Automated Software Defect Prediction Using Machine Learning

Leandro L. Minku
School of Computer Science
University of Birmingham
Software Defect Prediction

Software code is composed of several components.
Software Defect Prediction

Testing all these components can be very expensive.
Software Defect Prediction

If we know which components are likely to be defective, we can increase testing cost-effectiveness.
Predictive models can be created to identify components likely to be defective by using past software releases and bug fixes as training data for learning machines.
Vectorising Past Projects

- Past projects need to be represented in a format suitable for learning machines.
- Example: vectorising components from past projects based on static code features.
- Quickly and automatically collected from the source code.

<table>
<thead>
<tr>
<th>Branch count</th>
<th>Code + comment LOC</th>
<th>Halstead difficulty</th>
<th>Cyclomatic complexity</th>
<th>...</th>
<th>Defective?</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>15</td>
<td>14.57</td>
<td>3</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>11.65</td>
<td>2</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>6.43</td>
<td>5</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>15</td>
<td>40</td>
<td>14.8</td>
<td>8</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>16</td>
<td>35</td>
<td>16.9</td>
<td>9</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>
Naive Bayes: an example of learning machine

- Bayes theorem:
  \[ p(C|F_1, \ldots, F_n) = \frac{p(C) p(F_1, \ldots, F_n|C)}{p(F_1, \ldots, F_n)} \]

- Assuming independence:
  \[ p(C|F_1, \ldots, F_n) \propto p(C) \prod_{i=1}^{n} p(F_i|C) \]

- Naive bayes classifier:
  \[ \text{classify}(f_1, \ldots, f_n) = \arg\max_c p(C = c) \prod_{i=1}^{n} p(F_i = f_i|C = c) \]
How to Use Naive Bayes: an illustrative example

<table>
<thead>
<tr>
<th>Branch count</th>
<th>Code + comment LOC</th>
<th>Defective?</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>15</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>No</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>No</td>
</tr>
<tr>
<td>15</td>
<td>40</td>
<td>Yes</td>
</tr>
<tr>
<td>16</td>
<td>35</td>
<td>Yes</td>
</tr>
</tbody>
</table>

classify\left(f_1, \ldots, f_n\right) = \arg \max_c p(C = c) \prod_{i=1}^{n} p(F_i = f_i | C = c).

Example: classify (bc = 16, loc = 39)

P(C = No) = 3/5 = 0.6
P(bc = 16 | No) = Gauss(x = 16, mean = 5.67, stdev = 3.06) = 0.0004
P(loc = 39 | No) = Gauss(x = 39, mean = 16.67, stdev = 10.41) = 0.0038

C = No --> 0.6 * 0.0004 * 0.0038 = 0.000000912
How to Use Naive Bayes: an illustrative example

<table>
<thead>
<tr>
<th>Branch count</th>
<th>Code + comment LOC</th>
<th>Defective?</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>15</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>No</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>No</td>
</tr>
<tr>
<td>15</td>
<td>40</td>
<td>Yes</td>
</tr>
<tr>
<td>16</td>
<td>35</td>
<td>Yes</td>
</tr>
</tbody>
</table>

\[
\text{classify}(f_1, \ldots, f_n) = \arg \max_c p(C = c) \prod_{i=1}^{n} p(F_i = f_i | C = c).
\]

Example: classify (bc = 16, loc = 39)

\[
P(C = \text{Yes}) = \frac{2}{5} = 0.4
\]

\[
P(bc = 16 | \text{Yes}) = \text{Gauss}(x = 16, \text{mean} = 15.5, \text{stdev} = 0.71) = 0.4385
\]

\[
P(loc = 39 | \text{Yes}) = \text{Gauss}(x = 39, \text{mean} = 37.5, \text{stdev} = 3.53) = 0.1033
\]

\[
C = \text{Yes} \rightarrow 0.4 \times 0.4384 \times 0.1033 = 0.0181
\]
How to Use Naive Bayes: an illustrative example

<table>
<thead>
<tr>
<th>Branch count</th>
<th>Code + comment LOC</th>
<th>Defective?</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>15</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>No</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>No</td>
</tr>
<tr>
<td>15</td>
<td>40</td>
<td>Yes</td>
</tr>
<tr>
<td>16</td>
<td>35</td>
<td>Yes</td>
</tr>
</tbody>
</table>

\[
classify(f_1, \ldots, f_n) = \arg\max_c p(C = c) \prod_{i=1}^{n} p(F_i = f_i | C = c).
\]

Example: classify \((bc = 16, loc = 39)\)

\(C = \text{No} \implies 0.6 \times 0.0004 \times 0.0038 = 0.000000912\)

\(C = \text{Yes} \implies 0.4 \times 0.4384 \times 0.1033 = 0.0181\)

Class = Yes
WEKA

Open source software that contains implementations of several learning machines.

http://www.cs.waikato.ac.nz/ml/weka/
Issues to Consider: class imbalance

- The number of examples of faulty modules is usually much smaller than the number of non-faulty modules.
- Machine learners may tend to classify everything as negative!
- Possible fix:
  - Undersample examples from non-faulty class.
  - Other more advanced techniques.

Issues to Consider: data availability

- There is no data from a project before its first version is rolled out.
- How to predict defects for a project in its first version?
- Possible fix:
  - Use data on other projects.


Issues to Consider: temporal behaviour

- Typically, all available examples from all previous versions of a software are used to build fault prediction models.

- However, changes may happen from one version to the other:
  - modules that are likely to be faulty in one version may not be faulty in another.

- Possible fix:
  - Try and identify which previous versions are more useful.

Thank you!

To appear in Dec 2014.