Failure Diagnosis Using Decision Trees

Authors: M. Chen; A.X. Zheng; J. Lloyd; M.I. Jordan; E. Brewer

IEEE ICAC 2004

Outline

- Introduction
- Data Collection
- Problem Description
- Decision Tree Learning Approach
- Experimental Setup
- Results
- Discussion and Related Work
- Conclusion
- Pros & Cons

Introduction

- Diagnose failure occurs in large scale Internet system automatically
- High diagnosis rate
- Few false positives
- Robust to noise
- Near real-time turn-around

Data Collection

- Dataset from eBay Internet Service System
- Centralized Application Logging (CAL)
- Collect data log with CAL
  - Basic request trace
  - Extended database trace
- Huge logging throughput

Problem Descirption

- Classify the failed and successful requests
- Finding system components that are correlated with failure
- Post-process the paths that lead to failure-predicting nodes and extract relevant components

Decision Tree Learning Approach

- Human-interpretable results
- Easy for developer to fix error

Learning Decision Trees

- Maximize information Gain
- Fully grown and pruned back
- Stop splitting when the Gain falls below a certain threshold

Learning Decision Trees

- Different splitting criteria lead to different path selection
- C4.5
  - Use Entropy as pureness measurement

$$Gain(x_i, t) = H(t) - H(x_i, t)$$

- MinEntropy
  - Probability that a failed request at a particular node t takes on a particular value
  - Only follow the child node j with the highest failure probability (P(xi=j; t))

$$P(x_i = j; t) = \frac{\#\text{of failed requests at node } t \text{ with } x_i = j}{\#\text{offaiLed requests at node } t}$$

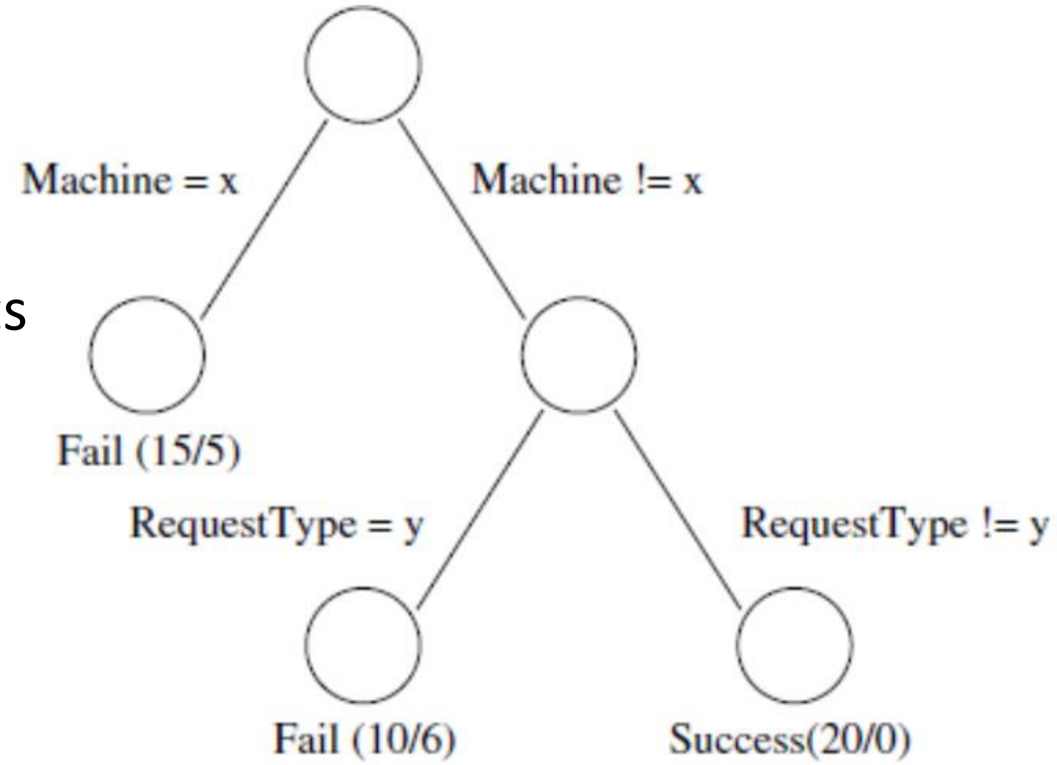
$$\text{and} \quad Gain(x_i, t) = -H(P(x_i; t))$$

Failure Diagnosis from Decision Tree Output

- 4 heuristics:
  - Ignore the leaf nodes for successful requests
  - Noise Filtering
  - Node Merging
  - Ranking

Failure Diagnosis from Decision Tree Output

- Learned Decision Tree structure
- Explanation 4 heuristics
  - Ignore the leaf nodes for successful requests
  - Noise Filtering
  - Node Merging
  - Ranking



Experimental Setup - Data Collection

- 10 one-hour snapshots of logs, each with system faults
- Totally 14 faults over 10 snapshots

Host	DB	Host, Host	Host, DB	Host, SW	DB, SW
2	4	1	1	1	1

- Complete request trace
  - Basic trace
  - Database trace

Type	Name	Pool	Machine	Version	Database	Status
10	300	15	260	7	40	8

Experimental Setup - Implementation

- MinEntropy with C++/Java
- C4.5 and association rules with Weka open-source ML package

Experimental Setup

- JDK 1.4.2 on quad-PIII 2GHz Linux 2.4.18 machines with 4GB of RAM

Results

- Compare with association rules
  - ranks all possible combinations of features according to their observed probability of request failure.
- Algorithm returns a set of candidates (combinations of system components), which are user-visible failures

Results

- Recall and Precision Metrics
- Recall measures the percentage of failure causes that are correctly diagnosed by the algorithm
- Precision measures how concise the candidate set is

Results

- **component** (Machine=x)
  → **candidate** (Machine=x and Version=n and RequestType=z)
  → **candidate set**
- Let C be **the number of effective components** in the candidate set,
  N be **number of distinct causes of failures**, and
  n be the **number of correctly identified failure causes**.
  We define:

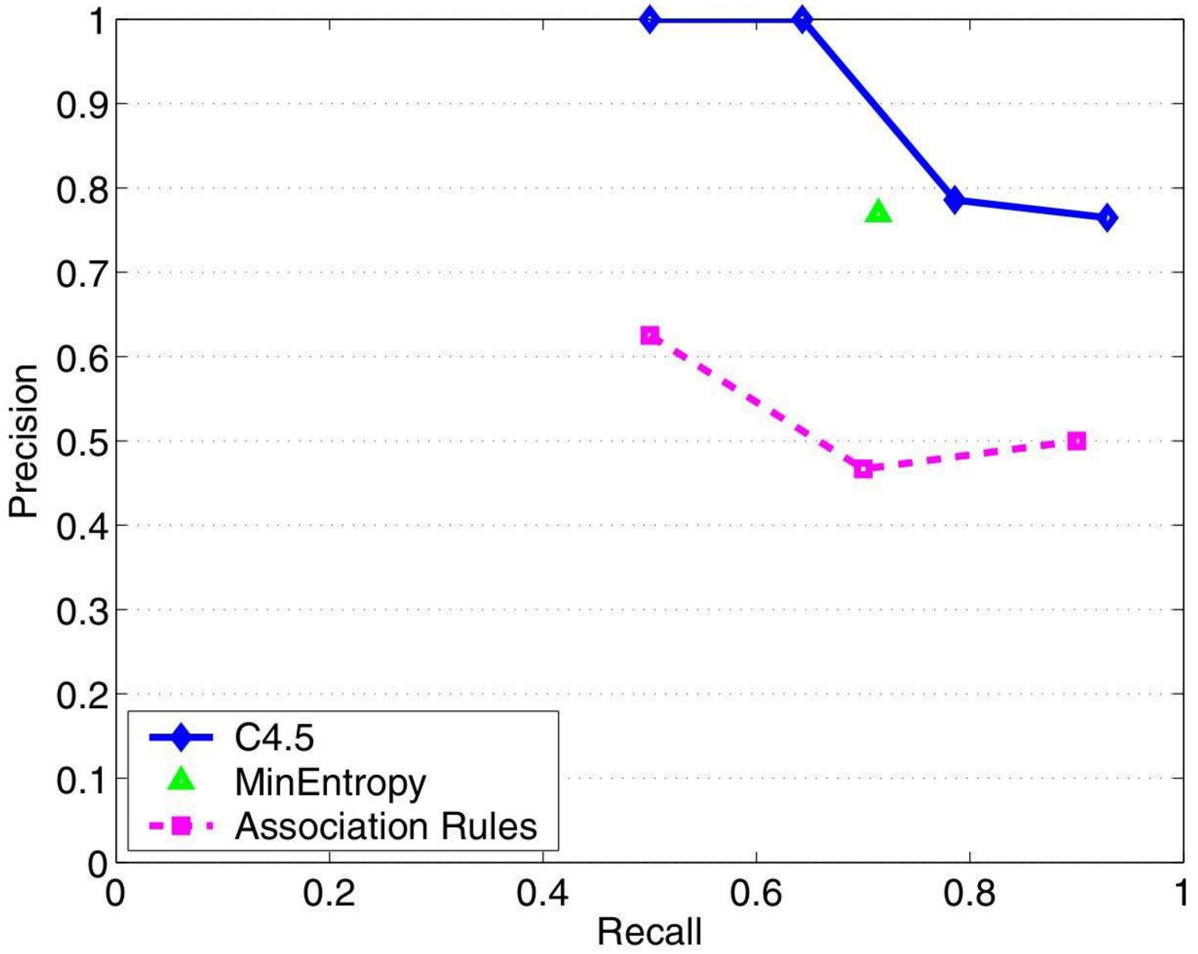
$$Recall = \frac{n}{N} \quad fpr = \frac{C - n}{C} \quad Precision = \frac{n}{C} = 1 - fpr$$

Results on Basic Request Traces

- Basic type of request trace is common to many Internet services
  - Record for performance monitoring, failure monitoring, and billing
- No database information, change cause from database faults to other faults

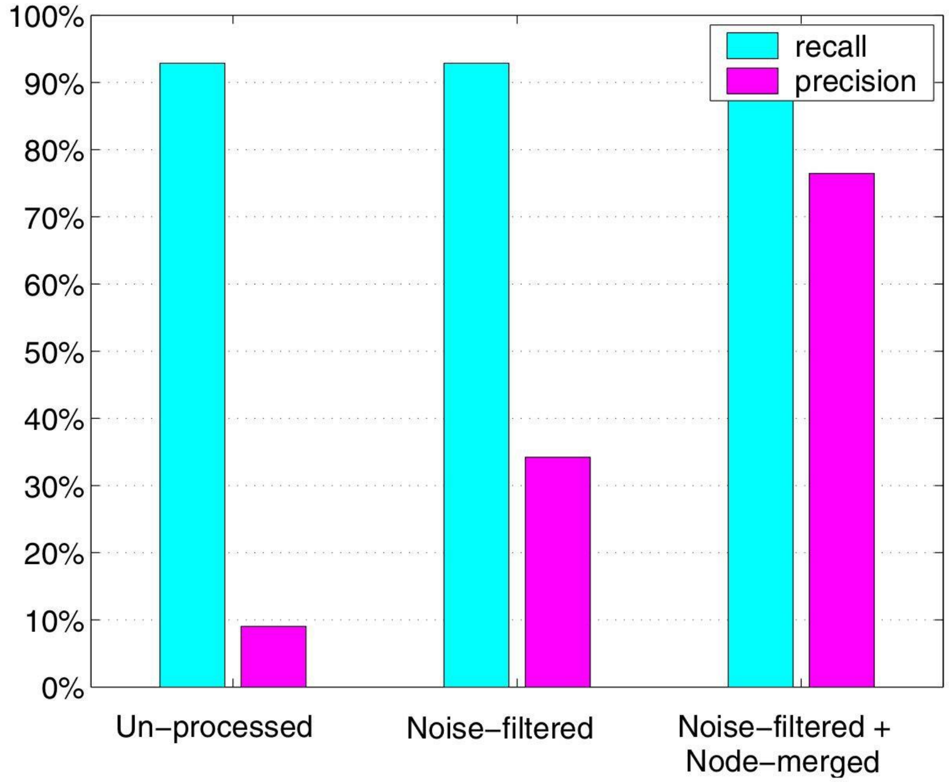
Results on Basic Request Traces

- C4.5, MinEntropy, association rules
- Iterating on failure rate cutoff threshold from 50% to 5%
  - Better recall
  - Worse precision



Experiments on Complete Traces

- Complete trace including basic and database traces
- Evaluate the 4 post-processing heuristics for Decision Tree output
  - C4.5 (no post-processing)
  - C4.5 + noise filtering
  - C4.5 + noise filtering + node merging
- Recall is all 13/14
- Precision is from 8% to 76%
  - False positive from 92% to 24%
  - 58% of components is essential for build tree but useless to indicate error, so removed

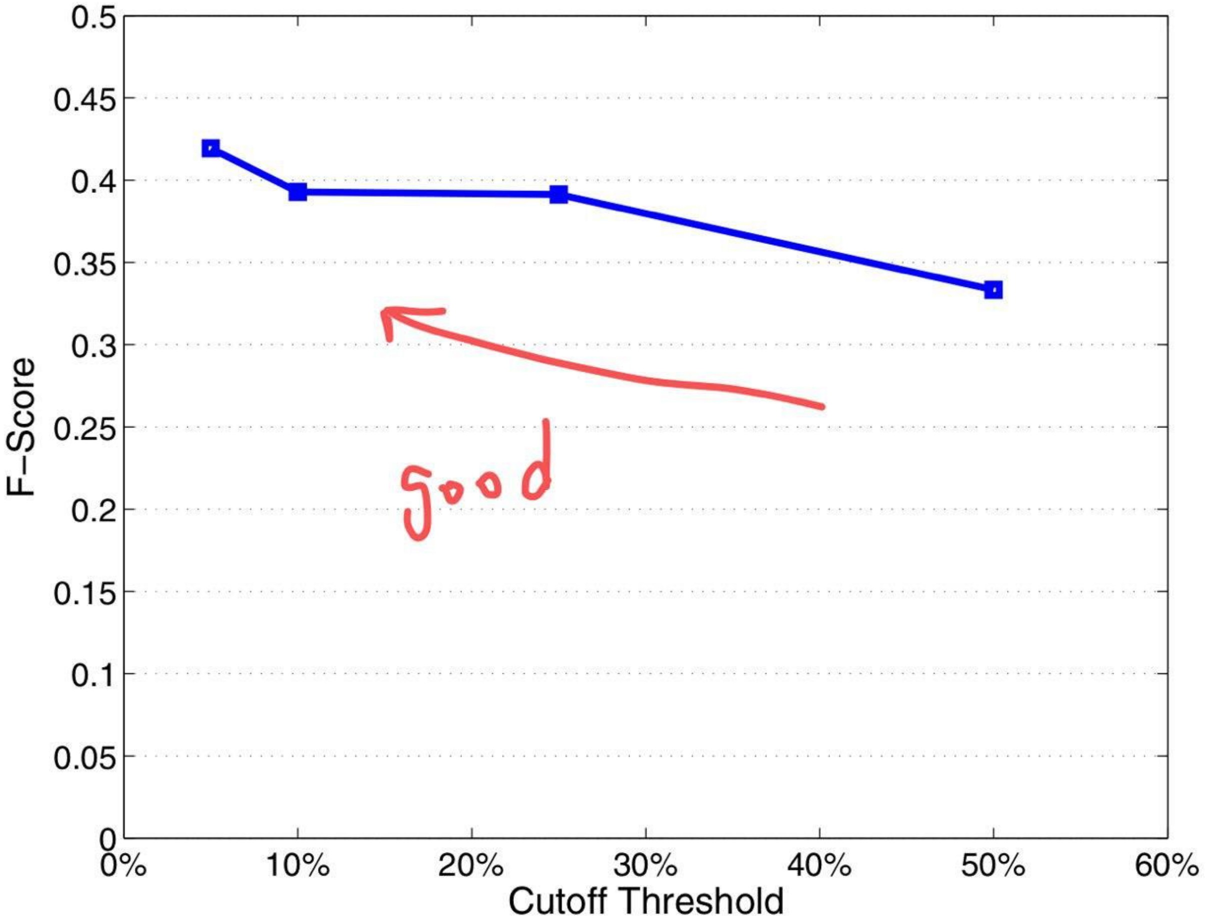


How Many Candidates to Keep

- If (# of failed requests / # of total failed requests) > (cutoff threshold c), then the path leading to node t is retained as a candidate
- Raising c would decrease the number of retained paths, vice versa
- Two approaches to select the threshold c:
  - Based on a metric that combines recall and precision (i.e., F-score)
  - Based on a metric that measures the expected recovery time.

F-score

- Plot the F-score against cutoff threshold
- From c = 5% with F-score = 0.4194 to c = 50%
    - Better precision
    - Worse recall
    - Worse F-score



Recovery time

- The time we need in 4 cases (Y as ground truth, Y^ as prediction):
  - Y=1, Y^=1: algorithm + recovery + verify
  - Y=1, Y^=0: algorithm + verify + manually examine + recovery
  - Y=0, Y^=1: algorithm + recovery + verify
  - Y=0, Y^=0: algorithm
- The estimated time:

$$
\begin{aligned}
\mathrm{E}[T] \quad = \quad & P_c(\hat{Y} = 1 | Y = 1) \cdot (a + r + v) \\
& + P_c(\hat{Y} = 0 | Y = 1) \cdot (a + v + m + r) \\
& + P_c(\hat{Y} = 1 | Y = 0) \cdot (a + r + v) \\
& + P_c(\hat{Y} = 0 | Y = 0) \cdot (a)
\end{aligned}
$$

Recovery time

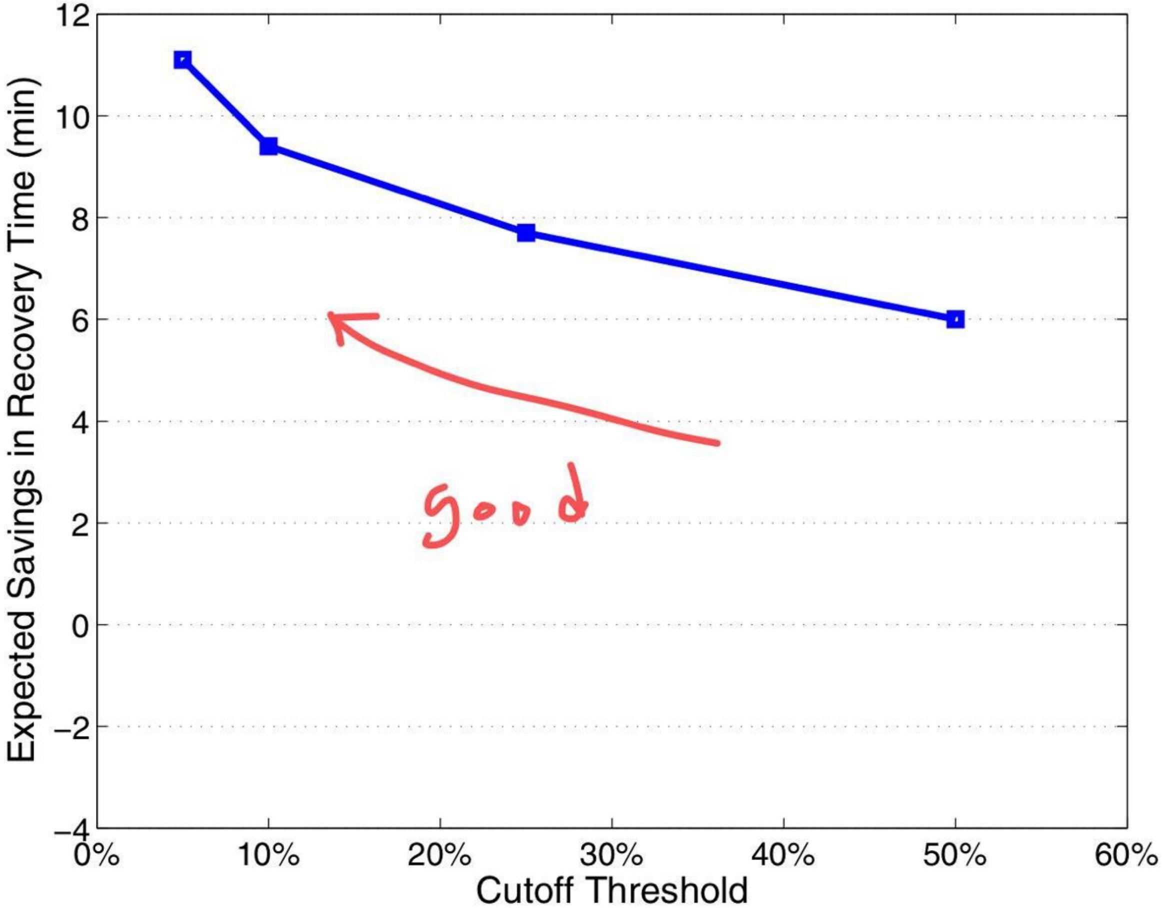
- The estimation of P in previous estimated time:

$$P_c(\hat{Y} = 1 | Y = 1) = Recall_c$$
$$P_c(\hat{Y} = 0 | Y = 1) = 1 - Recall_c$$
$$P_c(\hat{Y} = 1 | Y = 0) = \frac{\text{\# of false positives}}{\text{\# of non-faulty components}}$$
$$P_c(\hat{Y} = 0 | Y = 0) = \frac{\text{\# of true negatives}}{\text{\# of non-faulty components}}$$

- Saved time = Manually examine time + recovery time – E[t]

Recovery time

- Plot the saved time against cutoff threshold
- c = 5% saves 11.1 minutes over manual diagnosis
  - Less time-consuming case (ground truth = 1, prediction = 0)



Discussion and Related Work

- Decision tree is less performance but easier interpreting
- Other works do feature selection but without post-processing
- Most of the data are unlabeled data
  - Use failure diagnosis model to label those data
- The features we define may not contain the true cause of the faults
  - Nead to distinguish a node with useful information or just noises
  - Use failure distribution
- Other works on causal network model

Conclusion

- A new approach to diagnosing failures in large system
- MinEntropy for single-fault cases has been deployed at eBay
- C4.5 for both single-fault and multi-fault cases
- Exploring other learning algotirhm to improve the diagnosis performance
- Experimenting streaming versions to be deployed on production system
- Extending our approach to wide-area system failures

Pros & Cons

- Pros
  - Take advantages of decision tree on binary classification problem
  - Producing good methods to find useful information from the tree structure
  - Take rcovery time into consideration to get the crucial cause
- Cons
  - Performance of decision tree on classification is limited by the algorithm itself
  - Getting dataset from single website cannot ensure the scalability

Thank You

林律穎

2022/11/25