Jacob Kilver

Knowledge Based AI

Assignment 5

10 July 2016

# Introduction

This paper will explore the use of diagnosis as applied to software debugging at my place of employment. My full-time job is as a developer on a support team for ATM software, so a large portion of my time is spent analyzing and diagnosing issues related to software. I could personally identify with much of what was discussed in the lecture on diagnosis.

# Problem Description

Diebold Inc. (<http://www.diebold.com/>) is a global manufacturer of automated teller machines (ATMs) and other financial institution products and services. While ATMs seem rather simple, there are many things that can go wrong with them. As a software support engineer, I am the third layer of engineering support, but the first team that has access to the source code of our ATM application. As such, my group handles a wide variety of software issues, which can make diagnosing an issue quite a challenge. Here is a sampling of the difficulties my team experiences in full-stack software debugging:

* Poor descriptions from the field.
  + Many times an escalation has false or missing data
* Incomplete data from field
  + We have a tool that is supposed to gather everything we need for analysis, but it usually misses a few things
* Lots of code
  + The application I help manage currently has a SLOC count of \_\_\_\_\_, not including the other layers
* Multiple software layers
  + The complete software stack consists of no fewer than 5 layers, not to mention all the software add-ons that are available
* Poor, missing, and unclear documentation/specifications
  + Defining what the correct behavior should be is not always easy
* Differing customer opinions/desires
  + Some customers want the software to behave like certain legacy applications
  + Others want customized solutions that do not accommodate all clients
* Multivendor environments
  + A surprising number of customers want hardware from one vendor and software from another
  + This requires knowledge that is legally not within your domain
  + At times this feels like trying to get Android to run on an iPhone

The only input my team receives typically is the “effect” from Figure 1. We know from our background knowledge about a few “rules” that can help solve the issue. As such, the only available tools available are induction and abduction, neither of which are truth preserving. There can be multiple explanations for the observed data, which further increases the complexity.

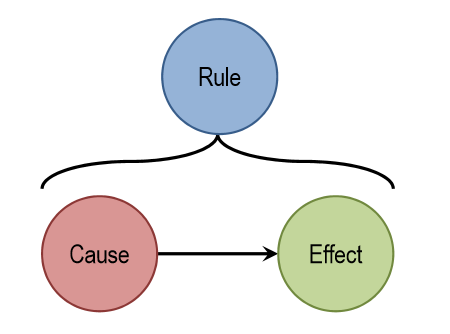


Figure : Diagram showing causal relationships

# Approach

Throughout my time in the OMSCS program I have attempted to design a number of systems that could aid my team in diagnosing issues. Up until now my focus has been on machine learning because we have a large amount of data from years of previously reported bugs. This would be an example of induction. An “effect” is reported from the field. Through whatever process, a cause is determined, and the cause and effect are stored in a database. When a new “effect” is reported, the database can be queried for similar issues. A number of machine learning techniques could be applied here (k nearest neighbors, support vector machines, decision trees). One method that I was intrigued with was ensemble learners.

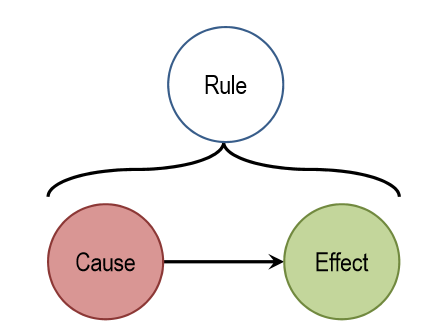


Figure : Process of induction - given a cause and effect, derive a rule

Prior to now I had not given much thought to using abduction for this task. However, this method fills in one of the gaps from machine learning. Using induction requires the cause to be known. In other words, it only works on problems that have already been solved. If an entirely new issue comes in the machine learning agent would not be able to handle it very well.

**Actually, ensemble learners might be better for abduction. What is good for induction then? Clustering?**

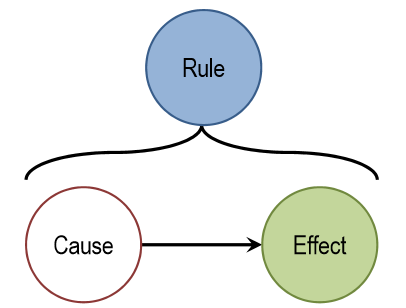


Figure : Process of abduction - given a rule and effect, derive a cause

I find it intriguing how these processes are mutually cooperative.

# Conclusions