Discovering Software Bugs with Bayesian Non-Parametric Models

Finale Doshi and Jurgen Van Gael, University of Cambridge

Abstract

We are interested in the problem of identifying bugs in computer programs and additionally patterns of use that cause computer programs to crash. Specifically, we begin with code that has been instrumented to record how many times certain lines are executed during a run through the program. Given successful and failed runs of the program, we wish to identify potential causes of failure (suggested in the Cooperative Bug Isolation project [1]). We assume that the number of times a line in the source code is executed, which we call software probes, reflect how often certain latent usage patterns are executed during the run. For example, we may expect to see one set of probe counts from a mouse-click, while another pattern of probe counts may correspond to changing a directory.

The Delta-LDA framework of Andrzejewski et. al. [2] first finds usage patterns in successful runs. They next fix these patterns and then try to learn new usage patterns in the failed runs, thus allowing for the fact that failed runs will contain a combination of innocuous patterns as well as the culprit patterns that caused the program to fail. We use a similar approach, but we try using a Hierarchical Dirichlet Process [3] and Indian Buffet Process [4] as a prior for the probes that we expect usage patterns to use. This approach allows us to overcome a shortcoming in the previous approach, namely, that the number of usage patterns must be prespecified in the Delta-LDA model, and setting that figure a priori is difficult. We develop sampling schemes to do inference in this model and demonstrate our approach on both simulated and real data.

Motivation

- **Question**: Are there particular usage patterns that cause software to fail? Knowing what type of use causes a program to crash may provide useful debugging information.
- **Approach**: suppose we instrument a program to keep track of various features, for example, how often a certain line of code is executed during a run of the program.
- **Goal**: find patterns that correspond to usage patterns (such as a mouse click) and determine what usage patterns are associated with program failures.

Instrumenting the Software

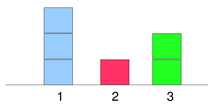
- The data was obtained from the Cooperative Bug Isolation project [1].
- Concept:
  - Code is instrumented to count the number of times certain lines are executed:

```
1: print “starting program”
2: filename = get_keyboard_input()
3: action = get_keyboard_input()
4: if action == “load”
5:       read( filename )
6: if action == “save”
7:       write( filename , “some data” )
8: print “finished execution”
```

Counter 1 → line 1; Counter 2 → line 5; Counter 3 → line 7

- Different usage patterns will produce different histograms of counts:

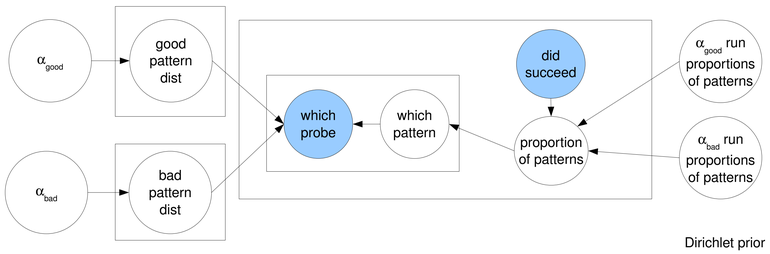
- If we consider multiple runs of the program, we will have a histogram containing a mixture of different patterns:



- For storage reasons, the counts are subsampled.
- Data consists of a few hundred runs with tens of probes, but we can imagine obtaining very large data sets (if many people run instrumented software on their desktops).

Parametric approach: DeltaLDA [2]

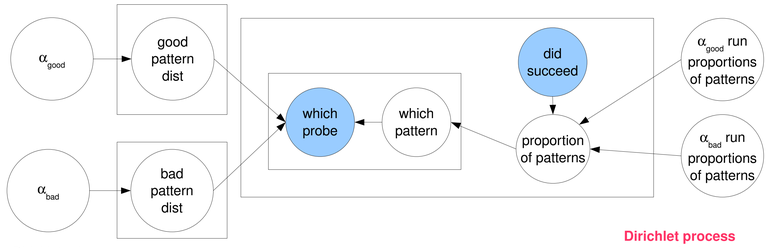
- Latent Dirichlet Allocation (LDA) assumes that each count is created by the following process:
  - Choose a usage pattern k with probability pk
  - Choose a probe based on its popularity within the usage pattern.
- Based on the following graphical model:



- Trained in two steps: first train with successful runs only, then add in failed runs holding patterns from successful runs fixed.
- DeltaLDA finds buggy usage patterns, but we need to specify the number of patterns in advance.

Non-Parametric Approach I: DeltaHDP

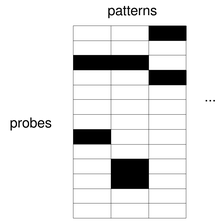
- The hierarchical Dirichlet process is a natural extension of the LDA model, using a Dirichlet process on the number of patterns instead of assuming a fixed number:



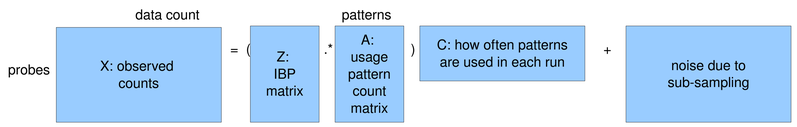
- Efficient inference procedures exist for it.

Non-Parametric Approach II: IBP

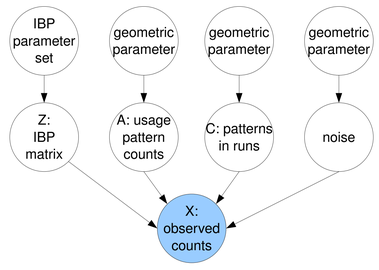
- Both the LDA and HDP approach focus on the proportions of probe counts, that is, the shape of the usage pattern histogram, rather than the actual values of the counts.
- The Indian Buffet Process places a prior over sparse 0-1 matrices:



- The IBP is a reasonable prior on which probes are part of which patterns if we believe that the number of probes used in any pattern will be small (each usage pattern touches only a small part of the code).
- We model the counts as being produced by the following equation:



- Corresponding to the following graphical model:

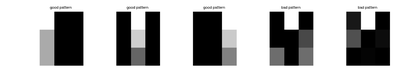


- This model might be more faithful to the actual form of the data (product of usage patterns).
- Initially, we learn Z and A for only the successful runs. Then we train on the failed runs holding the original patterns fixed but allowing additional patterns to be added to Z and A.
- However, inference in the IBP is trickier. We combine:
  - Gibbs sampling
  - MH proposals for splitting and merging patterns, creating and destroying patterns, and scaling patterns

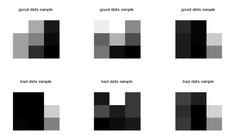
Results on Toy Data

A small toy example:

The actual patterns:



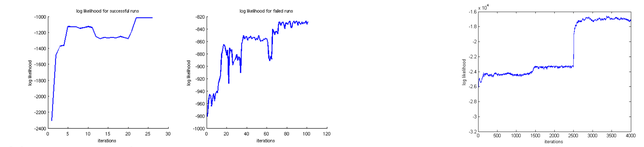
Some sample data (100 total runs):



Patterns found by the IBP model:

Patterns found by the HDP model:

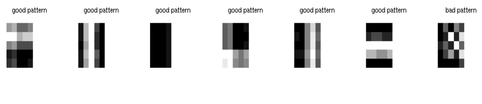
Log-likelihoods under the models (IBP, HDP):



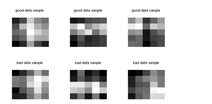
A larger toy example:

The actual patterns:

Patterns found by the IBP model:

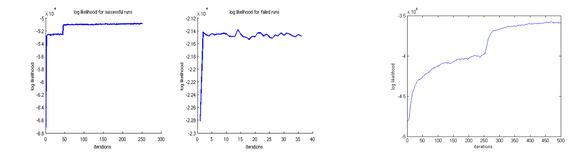


Some sample data (700 total runs):



Patterns found by the HDP model:

Log-likelihoods under the models (IBP, HDP):



Results on Real Data

- Results from the HDP (right) match those found by DeltaLDA without needing to assume a particular number of patterns.

- Could not run IBP model because inference took too long.

Future Work

- More efficient inference procedures for the IBP
- For both models, can we handle the data sequentially?
- How useful is this for finding bugs in software, and what are other applications?

References:

[1] Liblit, B. Cooperative Bug Isolation: Winning Thesis of the 2005 ACM Doctoral Dissertation Competition. Lecture Notes in Computer Science Vol. 4440. Springer 2007.
[2] Andrzejewski D., Mulhern, A., Liblit, B. Zhu J. Statistical Debugging using Latent Topic Models. ECML 2007
[3] Y. W. Teh, M. I. Jordan, M. J. Beal, D. M. Blei, Hierarchical Dirichlet Processes, Journal of the American Statistical Association, 2006
[4] Griffiths, T. and Ghahramani, Z. Infinite Latent Feature Models and the Indian Buffet Process. Gatsby TR 2005-001, 2005.