Technical details and implementation of the ALERT algorithm

Nicholas G Reich, Stephen A Lauer et al.
(full author list TBD)

January 27, 2014

1 Introduction to the ALERT algorithm

In this document we examine and describe the utility of the "Above Local Elevated Respiratory illness Threshold" (ALERT) algorithm in prospectively determining the start and end to a period of elevated influenza incidence in a community. This algorithm could provide a valuable tool to communities, schools, hospitals and other institutions looking for a simple method to objectively define a period when, for example, enhanced patient contact precautions, increased office visits, or other prevention measures should be implemented. The ALERT algorithm is a simple metric that can be simply operationalized and is not meant to be a sophisticated prediction model for influenza.

The ALERT algorithm uses current and prior flu season information to determine an ALERT threshold. The ALERT period begins when the reported number of laboratory-confirmed cases for a given week exceeds the established ALERT threshold. This serves as the flu season trigger, signaling larger than expected fluctuations. To account for reporting delays and possible delays in implementation of any policies, the user may specify a lag period: a number of days between the reporting date associated with the trigger and the date the ALERT period should be put into effect. The ALERT period ends when the reported number of cases falls below the same threshold, after a minimum passage of eight weeks. (This grace period is chosen by default to be eight weeks but its duration can be modified by the user.)

The ALERT algorithm can be implemented either via the ALERT R package, or an Excel spreadsheet. The R package contains some features that the spreadsheet does not. Namely, computation of a more complete set of metrics for the possible thresholds and the cross-validation procedure that can be used to evaluate different decision rules for choosing a threshold.

This document describes the ALERT algorithm in detail and provides example R code and output.

Ways to use the ALERT algorithm in practice

There are two ways one can use the ALERT algorithm to determine a threshold in practice.

- **The simpleALERT algorithm** This first method is simple, straightforward, and can be accomplished using the Excel spreadsheet or the R package. To choose a threshold using this method, you use one of the ALERT tools to create a table of historical performance of ALERT thresholds. You then choose the threshold that appears to perform the best, as measured by the calculated performance metrics. This method provides simple way to obtain a threshold, but does not provide a statistically robust assessment of future performance.

- **The robustALERT algorithm** This second method requires one to use the ALERT R package and to pre-specify a set of decision rules any of which could choose an acceptable threshold. The rules are evaluated prospectively via a leave-one-season-out cross-validation algorithm. The rule that has optimal performance is chosen. Then the historical performance of all thresholds are computed and the optimal rule is applied to choose a threshold.

2 Methodological details for the ALERT algorithm

The ALERT algorithm performs two distinct tasks. It can

1. calculate the historical performance of possible ALERT thresholds, thereby providing the information needed to choose an appropriate threshold, and

2. evaluate the prospective performance of the ALERT algorithm under one or more threshold decision rules.

Each of these steps is discussed in more detail in the subsequent sections.

Calculating historical performance of possible ALERT thresholds

To define the ALERT period, we use past surveillance data to evaluate the retrospective performance of possible thresholds. The ALERT algorithm defaults to choosing potential thresholds as the 10th, 20th, 30th, 40th, and 50th percentiles of all of the non-zero historical weekly case counts. In the R implementation of the ALERT algorithm, this can be specified as all integer thresholds between the 10th and 50th percentile.

For each threshold considered, the ALERT algorithm summarizes data from previous years as if that threshold had been applied. Say that we have historical data on N seasons. Let $X_{i,t}$ be the percentage of cases captured in the ALERT period for season i ($i = 1, ..., N$) and threshold t. Let $D_{i,t}$ be the duration of the ALERT period for season i and threshold t. For each threshold considered, the ALERT algorithm calculates and reports the following metrics:

1. Across all seasons, the average percentage of all influenza cases contained within the ALERT period, $\bar{X}_t = \frac{\sum_i X_{i,t}}{N}$.

2. The minimum, maximum, and sd of the percentage of all influenza cases contained within the ALERT period.

3. The average ALERT period duration, $\bar{D}_t = \frac{\sum_i D_{i,t}}{N}$.

4. The fraction of seasons in which the ALERT period contained the peak week.

5. The fraction of seasons in which the ALERT period contained the peak week $+/-k$ weeks (k is specified by the user).

6. The average number of weeks included in the ALERT period with counts less than the threshold.

7. The average difference between, for each season, the duration of the ALERT period and the duration of the shortest period needed to capture P percent of cases for that season. (This metric requires a bit more computation time, and is only computed if the user specifies a P.)

[[Describe R function `createALERT()`.]]

Looking at the historical performance metrics can provde a useful snapshot of the performance of different thresholds. In many settings, this may provide enough information to choose a threshold for use in the future. However, on its own, evaluating the past performance of these thresholds is not a statistically robust way to predict the future performance of a given threshold.

Evaluating the prospective performance of the ALERT algorithm

In practice, it is difficult to automate the selection of an optimal threshold, as the factors that determine an 'optimal' threshold vary by setting. For example, in one setting, limiting the expected duration of the ALERT period may be very important to keep costs down. In another setting, duration may not play a factor and it is more important to capture a specific fraction of all cases each year.

Therefore, it is vital for users of the robustALERT algorithm to state clearly what criteria they will use to choose a rule. Here are some examples:

- *We are interested in the highest threshold that has historically captured on average over 85% of cases.*
  Using Table 1, this would suggest a threshold of 6 cases.

- *We want the lowest threshold that has had an average duration of less than 12 weeks.*
  Using Table 1, this would suggest a threshold of 5 cases.

- *We would like a threshold that has historically captured the peak and the two weeks on either side at least 80% of the time.*
  Using Table 1, this would suggest a threshold of 3 cases.

Given a decision rule such as any of the above, the ALERT algorithm will be able to find an optimal rule. In absence of such a decision rule, a user would make a subjective decision about the best threshold, based on the results shown. Additionally, given a set of decision rules,

the robustALERT algorithm can provide statistically cross-validated measures of expected future performance of thresholds chosen using a particular rule.

To evaluate a particular rule, the ALERT algorithm performs the `evalALERTRule` routine, which conducts the following steps for each season i:

- Create a dataset that includes all seasons except season i. We will refer to the included seasons as the "training seasons".
- Run the `createALERT()` function to calculate the performance of potential thresholds across the 'training seasons.'
- Choose the best threshold t based on the rule provided.
- Run the `applyALERT()` function using season i and threshold t.
- Save the ALERT performance metrics for season i.

To run the robustALERT algorithm, one follows the following steps:

- For each rule considered, run `evalALERTRule`.
- Choose the rule that has the best performance.
- Use the `createALERT` routine to determine historical performance of a range of thresholds.
- Apply the chosen rule to the output from `createALERT` to choose a threshold to use prospectively.

3 An implementation of the simpleALERT algorithm

The following code loads the code to run ALERT, loads a test set of data, and computes the ALERT algorithm metrics on historical data.

```
source("../R/ALERT.R")
load("../R/ALERT_test_data.rda")
```

```
tmp <- createALERT(subset(data, Date < as.Date("2011-08-14")), allThresholds = TRUE,
    k = 2, firstMonth = 8, target.pct = 0.85)
out <- as.data.frame(tmp$out)
```

Table 1 shows slightly reformatted results from the `tmp$out` object, showing the historical performance of different thresholds.

[[Add code showing application of a threshold to the two left-out years.]]

Table 1: Printed table of the tmp$out object.

		% of cases captured				% captured			
threshold	avg dur	mean	min	max	sd	peaks	peaks+/-k	low weeks	diff
1.0	18.9	96.0	72.6	99.4	8.3	100.0	90.0	2.3	8.6
2.0	15.3	93.8	69.9	98.3	8.7	100.0	90.0	1.2	5.0
3.0	13.1	91.3	68.9	97.8	8.6	100.0	90.0	1.2	2.8
4.0	12.4	89.5	68.9	96.6	8.2	100.0	70.0	1.3	2.1
5.0	11.8	87.8	68.9	96.1	8.6	100.0	70.0	1.1	1.5
6.0	11.3	85.7	68.9	96.1	10.7	100.0	70.0	1.3	1.0
7.0	10.2	82.6	60.7	96.1	12.2	100.0	60.0	1.4	-0.1

4 An implementation of the robustALERT algorithm

```
tmp2 <- robustALERT(subset(data, Date < as.Date("2011-08-14")), allThresholds = TRUE,
    k = 2, firstMonth = 8, lag = 7, minWeeks = 8, minPercent = c(0.8, 0.85,
        0.9), maxDuration = c(12, 13, 14))
```

Table 2 shows slightly reformatted results from the `tmp2` object, showing the mean cross-validated results of different rules.

Table 2: Printed table of the tmp2 object.

rule	thresh	total cases	dur	ALERT cases	%	peaks	peaks+/-k	low weeks	diff
minPercent = 0.8	7.1	429.6	10.2	384.5	82.0	100.0	50.0	1.6	0.7
minPercent = 0.85	7.1	429.6	10.2	384.5	82.0	100.0	50.0	1.6	-0.1
minPercent = 0.9	7.1	429.6	10.2	384.5	82.0	100.0	50.0	1.6	-1.6
maxDuration = 12	5.2	429.6	11.9	397.3	88.0	100.0	70.0	1.1	
maxDuration = 13	3.6	429.6	12.6	400.8	90.0	100.0	80.0	1.2	
maxDuration = 14	3	429.6	13.1	404.1	91.0	100.0	90.0	1.2	