PREDICTIVE TECHNIQUES IN SOFTWARE ENGINEERING: APPLICATION IN SOFTWARE TESTING

Jelber Sayyad Shirabad
Lionel C. Briand, Yvan Labiche, Zaheer Bawar

Presented By: Faezeh R. Sadeghi
Overview

- Introduction
- Process of Applying ML to Software Engineering (SE)
- Applications of Predictive Models in SE
- MELBA (MachinE Learning based refinement of BLAck-box test specification)
- Future Directions
Software Development Cycle

- Data is created at different stages in the software development cycle

- i.e. Creation of documents in the analysis stage, diagrams in the design stage, code in implementation stage, and test cases in the testing phase
Use of Technology to ...

- Better understand a system
- Make more informative decisions as needed through the life of an existing system
- Apply lessons learned from building other systems to the creation of a new system

- MACHINE LEARNING (ME)!
Steps in Applying ML to SE

1. Understanding the Problem
2. Casting Original Problem as a Learning Problem
3. Collection of Data and Relevant Background Knowledge
4. Data Preprocessing and Encoding
5. Applying Machine Learning and Evaluating Results
6. Field Testing and Deployment
Step 1: Understanding the Problem

- Estimate cost or effort involved in developing a software
- To be able to characterize the quality of a software system
- To be able to predict what modules in a system are more likely to have a defect
Step 2: Casting the Original Problem as a Learning Problem

- Decide on how to formulate the problem as a machine learning task. i.e. Classification vs. Numeric prediction problem
- May not be straightforward and require further refinement of the original problem into sub problems
Step 3: Collection of Data and Relevant Background Knowledge

- Easier to collect data than knowledge background
- i.e. It is easy to collect data regarding faults discovered in software system and changes applied to the source to correct a fault, but there is no agreed upon domain theory describing software systems.
- We need to limit ourselves to incomplete background knowledge, i.e. Choosing a subset of the system
Step 4: Data Pre-Processing and Encoding

- Noise reduction
- Selecting appropriate subset of collected data
- Determining a proper subset of features that describe concepts to be learned
- Data and background knowledge need to be described and formatted in a manner that complies with the requirements of algorithm used
Step 5: Applying Machine Learning and Evaluating Results

- Need to measure the “goodness” of what is learned
- Accuracy in classification problems
- Mean Magnitude of Relative Error (MMRE) in numeric prediction
- Software Engineering: Percentage of the examples with magnitude of relative error (MRE) $\leq x$ (known as PRED) to assess cost, effort, and schedule models
- MMRE $< 25\%$ is considered good, PRED $> 75\%$
Step 5: Applying Machine Learning and Evaluating Results (cont.)

- If what is learned is inadequate
  - Retry this step by adjusting parameters OR
  - Reconsider the decisions made in earlier stages
Step 6: Field Testing and Deployment

- Used by intended users
- Unfortunately number of articles discussing actual use and impact of the research in industry is very small
  - Confidentiality
Applications of Predictive Models in SE

- Software Size Prediction
- Software Quality Prediction
- Software Cost Prediction
- Software Defect Prediction
- Software Reliability Prediction
- Software Reusability Prediction
- Recent Uses (Software Test Suite Reliability Prediction)
Software Size Prediction

Methods for measuring

- Lines of code (LOC)
  - Regolin, de Souza, Pozo, and Vergilio (2003) used neural networks (NN) and genetic programming (GP)

- Component-based method (CBM)
  - Dolado (2000) concluded that NN and GP perform as well or better than multiple linear regression

- Function points
  - Pendharkar (2004) used decision tree regression to predict the size of OO components
Software Quality Prediction

- ISO 9126: Functionality, reliability, efficiency, usability, maintainability, and portability

- McCall: factors composed of quality criteria

- Fault Density: a measure for quality
  - Genetic programming used by Evett and Khoshgoftaar (1998) to build models that predict the number of faults expected in each module
  - Xing, Guo, and Lyu (2005) used SVM
  - Seliya and Khoshgoftaar (2007) used an EM semi-supervised learning algorithm
Software Cost Prediction

- Early estimation allows us to determine the feasibility
- Detailed estimations allow managers to better plan the project
Cost Estimation Methods

- Expert opinion
- Analogy based on similarity to other projects
- Decompose project
  - Components to deliver
  - Tasks to accomplish
  - Total estimate of cost of individual components
- Use of estimation models
Examples of Cost Prediction:

- Shepperd and Schofield (1997)
  - Use of analogies for effort prediction
  - Projects characterized in terms of attributes such as the number of interfaces, development method, or the size of functional requirements document
  - Euclidean distance in n-dimensional space of project feature

- Decision trees and neural networks used

- More recently: Oliveira (2006) uses support vector regression (SVR), radial basis function neural networks (RBFNs) and linear regression-based models
  - In an experiment SVR outperforms RBFN and linear regression
Software Reliability Prediction

- The probability of a failure-free operation of a computer program for a given environment for a given time
- Pai and Hong (2006), SVM-based models with simulated annealing perform better than existing Bayesian models
- Neural networks can also be used
Software Reusability Prediction

- Use of existing software knowledge
- Aim:
  - Increase productivity of software developers
  - Increase quality of end product
- Label code as reusable or non-reusable, then use software metrics to describe the example of interest
Related Work on Software Reusability

- Mao, Sahraoui, and Lounis (1998)
- Verify hypothesis of correlation between reusability and quantitative attributes of a piece of software: inheritance, coupling, and complexity
- Four labels, from “totally reusable” to “not reusable at all”
Recent Uses:

- Predict defect contents of documents after software inspection.
- Predicting stability of object-oriented software
  - Grosser, Sahroui, and Valtchev (2002)
- Software Test Suite Reliability Prediction
  - Lionel C. Briand, Yvan Labiche, Zaheer Bawar (2008)
Software Test Suite Reliability Prediction

- Problem:
  - Developers are faced with test suites which have been developed with no apparent rationale
  - Developers have to evaluate and reduce or augment them depending on the level of weakness or redundancy
  - Quality prediction???
Goal:

- An automated methodology based on machine learning to help software engineers analyze the weakness of test suites to be able to improve them
- Re-engineering of test suites

- Machine Learning based refinement of Black-box test specification (MELBA)
Melba Methodology
Category Partitioning:

Tested function

- Function’s parameters and environment variables
  - Parameter/variable characteristics
    - Characteristic’s blocks
      - Values of block
Design

- First case: Test suite but not CP specification
  - Testers must then build the CP specification based on their understanding of the software functional behaviour
  - iteratively refine it

- Second Case: CP specification is used to generate the test suite and the test suite must evolve to account for changes in the system under test (Evolution context).
Potential Problems and Causes in Test Suite

- Missclassifications → Missing Category
- Too Many Test Cases for a Rule → Ill-defined Choices, Missing Test Cases
- Unused Categories → Redundant Test Cases
- Missing Combinations of Choices → Useless Categories, Impossible Combinations of Choices
Case Study: PackHexChar Program

- **Inputs:**
  - RLEN String of characters
  - ODD_DIGIT

- **Program output**
  - Array of Bytes and Integer value
  - -1 for even RLEN
  - -2 for illegal RLEN
  - -3 for illegal ODD_DIGIT
Parameter RLEN
CATEGORY 1—Valid values for rlen
  C1: rlen = 0
  C2: rlen = [1…sLength]

CATEGORY 2—Invalid values for rlen
  C3: rlen < 0
  C4: rlen > sLength

Parameter ODD_DIGIT
CATEGORY 3—Valid values for odd_digit
  C5: odd_digit = -1
  C6: odd_digit = [0…9]
  C7: odd_digit = [A…F]
  C8: odd_digit = [a…f]

CATEGORY 4—Invalid values for odd_digit
  C9: odd_digit < -1
  C10: odd_digit > 15

Parameter String S
CATEGORY 5—Length of S
  C11: sLength = 0
  C12: sLength > 0

CATEGORY 6—Type of characters in the first rlen characters of S
  C13: AllHexadecimal
  C14: AllNonHexadecimal
  C15: MixedChars

CATEGORY 7—Case of chars in the first rlen chars of the string S
  C16: allNumbers
  C17: allLowerCase
  C18: allUpperCase
  C19: MixedCase

CATEGORY 8—The first rlen characters of the string contains boundary Value [0, 9, a, f, A, F]
  C20: Contains [0]
  C21: Contains [9]
  C22: Contains [a]
  C23: Contains [f]
  C24: Contains [A]
  C25: Contains [F]
  C26: ContainsMixed
  C27: ContainsNone

CATEGORY 9—Number of hexadecimal characters in the first rlen characters of S
  C28: Odd
  C29: Even
  C30: Zero

CATEGORY 10—Position of the first non-hexadecimal character in the first rlen characters of S
  C31: First
  C32: Middle
MELBA: Snapshot of Decision Tree
# Melba Snap Shot of Unused CP Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Definition</th>
<th>Possible causes</th>
<th>Relevant information</th>
<th>Analyzed</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C 6.Ch 6.2)</td>
<td>all hexa chars[]</td>
<td>Investigate</td>
<td>Investigate</td>
<td></td>
</tr>
<tr>
<td>(C 6.Ch 6.3)</td>
<td>mixed hexa chars[]</td>
<td>Investigate</td>
<td>Investigate</td>
<td></td>
</tr>
<tr>
<td>(C 9.Ch 9.2)</td>
<td>all hexa chars[]</td>
<td>Investigate</td>
<td>Investigate</td>
<td></td>
</tr>
<tr>
<td>(C 9.Ch 9.3)</td>
<td>mixed hexa chars[]</td>
<td>Investigate</td>
<td>Investigate</td>
<td></td>
</tr>
</tbody>
</table>
Melba : Snap Shot of Unused Combination of Choices

<table>
<thead>
<tr>
<th>Combination missing in the tree</th>
<th>Possible causes</th>
<th>Relevant information</th>
<th>Analyzed</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Combination](C 12.Ch 12.2) ![Combination](C 11.Ch 11.3)</td>
<td>Already used in the ATS</td>
<td>Investigate</td>
<td><img src="unchecked" alt="Check Box" /></td>
</tr>
<tr>
<td>![Combination](C 12.Ch 12.2) ![Combination](C 11.Ch 11.1) ![Combination](C 1.Ch 1.1) ![Combination](C 2.Ch 2.2) ![Combination](C 5.Ch 5.1) ![Combination](C 8.Ch 8.1)</td>
<td>Already used in the ATS</td>
<td>Investigate</td>
<td><img src="unchecked" alt="Check Box" /></td>
</tr>
<tr>
<td>![Combination](C 12.Ch 12.2) ![Combination](C 11.Ch 11.1) ![Combination](C 1.Ch 1.1) ![Combination](C 2.Ch 2.2) ![Combination](C 5.Ch 5.1) ![Combination](C 8.Ch 8.2)</td>
<td>Already used in the ATS</td>
<td>Investigate</td>
<td><img src="unchecked" alt="Check Box" /></td>
</tr>
</tbody>
</table>
Results:

- Improvement of cp specification
- Improvement of test suites
Future Work for MELBA

- Investigate other black box testing specifications
- Additional evaluations of the iterative process on programs of varying sizes and complexities
- User-friendly automated tool support
Future Directions in SE:

- Self-configuring
- Self-optimizing
- Self-healing
- Self-protecting