
</tr>
<tr>
<td>Recent experience (REXP)</td>
<td>Number of past commits of the same developer, each weighted proportionally to the number of years between that commit and the measured one</td>
</tr>
</tbody>
</table>
Experimental setting (3/4)

- Repartition of training and test sets:
  - Training sets duration: 9 months
  - Test sets duration: 3 months
Experimental setting (4/4)

♦ Models used:
  ▪ J48
  ▪ OneR
  ▪ NaiveBayes

♦ Performance metric:
  ▪ F-measure = \[
    2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}
  \]
Results: Static vs Dynamic model
Discussion

Dynamic model outperforms static

But there are two situations in which neither can predict defectiveness with sufficient accuracy
Future challenges

♦ Assessment of the influence of parameters like
  ▪ Training windows extension
  ▪ Frequency of evaluations
  ▪ Performance measure choice

♦ Problem: lack of knowledge on recent commit defectiveness
Thank you!
Questions?