Activity: Time Series Analysis

Your goal in this activity is to analyze time-series data, find anomalous data points and visualize them. You will use a simple moving window approach for this purpose.

Starter: **ac5.zip** (under Piazza > Resources)
Files to submit: **ac5.R** (all TODOs implemented), **ac5.pdf** (single output graph)
Due: **Sunday, April 2** at 11:59PM-12:00UTC on T-Square
Binary Grading: Your submission will be evaluated using an autograder (see Evaluation section below for details). If you pass all the tests, you will receive a 1, otherwise you'll receive 0.
Extra Credit: This activity counts towards 1% of your course grade.

Getting Started

- Download and unzip **ac5.zip** from Piazza > Resources. It contains the following:
  - **ac5.R**: Code that you need to implement.
  - **test_ac5.R**: Unit tests for you to check your implementation.
  - **data/**: CSV files containing time series data.
  - **labeled_windows/**: Ground truth time windows representing anomalous events.
- Open ac5.R and start implementing the steps marked as **TODO**. Further instructions and hints are provided there as appropriate. Some steps maybe marked "optional" - you need not implement them.
- Note: You only need to make changes _inside_ the following three functions:
  - `load_ts()`: Load and return time series data from a CSV file.
  - `find_anomalies()`: Find anomalous data points in a time series.
  - `visualize()`: Visualize the results of anomaly detection.
- Please do not change the parameters of the function as they will be called by an autograder.
- Run ac5.R as you develop, either from RStudio or using `source("ac5.R")` from an R interpreter.

Main Block

There is a block of code at the end of ac5.R that begins with the following line:

```
if(getOption("run.main", default=TRUE)) {
    ...
}
```

This is what gets executed when you source the script directly, but it will be suppressed when unit tests or the autograder are run. You are free to change anything within this block, e.g. trying out different input data/parameters, printing debug messages, etc. But please don't add any code at the top level outside this block. When running unit tests and the autograder, we don't want arbitrary code to be executed.

Testing

A set of unit tests are provided in test_ac5.R, which you can run from RStudio, or using `source("test_ac5.R")` from the R interpreter, or `Rscript test_ac5.R` from a terminal. If your code passes all the tests, you won't see any outputs. Otherwise an appropriate message will indicate what test you failed, or if there is any other error in your code.

Only the two functions `load_ts()` and `find_anomalies()` are tested, first to check for proper return values, and then for correctly-detected anomalies using an approximate matching criteria:

- Out of the total number of anomalies reported (num_reported), we count how many lie within a ground truth window (num_hits).
- Out of these hits, we see how many are unique for a window (num_unique) and how many are duplicates (num_dupes). An ideal detector would result in 1 unique hit per window and 0 duplicates.
- We also count how many false positives are reported (num_false), i.e. detections that are outside any of the ground truth windows.
- Using these values, we check that:
  - Accuracy (num_unique / no. of actual ground truth windows) is at least 75%.

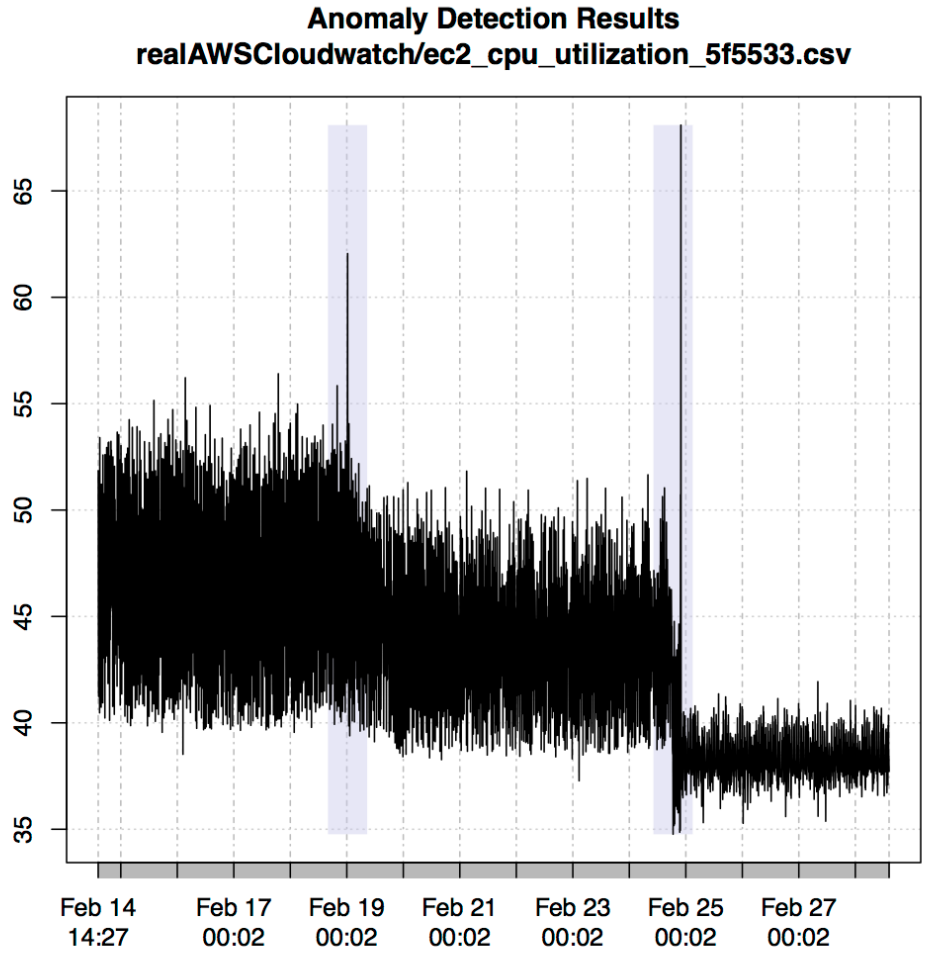
- False positive rate (num_false / num_reported) is at most 50%.
- Duplicate rate (num_dupes / num_reported) is at most 75%.

Note that these are explicit interpretations of the terms "accuracy", "false positive rate" and "duplicate rate" that we have adopted for the purpose of this activity.

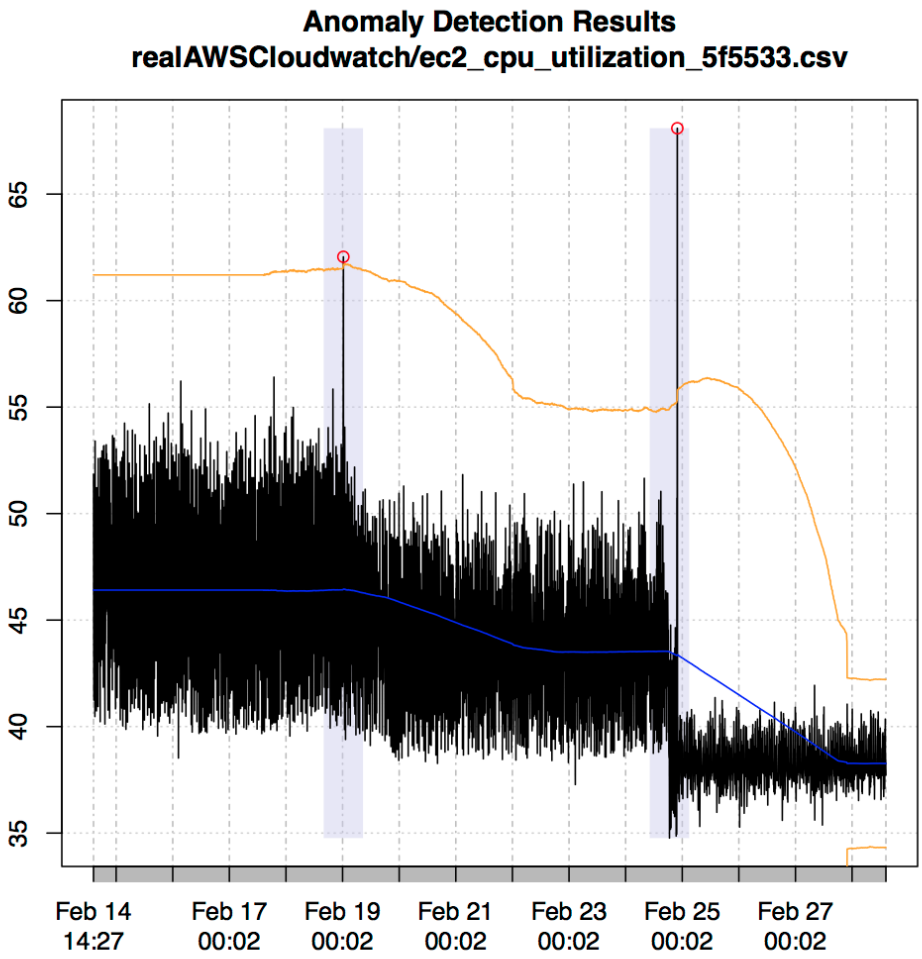
Also note that one of the testing criteria is to check that `load_ts()` and `find_anomalies()` do not print anything to the console or generate any other output. They should only return the appropriate object(s) as listed in their respective function documentation. Please ensure this is the case before running unit tests or submitting your code.

Visualization

Visualizing time-series data can be tricky, especially when you may have irregularly-sampled data points. Fortunately, R makes it very easy to plot time series data. The `visualize()` function included in the starter code should produce a plot like this on a valid time series:



Follow the instructions in `visualize()` and implement the steps marked TODO. The resulting plot should look somewhat like:



Note that the `visualize()` function is not called for testing or evaluated for grading, but you must still implement it and include a single output graph (on any of the input files of your choice) as ac5.pdf, as part of your submission. You may implement it as you wish, e.g. with ggplot2 instead of base graphics, and improve the resulting plot, e.g. shade the portion between upper and lower margins (don't change the function parameters, though).

Evaluation

Your code must pass all tests in test_ac5.R as described above, producing no output (except any warnings when loading packages). An autograding script will essentially run test_ac5.R and assign you a 1 if you pass, 0 otherwise.

In addition, we will randomly check for ac5.pdf, so please include it. If it is missing or doesn't have any of the following items clearly shown, you will get a 0:

- Input time series with anomaly windows highlighted (already included in starter code).
- Rolling mean and margins (above and below the mean, with optional shading).
- Anomalous data points.

Extensions

If you found this activity interesting, there are several ways of improving or extending on it. Anomaly detection is hard. Here we have explored only a very simple approach to the problem, and it is only capable of detecting spike anomalies - i.e. data points that far from some moving average. What about temporal anomalies, where the pattern of the data changes for a period of time? Something like that is not easy to detecting looking at individual samples; instead, you have to look at the statistical distribution over a window of time and detect if the *distribution* changes significantly.

Another challenge is to tune your detection algorithm to favor more coverage vs reduce false positives. And this depends on the domain you are applying your detection algorithm to - for instance, in a medical diagnosis scenario, you would probably want to detect as many anomalies as you can even if it means allowing some false positives to creep in; however, in a lower risk scenario like server health monitoring, you may want to lower the number of false alarms generated.

Resources

- The data for this activity has been borrowed from the Numenta Anomaly Benchmark (NAB). You will find additional approaches, implementations and a more refined evaluation criteria there.
- If you are interested in time series analysis, take a look at the following:
  - https://cran.r-project.org/web/views/TimeSeries.html
  - http://www.stats.uwo.ca/faculty/aim/tsar/tsar.pdf
  - https://blog.rstudio.org/2015/04/14/interactive-time-series-with-dygraphs/

Once you have completed and submitted this activity, feel free to explore the topic further, improve or completely change your detection algorithm, or even apply it to a real-life scenario of your choosing and share your findings with the class below!

#pin