Jacob Kilver
Knowledge Based AI
Assignment 5
10 July 2016

1. 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.

2. 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
  - While we don't keep official statistics, I estimate the application I help manage currently has a SLOC count of over 1 million, 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 not always 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.

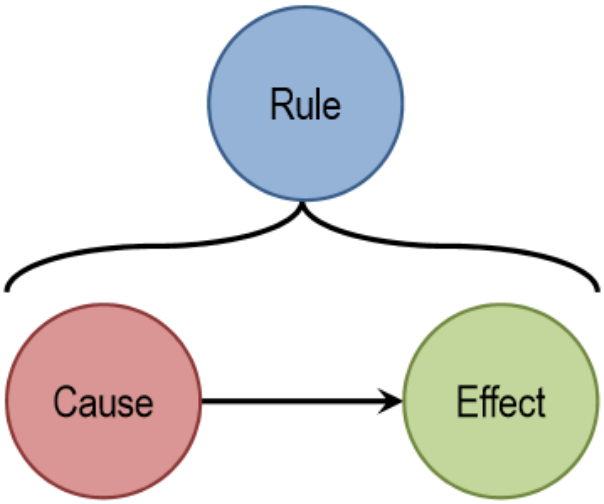

Figure 1: Diagram showing causal relationships

3. 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 in order to leverage the large amount of data from years of previously reported and solved bugs. However this could be viewed as a form of cognition, namely inductive reasoning (Figure 2). Using the vast amount of background knowledge, we could create an agent that can derive rules for later use. The machine learning technique that best applies here would be clustering – labelling sets of unlabeled data. Clustering attempts to group similar data points together, so in this case it would attempt to place similar cause-effect relationships together. It is important that the clusters be formed at the right level of abstraction, though. If the clusters are too large, it is likely that not much unique information is within the cluster. The rules would be too vague. If the clusters are too small the rules would be too narrow to be of any use. Fortunately, there are techniques that have been developed that can find the optimal number of clusters (rules) that a data set should have.

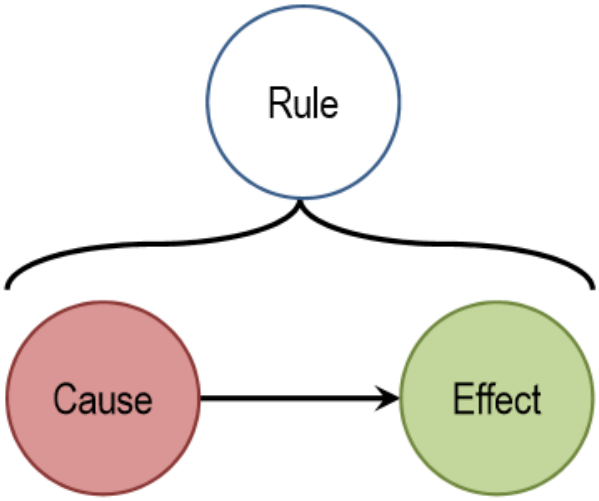

Figure 2: Process of induction - given a cause and effect, derive a rule

With the "effects" from the field and the "rules" derived from clustering, we have the two parts necessary to perform abductive reasoning (Figure 3). Abductive reasoning attempts to use known cause-effect relationships to explain effects that are observed in the world. The technique I have found most intriguing to implement abductive reasoning is ensemble learning. Rather than having one definitive rule for classifying incoming data, ensemble learning uses many weak learners to classify input. So from the clustering step prior to this, the rules should be more on the narrow, simple side so that when they are combined they form a more complex rule.

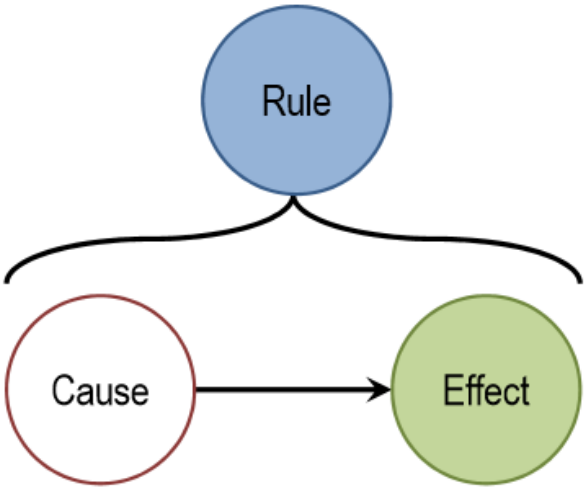

Figure 3: Process of abduction - given a rule and effect, derive a cause

Let's take an example of an issue a recently diagnosed and see how an agent could use abductive reasoning and ensemble learners to solve it. First, an explanation of the issue. The customer was concerned about the growing number of swipe (a.k.a dip) card readers in their fleet experiencing hardware faults which caused the ATM to go out of service. Initial analysis revealed that there was a jam in the card reader but there was no actual problem with the card reader itself. Further investigation showed that just prior to the jam a user had incorrectly inserted a card in the reader. However, the

customer had not configured the software to display a screen telling the customer to try again. Instead, they had the software configured to simply attempt to read the card again, which resulted in the jam status.

Second, let's assume that the agent has some rules that it found by analyzing past issues which could be applied here. See example rules in Table 1. In order to keep this paper a reasonable length and to avoid disclosing any proprietary information, I have kept these on the simple side. The rule is simply the combination of the cause and effect.

Table 1: Subset of possible rules for agent

Cause	Effect	Solution
Card jammed in reader	Card jam status reported	Remove jam
Card inserted on ATM startup	Hardware error reported	Remove card
Card reader not available	ATM goes out of service	Repair card reader

In this case all three effects were reported: a card jam is reported along with a hardware error and the ATM goes out of service. With this information the agent has the potential to abduce that the card jam is caused by a card that is inserted into the machine, causing the card reader to be unavailable to accept new cards, and thus the ATM goes out of service. While this is not the most thorough example, I hope it illustrates how an ensemble learner combines simple rules to form complex answers.

I find it intriguing how the process of clustering (induction) and ensemble learning (abduction) are mutually beneficial. One process provides information that the other uses. This positive feedback loop makes the agent smarter over time without the need for additional training. It simply becomes smarter as its knowledge base increases.

4. Conclusions

Software debugging is a process that relies heavily on abduction to explain problems within code. As software is becoming more prevalent and increasingly complex, tools need to be developed to aid in the debugging and diagnosis of software. This is where machine learning coupled with cognitive systems can help. As shown in this paper, background knowledge from previous software issues can be grouped into rules which can be used to reason about issues seen in new problems. While this was a far from comprehensive treatment, it did show some of the power that such a system could have when properly equipped.