A Fast Decision Rule Engine for Anomaly Detection

James Thomas

Overview

- Introducing a classifier based on one- and two-feature decision rules as an interpretable approach for supervised anomaly detection
  - Practical due to fast implementation
- Pandas API and demo

Supervised Anomaly Detection Problems

- Binary classification with large class imbalance
  - Normal ML methods can struggle
- Want interpretability because humans often involved in addressing anomaly
  - Why was this classified as an anomaly?

Decision Rules for Categorical Tabular Data

Cellphone telemetry data:

OS Version	Manufacturer	Device Age	Region	Has error (class)
4.1	Samsung	1	US	1
4.1	Nokia	1	Europe	0
4.2	HTC	3	US	0
4.1	HTC	2	Asia	0
4.1	Nokia	1	Europe	1
4.3	HTC	1	Asia	0

Decision Rules for Categorical Tabular Data

Cellphone telemetry data:

OS Version	Manufacturer	Device Age	Region	Has error (class)
4.1	Samsung	1	US	1
4.1	Nokia	1	Europe	0
4.2	HTC	3	US	0
4.1	HTC	2	Asia	0
4.1	Nokia	1	Europe	1
4.3	HTC	1	Asia	0

Potential one-feature rule to select anomaly class:

OS Version = 4.1 (Precision 50%, 4 examples)

Decision Rules for Categorical Tabular Data

Cellphone telemetry data:

OS Version	Manufacturer	Device Age	Region	Has error (class)
4.1	Samsung	1	US	1
4.1	Nokia	1	Europe	0
4.2	HTC	3	US	0
4.1	HTC	2	Asia	0
4.1	Nokia	1	Europe	1
4.3	HTC	1	Asia	0

Potential one-feature rule to select anomaly class:

Manufacturer = Samsung (Precision 100%, 1 example)

Decision Rules for Categorical Tabular Data

Cellphone telemetry data:

OS Version	Manufacturer	Device Age	Region	Has error (class)
4.1	Samsung	1	US	1
4.1	Nokia	1	Europe	0
4.2	HTC	3	US	0
4.1	HTC	2	Asia	0
4.1	Nokia	1	Europe	1
4.3	HTC	1	Asia	0

Potential two-feature rule to select anomaly class:

OS Version = 4.1 && Device Age = 1 (Precision 66%, 3 examples)

Decision Rules Are Interpretable

- When limited to one or two features
- Even decision trees (especially deep ones or random forests) and linear models are hard to fully understand

Extending Decision Rules to Continuous Features

- Find min and max of continuous feature and discretize into equally sized buckets (15 buckets in our system)
- Other discretization schemes possible

Combining Decision Rules

- Single decision rule unlikely to be enough to classify well
- Create a classifier with many good decision rules; if any of them fires, anomaly is detected (logical OR of rules)
- Still interpretable – human can see which rule(s) fired for particular example

Evaluating Decision Rules: Counts

- Maintain count of anomalies and total examples for all one-feature and two-feature decision rules

Feature #1	Feature #1 Value	Feature #2	Feature # 2 Value	# Anomalies	# Total Examples	Precision
OS Version =	4.1	--	--	2	4	0.5
OS Version =	4.1	Region =	Asia	0	1	0.0
OS Version =	4.1	Region =	Europe	1	2	0.5

.
.
.

Computing Two-Feature Counts

- Gets expensive with large number of features (all pairs)
- We have a fast C++ implementation with experimental GPU/FPGA acceleration available
  - Can scale to the ~1000 feature range for large datasets

Selecting Decision Rules: Filter on Precision/Count

- Create a classifier with all decision rules having precision >= *p_thresh* and total examples >= *c_thresh*

Pruning Decision Rules from Classifier

- Overall classifier will likely have lower than *p_thresh* precision because there will be more overlap in the rules' correctly classified anomalies than in false positives
- Need way to prune redundant rules
  - Fewer rules also easier for human to process

Pruning Decision Rules from Classifier

- One heuristic: sort selected rules descending by anomaly count
- Iterate through rules and compute incremental precision (new true positives / new false positives) over previous rules
- Discard rules with incremental precision < *p_thresh’* and incremental classified examples < *c_thresh’*

Pruning Decision Rules from Classifier

- With *p_thresh'* very small and *c_thresh'* = 1, eliminates only rules that have no correct incremental classifications
- Strictly improves classifier precision with no change to recall
- Other heuristics possible…

Pandas API Summary

- `s = compute_sums(train_set, class_name)`
  - Compute all one-feature and two-feature counts
- `r = s.get_rules(p_thresh, c_thresh)`
  - Return dataframe of all rules with precision >= `p_thresh` and total training examples classified >= `c_thresh`

Pandas API Summary

- `r = s.prune(r, examples, p_thresh’, c_thresh’)`
  - Prune rules based on incremental performance; `examples` can just be the training set
- `s.display_rules(r)`
  - Display all rules in human-readable format
- `s.evaluate_summary(r, test_set)`
  - Return precision and recall of classifier consisting of all rules in `r`

Demo

- https://github.com/jjthomas/rule_engine
- jamesjoethomas@gmail.com