SCOPE Model Case Study

An analysis of past, present, and potential future models.

Stefanie Molin

June 8, 2018

Contents

Introduction

SCOPE is a suite of anomaly detection tools for performance- and technical-based metrics. Our principal offering is daily alerts sent to the employees running the affected account(s) for further investigation. SCOPE has been using rules-based approaches for anomaly detection since its inception, but I want to change that—I want to make SCOPE smarter.

How can we know if a new approach is indeed better than the current? We need to establish a set of criteria for comparing models and a baseline for evaluating the performance of various candidate models. In order to accomplish this, we need to understand how models—past, present, and potential future—work, perform, and compare. This case study will document the models, how they work, and their performance on the same data sets. I will also be looking for any strengths, weaknesses, or patterns that I notice with each model's output. Ultimately, I will evaluate them against a set of criteria—and each other—to determine the next model for SCOPE.

The Data

We can't do an analysis without data, so we are going to use the data we have been collecting with SCOPE to perform this case study. In order to get an accurate look at how the models perform, we are going to use data from several days for this analysis: some from the beginning of the month, some from the middle, and some from the end. Additional dates have been pulled for the territory RexT data, since it is comparatively sparse. Each series must have at least 25 days of data in the last 30 days and TAC greater than 0 for the last 2 days (for clients); this leaves us with 25-60 days per series (30 for territory RexT series).

Collection of Labelled Data: Metis

The SCOPE team and I added feedback buttons to the SCOPE emails; these only give us feedback on what the model flagged as positive (series flagged as negatives are never sent in the emails). However, in order to accurately measure our models, we need user input on what we flag as negative since we can be wrong. To make this possible, I built a web app called Metis (Figure 1) which shows users time series (KPI evolutions) and gives them the option to classify the last point in the series as non-anomalous (dare I say normal?) or something worthy of further inspection/notification (an alert).

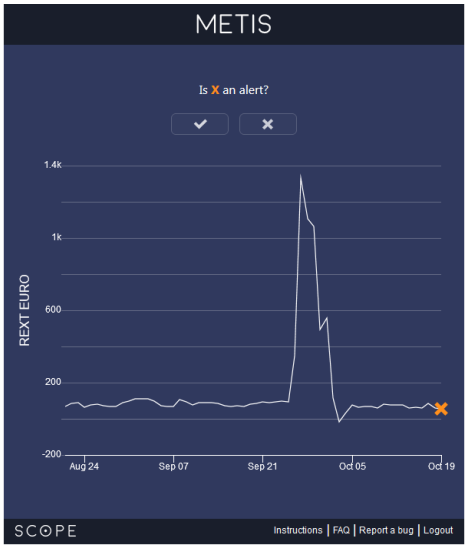


Figure 1: Collection of labelled data through Metis

This data allows us to compare how users classified time series to how the models classifed them; we can see when they: agreed it was an alert (true positive); agreed it was not an alert (true negative); disagreed because the users said it was an alert but the model didn't (false negative), or because the model flagged it but the users didn't (false positive). More on this later in the Evaluation Criteria section. Although the feedback from the buttons in the emails is collected with `Metis`, only data points collected through use of the app itself will be used for this case study; this is because the dates collected via the feedback buttons don't overlap with the dates used for this study and the email responses are from users with additional information—we don't want to introduce an information bias here.

I had users from various teams (AS, AX, and TS) provide data for this case study using `Metis`; a user's team determines which KPIs they can classify. Users who receive `SCOPE` alerts for certain metrics would be the ones classifying those, since we are trying to collect user opinions; for example, TS are the only group that can classify site events, but they can't classify spend which is done by AS. `Metis` has all the same data we are using for the case study, but, since it wouldn't be possible to have all users classify everything quickly, `Metis` first randomly (to avoid potentially introducing a selection bias) shows all series that have *never* been classified. This way everything will get classified once before others get a second classification. Note that this is by user group: user groups with less KPIs to classify will finish sooner and start giving second and third classifications before another group finishes its first round. The only restriction we put on this process is that a user can only classify a series once; we want multiple opinions on each time series, so we can average the responses, just not from the same user. This also means that, since we don't have everything classified, the count of alerts triggered by each metric will be higher than the count of alerts flagged by the model and also in `Metis`; we will only be able to test the model against real world opinion for a subset of all the KPI evolutions—a random sample of them.

This is going to be a very subjective task by nature (even though we show them instructions of how to respond correctly) because everyone sees things differently, and they all have different ways of managing an account; in fact, as I was conducting user sessions, I noticed how each user approached the task differently: some were trying to gauge how much the series changed in the last day, some looked for seasonality that could explain the change, others looked for deviation from a central tendency, a few even flagged metrics that had flat-lined for the full 60 day range shown in the graph saying that something probably wasn't set up properly, and therefore, they would want to know.

After these initial 1-1 user sessions, when `Metis` was ready for multiple users, I conducted user groups where we spent the first half using `Metis` and the second half discussing what they were looking for when they classified series. Here, I will provide the results by group:

- AS
  - Dates
    - day of week is useful in classifying; vertical not so much
    - look at quarter to date (for goaling), last 30 days, last 2 weeks, last week, and yesterday so these date ranges are important
    - time of month; last 2 weeks and day before are common viewpoints
    - if it is going crazy, then check back the full date range
  - Patterns
    - volatility throughout (steady in last week = no alert)
    - huge relative change/departure from pattern
    - fluctuations outside the pattern
  - Values
    - magnitude of change over days past
    - straight lines at 0 are irrelevant because they would know before then
- AX
  - Dates
    - 7-14 days is best (60 days is often noise); show only 2 weeks or 30 day max
    - number of points is more important than the dates themselves
    - with long date range: patterns that were abnormal and persistent for a minimum of a few days
    - look at duration of anomaly

        * past 3 days to see if alert was redundant, full timeline to see if within long-term normal range.
        * good to confirm drops/spikes even if they are expected end of month (not too worried about seasonality)
    - Patterns
        * increasing or decreasing outside what has been normal in the past
        * record low/high and patterns (check if fluctuates a lot)
        * when to disqualify an alert for noise
    - Values
        * if there was an increase/decrease that was strong and outside of the boundary (+/- a few percentage points) of the last few days week of data
        * if the data came to zero
        * magnitude of the drop, not just percentage (15 to 5 isn't as worrisome as 150K to 50K)
        * if yesterday's value is negative even if the change isn't large or in a range it should never be (yesterday is in wrong spot – i.e. margin shouldn't be negative ever)
        * looked at closest days to last point and check if it is an anomaly compared to recent data points
- TS
    - Dates
        * length of time at a certain range and then a change
        * weekends/seasonality; end, middle, beginning of month
    - Patterns
        * pattern across whole time span and seasonality/volatility
        * change in variation between the steps
        * in-line changes that are large in magnitude but might be normal
        * slow decline for events in general – no alert; this should be brought to TS attention by the AS
        * InApp tags pattern will be a slow decline if there was a bad release as more and more people update the app
    - Values
        * large percentage drops on small magnitudes is not too important (i.e. from 15 to 5 is not but 100k to 50k is)
        * lowest or highest ever seen and still increasing/decreasing in same direction
        * last point higher or lower than avg and max/min
        * magnitude of change
        * triggered more alerts on decrease (increase needs to be more egregious)

Each group definitely has their intricacies, but they are all looking for departures from patterns and paying attention to the magnitude of the changes—not just the percentage.

Dates Used for Analysis

Since `SCOPE` is being pulled to look at *yesterday's* data, the run dates (the day `SCOPE` runs, i.e. "today") below will be 1 day after the date the models will look for alerts on. Therefore, a date of November 1st is actually end of the month. Table 1 contains the dates for this analysis. *Entries marked with * are RexT only.*

Table 1: Dates for analysis

Beginning of Month	Middle of Month	End of Month
2017-11-02	2017-10-20	2017-10-25*
2017-11-03*	2017-11-13*	2017-10-26*
2017-11-06*	2017-11-14*	2017-10-27*
2017-11-07*	2017-11-15*	2017-10-30*
2017-11-08*	2017-11-16*	2017-10-31*
2017-11-09*	2017-11-17*	2017-11-01

Structure and Content

I am capturing data across several dimensions for AS, TS, and RexT-based alerts on clicks, conversions, COS, CR, CTR, displays, margin, order value, RexT, RexT Euro, RexT local, site events, spend, TAC, tag events, and territory RexT. Below you will find an example of how each data set looks, including the log data we are collecting from `Metis`. I am pulling in more dimensions than we currently use in order to evaluate their potential utility in designing a new model. If they prove to have no added value when performing this case study, they will be removed from `SCOPE`'s data collection.

Note that ***`run_date`*** *is the date the alerts would have been run while* ***`day`*** *is the day the data is for.*

Preview of AS data

For the AS data (Table 2), I am pulling campaign- and client-level stats by day including dimensions related to campaign setup like bidding strategy, campaign type, campaign funnel, geo, and vertical.

Table 2: Snapshot of AS data

day	client_name	client_id	global_account_name	RTC_vertical
2017-10-31	[redacted]			
2017-10-31				
2017-10-31				
2017-10-31				
2017-10-31				

vertical_name	campaign_scenario	campaign_id	campaign_name
[redacted]			

campaign_type_name	campaign_revenue_type	campaign_funnel_id	cost_center	ranking
[redacted]				

country	subregion	region	run_date	kpi	value
US	NORTH AMERICA	AMERICAS	2017-11-01	clicks	[redacted]
US	NORTH AMERICA	AMERICAS	2017-11-01	clicks	
US	NORTH AMERICA	AMERICAS	2017-11-01	displays	
US	NORTH AMERICA	AMERICAS	2017-11-01	displays	
US	NORTH AMERICA	AMERICAS	2017-11-01	conversions	

Preview of TS data

For the TS data, I am pulling tag- and site-level stats by day. I capture both `site_type` and `event_name` as well as geo and vertical data (Table 6).

Table 6: Snapshot of TS data

global_account_name	RTC_vertical	vertical_name	partner_id	partner_name
[redacted]				

cost_center	ranking	country	subregion	region	site_type
[redacted]					

event_name	day	run_date	kpi	value
[redacted]	2017-11-01	2017-11-02	site_events	[redacted]
	2017-11-01	2017-11-02	tag_events	
	2017-11-01	2017-11-02	tag_events	
	2017-11-01	2017-11-02	tag_events	
	2017-11-01	2017-11-02	tag_events	

Preview of RexT data

Here, I am only collecting territory RexT data by day where the territory gets as granular as `country` (i.e. US) and as broad as `region` (i.e. Americas). Some sample rows are provided in Table 9.

Table 9: Snapshot of RexT data

country	subregion	region	ranking	day	run_date
[redacted]				2017-10-20	2017-11-01
				2017-10-20	2017-11-01
				2017-10-20	2017-11-01
				2017-10-20	2017-11-01

kpi	value
territory_rext	[redacted]
territory_rext	
territory_rext	
territory_rext	

Preview of Log data

This data set is different than the rest. We aren't recording time series here, but rather, logging all classifications that are collected with `Metis` (Table 11). Here, I will only be focusing on input that came from the web app (not looking at the email feedback buttons, since our data is pre-feedback buttons).

Table 11: Snapshot of logs

source	username	datetime_utc	client_id	partner_id	campaign_id	cost_center	ranking
web_app	s.molin	2017-11-07 15:49:00	[redacted]				
web_app	s.molin	2017-11-07 15:49:00					
web_app	s.molin	2017-11-07 15:49:00					
web_app	s.molin	2017-11-07 15:49:00					
web_app	s.molin	2017-11-07 15:49:00					

country	subregion	region	site_type	event_name	run_date	kpi
[redacted]					2017-11-01	ctr
					2017-10-20	client_rext
					2017-10-20	cos
					2017-10-20	displays
					2017-11-02	cos

series	is_alert
[redacted]	FALSE
	FALSE
	FALSE
	FALSE
	FALSE

Note that most of these entries will have is_alert = FALSE. This is expected because anomalies shouldn't be expected to happen most of the time.

Data Prep

The aforementioned data snapshots are raw data. We need to prep and clean it before we can use it in the models. To do so, we have a 3-step process.

1. Disqualify any series with less than 25 rows of data or traffic aquisition costs of 0 for 2 consecutive days. *(may not be used for all models)*
2. Replace all non-graphable values (`NaN`, `Inf`, `x/0`, etc.) with 0.
3. Fill in any dates missing between the minimum date of the series and the maximum date of all series with the same run date with either:
    a. the median values of each metric, if the data is missing across all series (meaning it isn't in the database due to some unknown error)
    b. 0, if the date isn't missing globally (meaning the client had no data recorded for that day).

Note that this process might not be the most optimal, but I will use this as the initial cleaning for all models I test, unless otherwise written, to be consistent.

Evaluation Criteria

The goal of this case study is to find the model that best identifies anomalies as a human would. However, we are only able to "predict" if something is an anomaly and that can be different to what it is in the real world, as illustrated in Figure 2. We will also need to decide if reducing false positives or false negatives is more important to us, as we may be faced with variations of the same model that reduce one or the other.

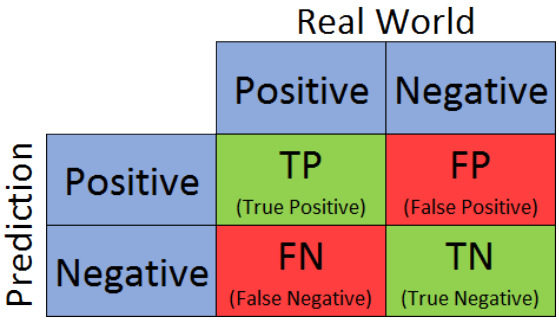


Figure 2: Prediction vs. Real World
Note the above abbreviations will be used throughout this paper

In an effort to improve our performance in this task, each model will be evaluated as follows:

1. Number of alerts triggered for date in question and by metric.
2. Combinations of alerts triggered (i.e. how are alerts distributed over clients?).
3. Comparison to data collected and classified with `Metis`. In cases where the model is looking at the series holistically, if any of the metrics are alerts, the series should be counted as such. We will be using the following metrics to evaluate how well each model matches up with user opinion:
    - **Precision** is the probability that a (randomly selected) retrieved document is relevant. It can also be thought of as the average probability of relevant retrieval.[1] Note that this is the only metric we were able to calculate without the data from `Metis` since we would only be looking at what the models flagged as alerts, however, it is much easier to collect the data through `Metis`.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

    - **Recall** is the probability that a (randomly selected) relevant document is retrieved in a search (also called `sensitivity` or true positive rate). Alternatively, it is the average probability of complete retrieval, meaning it measures the proportion of positives that are correctly identified as such.[2,3] Note that without the data from `Metis`, we were unable to calculate this metric because we had no insight into what the models marked as not an alert.

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

    - **F1-score** (also F-score or F-measure) is a measure of a test's accuracy in statistical analysis of binary classification. It considers both the `precision` and the `recall` and is the harmonic average of the `precision` and `recall`, where an `F1-score` reaches its best value at 1 (perfect `precision` and `recall`) and worst at 0.[4] We will treat this as our main metric.

$$F1\_score = 2 * \frac{precision * recall}{precision + recall} \tag{3}$$

[1] Precision and recall
[2] Precision and recall
[3] Sensitivity and specificity
[4] F1 Score

- **Specificity** (also called the true negative rate) measures the proportion of negatives that are correctly identified as such (e.g. the percentage of healthy people who are correctly identified as not having the condition).[5] Since the task of anomaly detection should result in very few anomalies, we should expect to have high `specificity` for everything because most series will be true negatives.

$$Specificity = \frac{TN}{TN + FP} \tag{4}$$

- **Accuracy** is a combination of both types of observational error (random and systematic), so high accuracy requires both high `precision` and high trueness.[6]

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{5}$$

- **ROC Curve** (receiver operating characteristic curve), is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings (see Figure 3).[7] Therefore, when our model doesn't yield us with a probability of alert, we won't have a threshold to vary; in these cases, we will simply plot the specific observations (time of month and by metric) to see how they perform compared to randomly triggering alerts (45-degree line). We strive to maximize the area under the ROC Curve (AUC), meaning that the best model will be the one with a ROC curve that gets closest to the top left corner (100% TPR, 0% FPR).

4. **Algorithm Complexity**. Big-O notation is used to classify algorithms according to how their running time or space requirements grow as the input size grows.[8] I will use this to give an idea of how efficient each of the models is. This is important because if a model takes a long time to run or calculate but is only a marginal improvement over another, it may not be the choice for scalability reasons. Where possible, I will include average-case (Θ) and best-case (Ω). Note that this measures the number of operations a algorithm requires, however, different implementations (i.e. a vectorized method vs. a `for`-loop) of the same algorithm may run at different speeds.

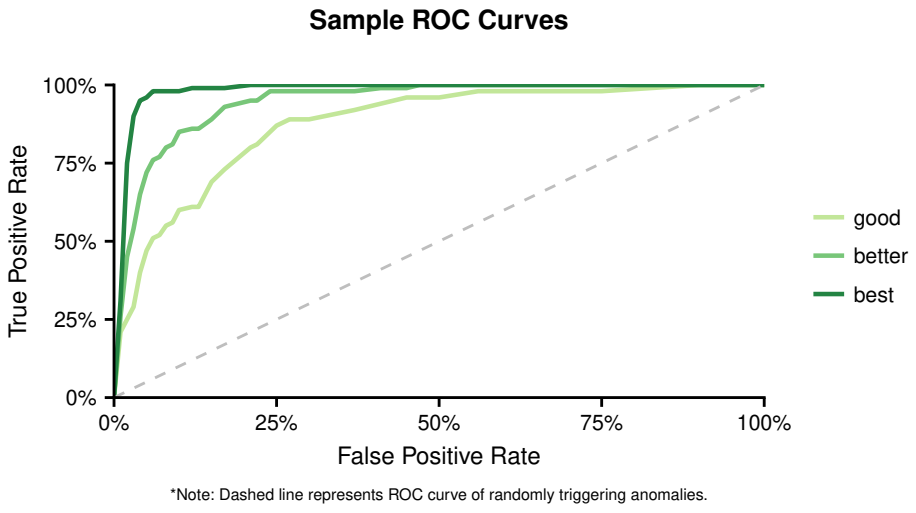


Figure 3: Comparing ROC curves

[5] Sensitivity and specificity
[6] Accuracy and precision
[7] Receiver operating characteristic
[8] Big O notation

Exploratory Data Analysis

Before we look into models, we should become more acquainted with our data and its intricacies. To do that, I will perform some exploratory data analysis (EDA) on the data for AS, TS, and RexT alerts (we don't need to find patterns in the logs), looking at the distributions of our data, correlation between metrics, the dispersion of the data, and seasonality. This will better inform future decisions as to what models to test and what parameters to tune and will also provide us with some insights into our data. Note that I will not be going over any summary statistics explicitly; while they are usually part of EDA, they are excluded here since the actual number values of those statistics don't matter for our purposes.

Distributions

Many statistical techniques require that the data you use it on have a certain underlying distribution, most often the normal. While many things are close to a normal, it makes sense to verify this assumption before moving forward with one of those such methods. Enter Quantile-Quantile plots (Q-Q plot for short). A Q-Q plot allows us to visually inspect if our data comes from a specific distribution and how well it matches up to that distribution. This is done by plotting the quantiles of the data (sample quantiles) against the quantiles drawn from the distribution to test (theoretical quantiles). A perfectly diagonal line would mean it is very likely from that distribution. Diagonal lines that are curved at the ends would indicate that the data has more extreme values than would be expected. Lastly, curves that are not close to diagonal lines indicate that the distribution is skewed (slight curve) or the data is most likely from a different distribution (no longer close to linear).[9]

In Figure 4, I included 2 Q-Q plots comparing the distribution of one of our metrics to the normal and to a Chi-square with 1 degree of freedom. It's pretty obvious that the distribution is not normal and closer to that of the Chi-square; the Chi-square Q-Q plot is curved at the ends meaning we have more extreme values than would be in the distribution—our anomalies.

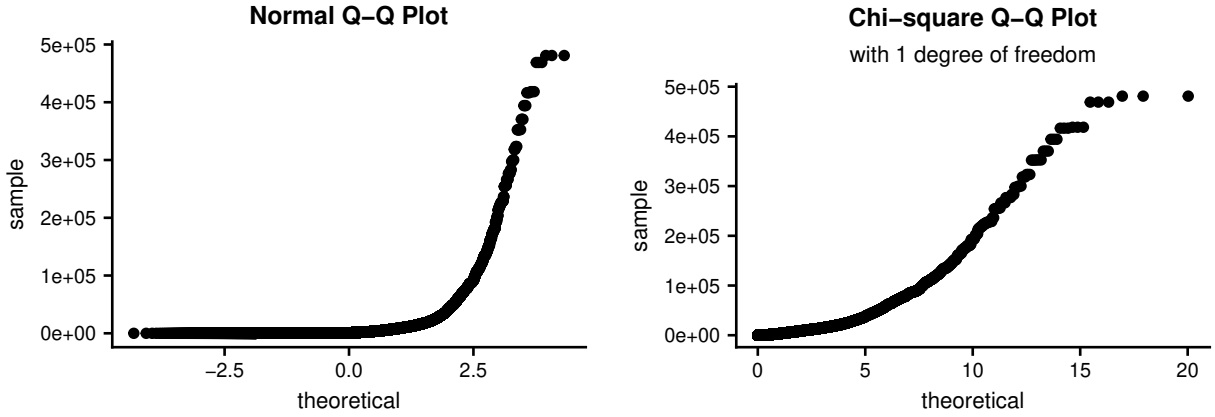


Figure 4: Sample quantile-quantile plots using normal and Chi-square distributions

There are several other ways we can look at the distribution of our data: boxplots, histograms, etc.; however, I have decided to introduce violin plots here. Violin plots are essentially a combination of the data we would get in a histogram and a boxplot: we get the density curve of the distribution and information about the summary statistics (first and third quartiles, median, min/max, etc.). This means not only do we get to see the distribution of the data and summary statistics, but we get some intuition into the dispersion measures (like range and interquartile range) which I will discuss again in a later section. I have also included information on the mean and standard deviations in the violin plots by metric in Figure 5 represented as the black dot and vertical line (horizontal lines are the first quartile, median, and third quartile, bottom to top).

[9] Understanding Q-Q Plots

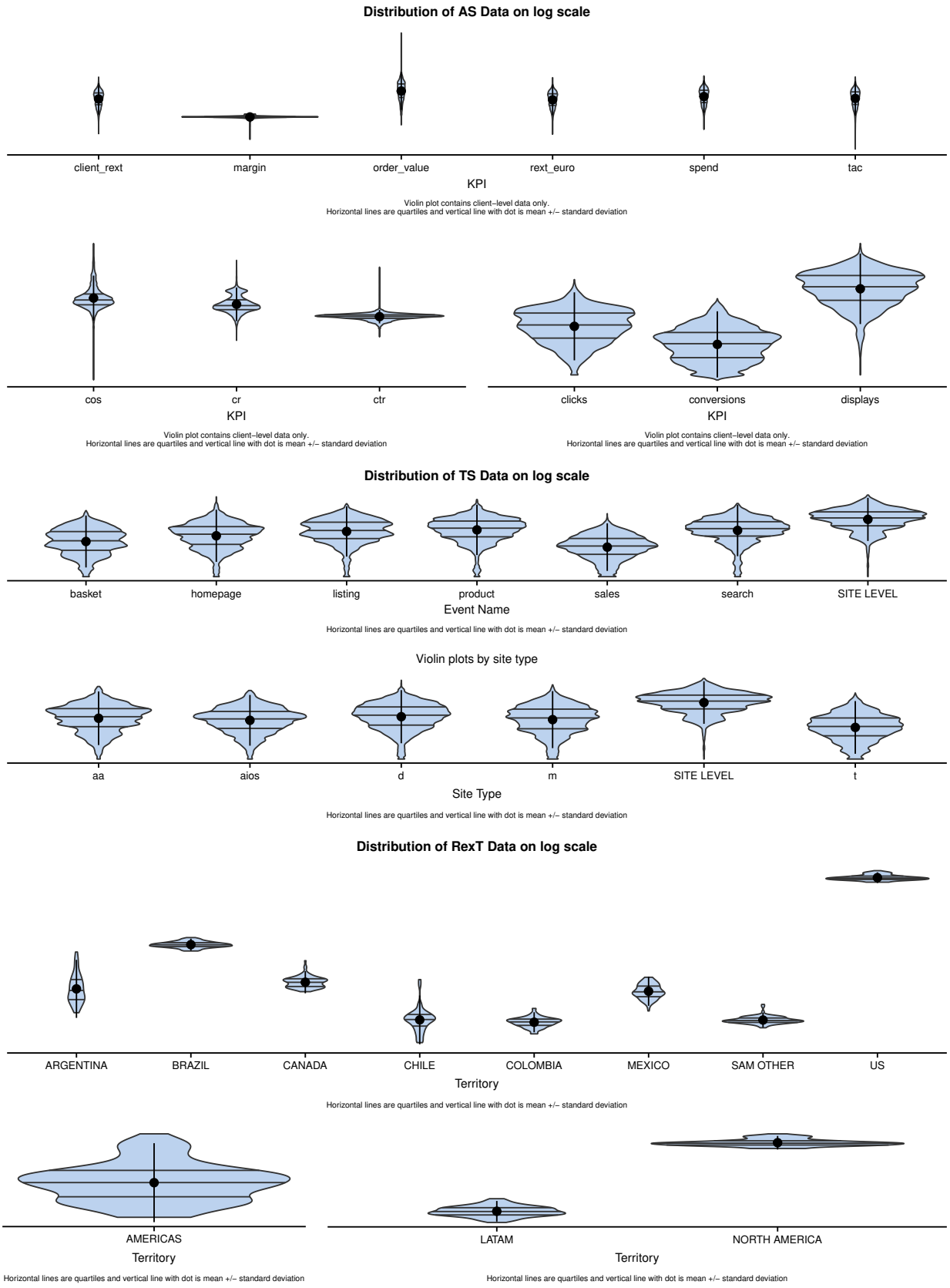


Figure 5: Distribution of data using violin plots

We can observe from the violin plots in Figure 5 that our data is definitely not normal. Most metrics are multi-modal (they have multiple modes—most common value). The means are very close to the medians meaning the data is not very skewed to the left or the right. There are some metrics that have very little variation—margin, CTR; most of their density is within a small range. Spend-related metrics have much lower densities throughout when compared to performance metrics; this makes sense, because we have budgets of all sizes represented and each client is different, thus a large range and long interquartile ranges ($IQR = Q3 - Q1$).

When comparing the distributions of the percentage metrics, we see that CTR has the lowest variance while COS has very long tails. There aren't any long tails with clicks, conversions, and displays which we would expect from looking at clients from the same tier. Notice also that the displays violin plot is higher up than the clicks and conversions violin plots; this follows our intuition—displays come first, a fraction of those get clicks, and then, some become conversions.

Site events violin plots are pretty similar in patterns, but, looking at the territory RexT ones, it's very easy to see the regions that contribute most (US and Brazil). All the regions have very different distributions, but observe that the higher plots seem to also be more consistent (less volatile).

Dispersion Measures

When looking at the violin plots in Figure 5, we could see the range ($max - min$), IQR, standard deviation from each side of the mean (and, therefore, also the variance). These are all measures of dispersion, or how far apart a given observation can be expected to be from other observations. Another commonly used one is the coefficient of variation which gives the typical deviation from the mean and can be expressed as a percentage. It is defined as:

$$Coefficient\_of\_variation = \frac{\sigma}{\mu} \tag{6}$$

where:

- σ = *standard deviation*
- μ = *mean*

In Figure 6, we have violin plots showing the coefficient of variation by metric. Spend-related metrics have nearly all of their density at or around 0% coefficient of variation with some long tails; this means these metrics are expected to be stable with few extreme values. This matches what we observe currently with the `SCOPE` alerts: RexT and spend are flagged much less often than the percentage performance metrics (COS, CTR, CR). There is more deviation with territory RexT most likely due to variation with the smaller territories being more common. Typical variation from the mean is very low for CTR, but much higher for COS and CR; it's no coincidence that those 2 metrics are the most commonly triggered in the current version of the alerts. Clicks, conversions, and displays are a little more stable, however, conversions has a very long right tail (meaning some very large deviations above the mean). Typical deviations from the mean for site events are essentially zero, as we would expect, because visitors to a site are pretty constant over time. The tails are much longer on non-app tags and on specific events.

Correlation — Relationships Between Metrics

When modeling how to detect anomalies, we need to keep in mind how the metrics are related to one another. If a change in one will cause a change in another, we should be using both to model that behavior. Linear relationships between the metrics can be captured by the Pearson correlation coefficient. A correlation coefficient of 1 (or -1) is perfect positive (negative) correlation meaning an increase in metric A will cause an increase (decrease) in metric B. The correlation coefficient is always between -1 and 1, so we can also get a sense of the strength of this relationship; strong relationships are close to ± 1 and weak ones are close to ± 0.5. A correlation coefficient of 0 represents no correlation, and the correlation of a metric to itself will always be 1.

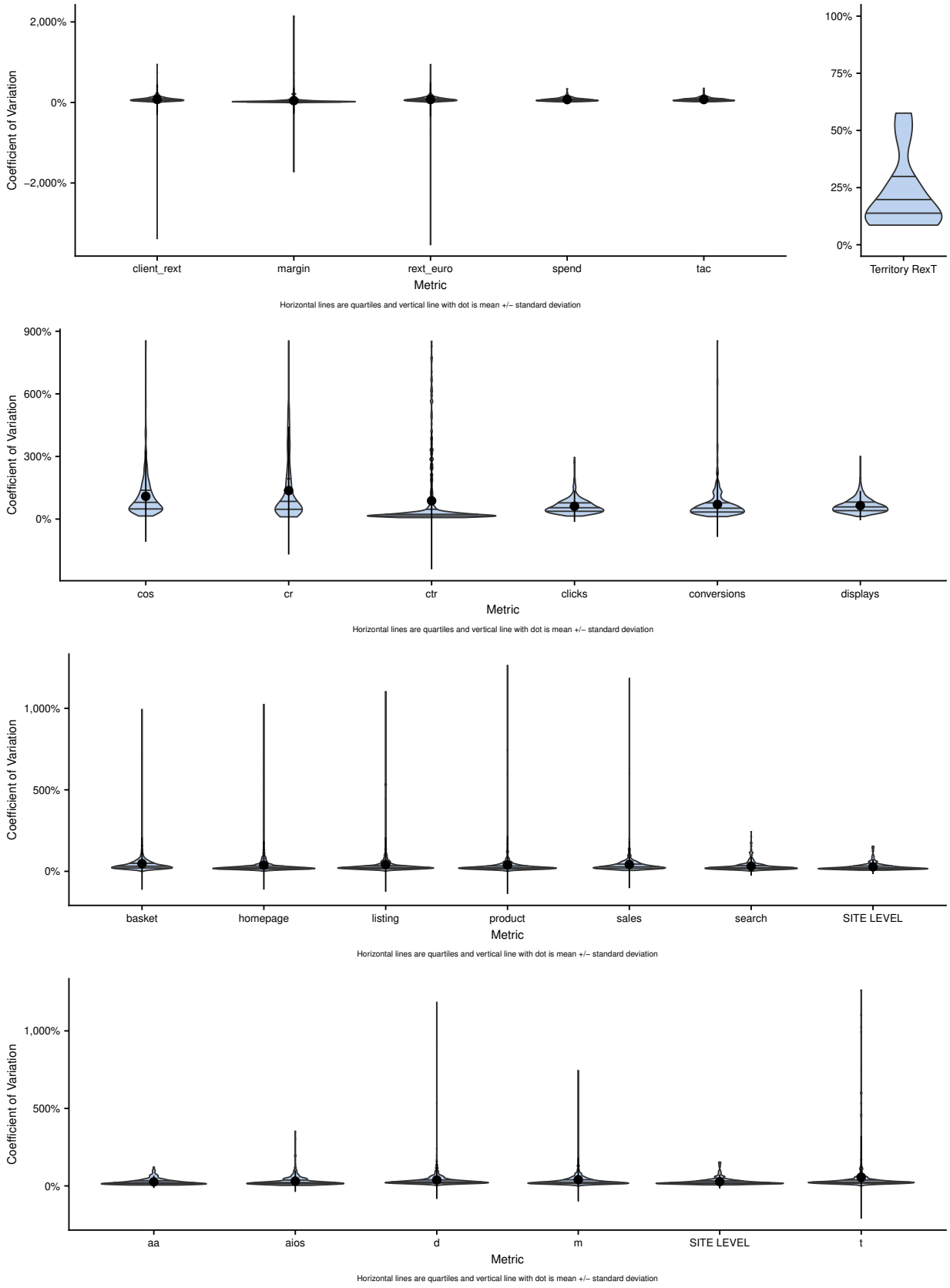


Figure 6: Distribution of coefficient of variation by metric

With that foundation, we can look at the linear relationships between our metrics using the correlation matrices in Figure 7. As we would expect, we have strong positive correlations between clicks and displays and amongst the spend-related metrics. We have weaker correlations between spend-related metrics and clicks, conversions, displays, as well as between clicks and conversions and displays and conversions. There is perfect positive correlation between North America and Americas, US and North America, and US and Americas; this is because the US is by far the largest component in North America and the Americas. Similarly, the strongest correlations for LATAM are with Brazil which is its largest component. Perhaps the strong correlation between the US and SAM Other comes as a surprise to you; this is most likely due to the US being an indicator of the Americas performance as a whole—we would expect others to follow it like with the Dow for the stock market. At the site events level, we don't have many noteworthy correlations; site level is highly correlated with desktop and mobile and somewhat with tablet — probably just due to app being such a small portion of site level.

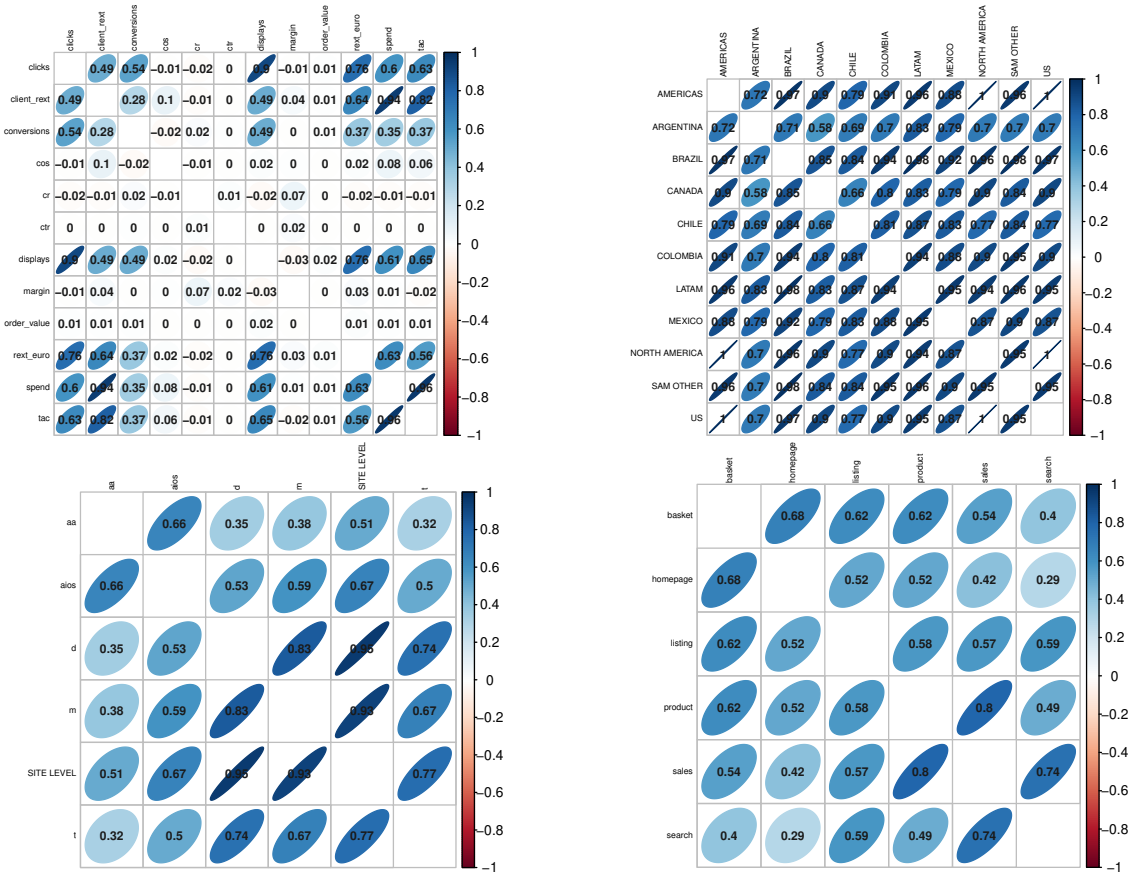


Figure 7: Correlation matrices (left to right from top-left: AS, RexT, site type, event name)

Seasonality

To get an idea of global seasonality trends, I made violin plots by day of week for all our metrics (Figure 8). We should expect that metrics with strong seasonality trends on the weekly level will have the plots at varying heights each day of the week. Quite interestingly, we can observe that we only have global weekly seasonality on the territory level. This debunks the thought that the weekend performs differently as a rule; if that were true, we would see it across the board. This means that this may be true, but in opposite directions for different clients, and thus cancelled out when aggregated. Alternatively, there could actually be no strong seasonality to pick up on.

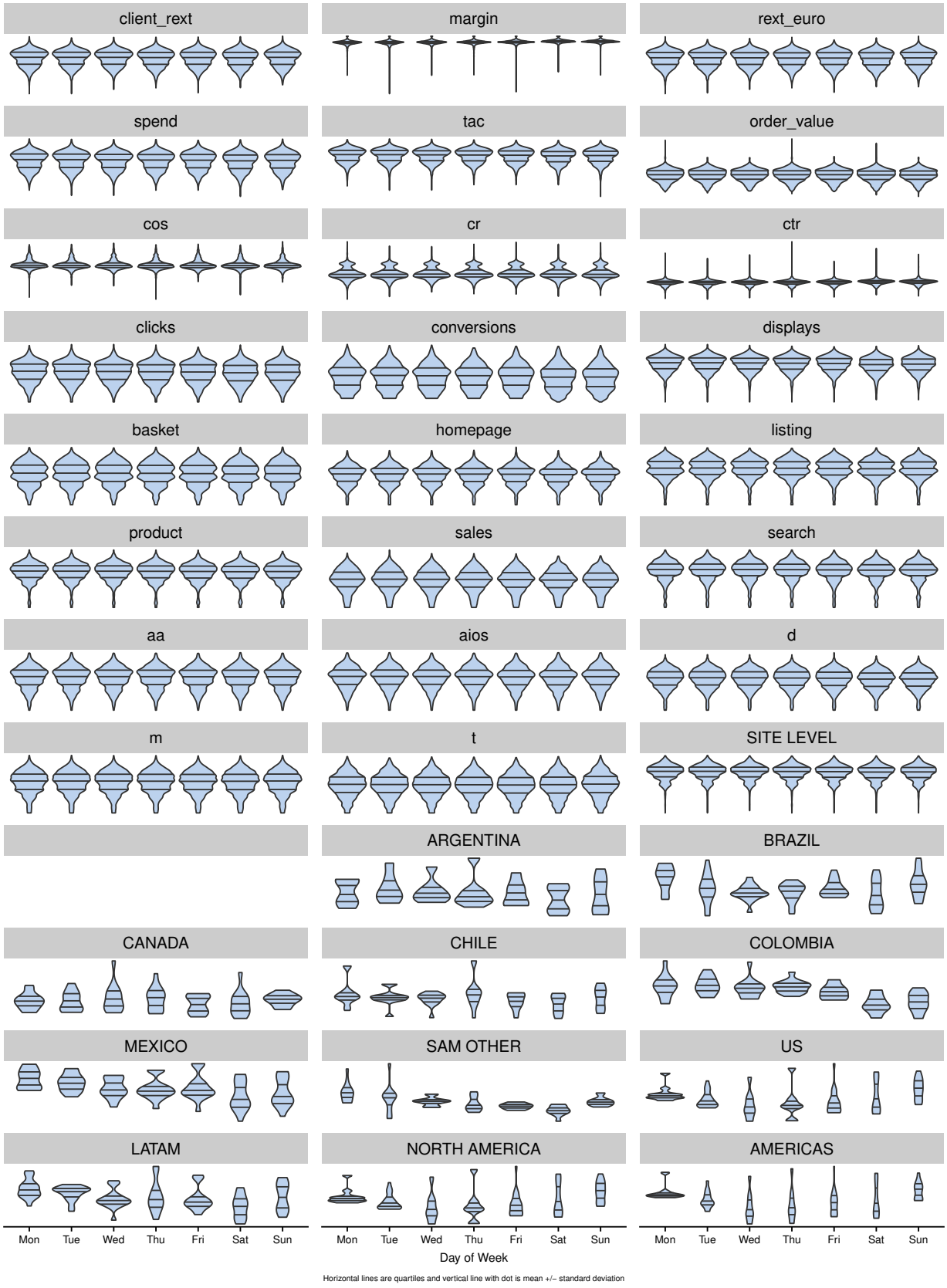


Figure 8: Distribution of metrics by day of week

Models

Now, we will take a look at the models SCOPE has used in the past along with variations on them, the current model, and potential future models. This section will serve as documentation of SCOPE's model history and provide us with the analysis we need to select the next new model. Along the way, we can hope to become more attuned to the idiosyncrasies of the data.

Since models often build on learnings from former ones, I will note any interesting comparisons between the models, in the model that was analyzed last. Here, we will go chronologically through the models starting with the original one; the potential new models don't have a specific order. You will notice how the models become more sophisticated compared to their predecessors, however, we have yet to have any machine learning algorithm due to the lack of labelled data prior to the creation of Metis and infrastructure to handle the additional computational load. The future proposed models will all have some element of machine learning now that we have access to a wealth of labelled data letting us know what is an alert and what isn't.

Ideally, there would be a way to tell how likely something is to be an alert, this way we could perhaps only send emails of the most likely alerts, but reserve the mid-range likelihood alerts for another medium, like a portal or dashboard. This would reduce the risk of desensitizing the audience to the emails with too many false positives, while having the audience trust that SCOPE is going to find things big or small. Of course, this introduces another optimization problem of finding what probability of alert makes the email and what doesn't, but that is more a matter of preference.

It is important to note that we aren't merely trying to build an anomaly detection system; we need to build a system that detects anomalies, but can also detect things before they are obvious anomalies. It became very clear to me during the user sessions I held with Metis to collect the labelled data for this case study that "alerts" aren't just drops or spikes; users want to be informed as to what is happening on their accounts, be it continuously trending downward/upward, flat-lining of certain metrics without recovery, sudden erratic behavior, drops after spikes and vice versa which may seem like returning to normalcy, etc. We need an alerting system that is robust enough to handle the "obvious" anomalies, but also alert us when the series starts to exhibit concerning behavior or when a series of metrics is concerning as a whole but not manifesting itself as a large change on each metric. For simplicity, I will refer to all these "alerts" as anomalies whether they are obvious or indications of possible future issues.

Note that some models will disqualify certain time series (not enough days, no TAC last 2 days, etc.), however, it is still possible to classify those in Metis; therefore, for the purposes of evaluating the models and penalizing them for having limitations that a human would not, those will be placed as "no alert" for the model and compared to user opinions. If the user also classified as "no alert", the model correctly disqualified the account (by ignoring it, it can't trigger an alert). Conversely, if the user classified it as an alert, then the model will be penalized with a false negative because it wrongly disqualified the account. The idea is to have a model that doesn't have many (if any) of these restrictions and can match up well with user opinions.

Prior Models

SCOPE has had 2 models prior to the model in production at the time of writing this paper (June 8, 2018): percent change from the average of prior days and Twitter's AnomalyDetection R package. Here, we will analyze those along with relevant variations. Note that for all these models we have to exclude some of the time series to make sure we have enough data for the tests. Users may still have flagged these as alerts, so will we flag them as not alerts to count everything, thereby penalizing these models for not being able to detect those types of alerts. AS and TS series need to have data for the day before the run date (the day to test) and have data for 25 of the last 30 days; additionally, for the AS data, we need to have TAC greater than 0 for the last 2 days.

Mean vs. Percent Change

When SCOPE was first created in April 2016 during a Hackathon, the model looked at the percent change of yesterday's value for CR, CTR, COS, and spend compared to the average of the 7 days prior. Each metric had its own alert threshold, and there were also different thresholds for top accounts versus other accounts, with top accounts having lower thresholds due to presumably less volatile performance. These thresholds ranged from 10 to 50 percent.

$$\%\Delta = \frac{yesterday - mean(L7D)}{mean(L7D)} \tag{7}$$

We have since added metrics, however, to be consistent with the future models, we will evaluate all metrics here. Furthermore, in our original model, we selected a 7-day lookback window for calculating the average; in this case study, I will also look at 30- and 60-day lookback windows before moving forward with the analysis on the best one (determined by ROC curves). As you can imagine, this model was *very* sensitive; as such, I won't use the original values for the thresholds, but rather select a threshold at an acceptable false positive rate (arbitrarily chosen).

Results

Our first step is to find the optimal lookback window and threshold for AS, TS, and RexT separately. We can expect the series to have different behaviors, since we have some very granular series with AS, some very broad series with RexT, and very seasonal data with TS series. To do so, we will graph ROC curves for each series for the 3 lookback windows (L7D, L30D, L60D) and a range of thresholds. From these, we will take the lookback window that maximizes the area under the ROC curve. Once we have our lookback window, we will choose a maximum false positive rate which will yield the threshold to use for flagging the alerts. Note that the maximum false positive rate will be roughly $1 - specificity$ when we dive into the results (see Equation 4).

AS Results

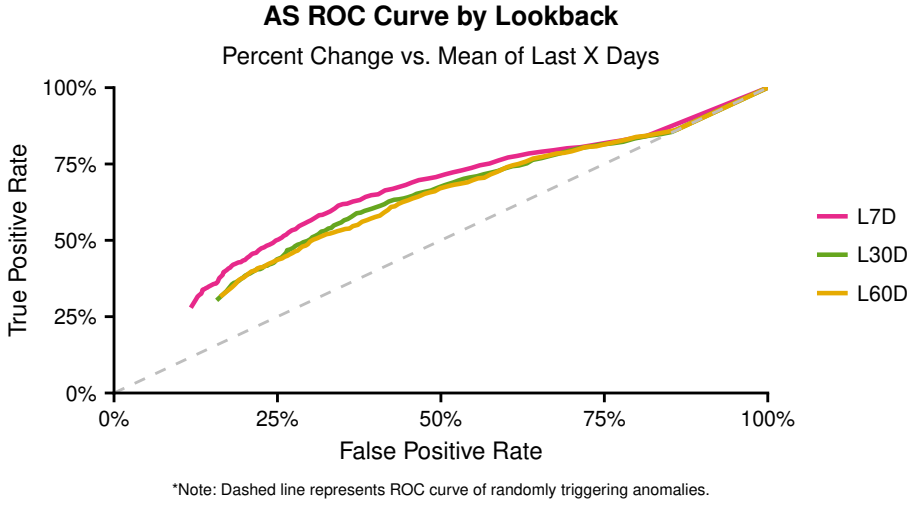


Figure 9: ROC curve for AS alerts using percent change from the mean and different lookback windows

When using the mean, last 60 days seems to be too much data for an average. This is most likely due to the propensity of an account to fluctuate quite a bit over time. It is closer between the 7- and 30-day lookbacks,

however, the 7 day is a better choice. AS users tend to weigh more recent events higher, due to account changes, so 7-day lookbacks carry slightly more weight—the 7-day lookback has more area under the curve in Figure 9.

It's interesting to note that at false positive rates above 80%, our performance converges to that of the method of randomly classifying (diagonal line). This means that at this point, since our threshold is so low (~8%), every positive we trigger at this threshold or lower has an equal likelihood of being a true positive or a false positive—not at all optimal. Using the ROC curve data, I have identified the threshold of 67% by choosing a threshold with a false positive rate of 20%. This will be used to check if AS metrics are anomalies when compared to the average of the last 7 days prior.

Now, we can start to gain an appreciation of just how sensitive percent change based methods are (see Figure 10). For the beginning of the month, we triggered alerts for 57% of the clients we looked at! (end = 56%, middle = 44%). Due to the nature of an anomalous event being rare, surely these cannot be so commonplace that we have half the clients with alerts on a daily basis. Not only are we triggering lots of client alerts, but campaign alerts as well.

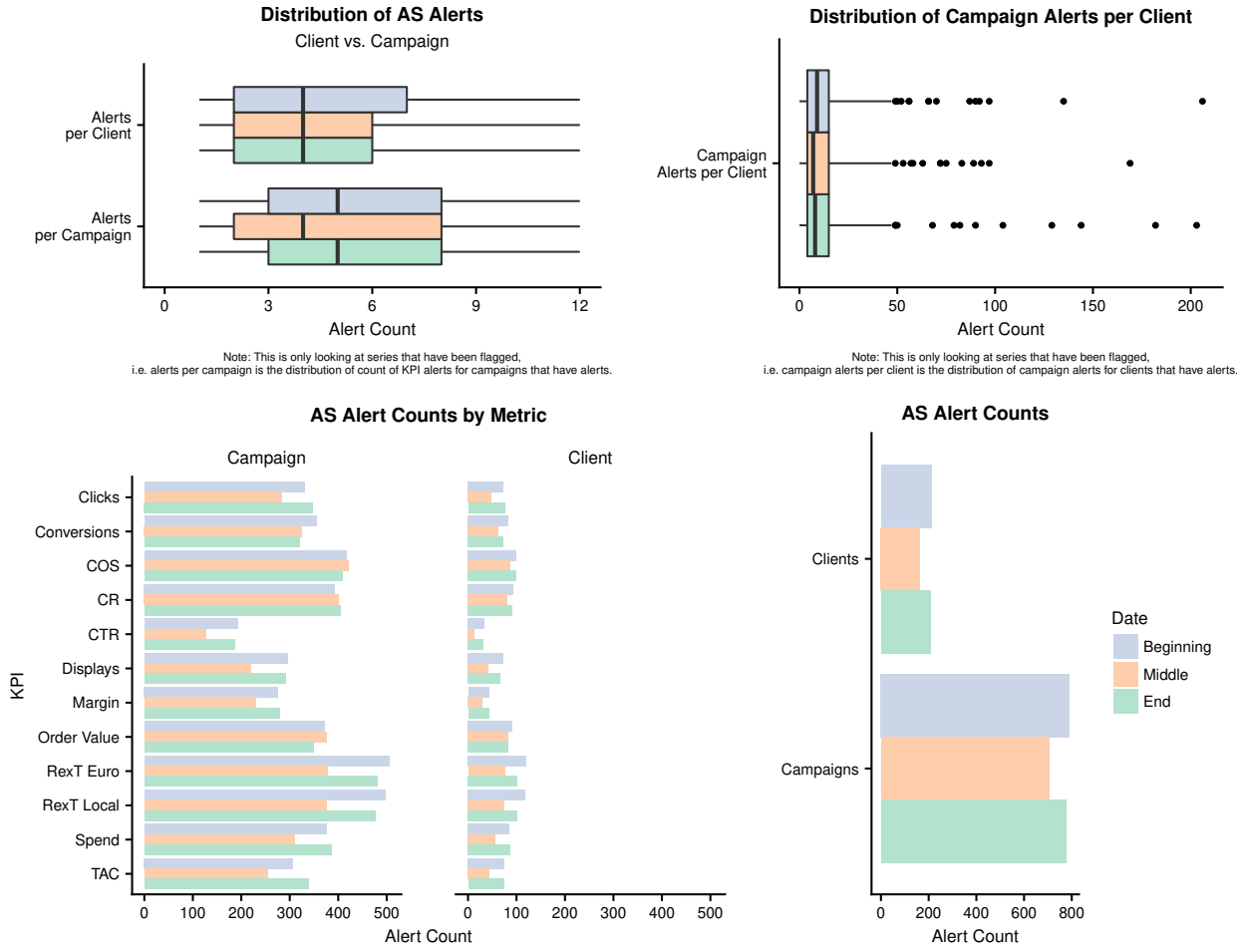


Figure 10: AS alert counts using Percent Change from the Mean

We trigger very few CTR and margin alerts compared to the other metrics we are investigating. This is probably due to the tendency of these metrics to be relatively steady on a day-to-day basis. We see the highest quantity of alerts on RexT metrics at the beginning of the month which is most likely due to spend increasing at the start of each month with new budget and the effect being more pronounced when the TAC

doesn't adjust accordingly (inventory can cost more at the end of the month and will be freed up at the start of month, meaning more supply and lower TAC). This same pattern can be observed at the campaign level, but at a higher volume.

Additionally, using the boxplots in Figure 10, we learn that this method triggers between 2 and 7 alerts for 50% of our clients with alerts and between 3 and 8 alerts for 50% of our campaigns with alerts. Notice how our minimum is always 1; this is because we are only looking at clients and campaigns, respectively, that have been flagged. Also, note that the maximum is always 12, because for AS alerts, we are looking at 12 KPI's. Furthermore, we see that when a client triggers an alert, the median number of KPI's that are flagged as client-level alerts is 4; this sounds kind of high, but since many of the metrics are correlated it's not surprising. This value is even higher for campaign metrics, meaning we tend to flag more campaign-level metrics than client-level metrics, which follows our intuition that campaign-level metrics are more volatile due to higher granularity.

It's also interesting to note that while nearly all of the clients we looked at have between 0 and 50 campaign alerts, there are quite a few with very large campaign alert counts: some have over 200, and this is just one day of data per date! This means that if the client had 20 campaigns, for example, *each* campaign had alerts for *at least* 11 out of the 12 KPI's checked. This is certainly an excessive volume of alerts.

When we compare the results to the data collected with `Metis` (see Table 14), we have pretty poor `precision` across the board due to our high percentage of false positives. Our `recall` is much better (although still mediocre), since it takes into account true positives (Equation 2), which we have lots of, because we are seemingly flagging everything, and false negatives, of which we hardly have any due to excessive flagging. Since the `F1-score` is a balance between the `precision` and `recall` (see Equation 3), it's pretty mediocre as well. Our `specificity` (true negative rate) is average, since we have a large amount of false positives due to over-sensitivity; notice how the `specificity` is roughly 1 minus the false positive rate threshold we chose to determine the cutoff threshold for alerts (see Equation 4). Our `accuracy` is pretty poor here; since it is a very forgiving (and sometimes misleading) metric, because it takes into account all 4 possibilities (TP, FP, TN, FN — equation in Equation 5), we want our `accuracy` to be in the 90's, at least.

Table 14: Prediction vs. Metis for AS alerts using Percent Change from the Mean

Date	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Beginning	245	353	1,281	263	0.41	0.48	0.44	0.78	0.71
Middle	164	305	1,466	272	0.35	0.38	0.36	0.83	0.74
End	196	348	1,344	255	0.36	0.43	0.39	0.79	0.72
TOTAL	605	1,006	4,091	790	0.38	0.43	0.40	0.80	0.72

Metric	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Clicks	43	43	192	77	0.50	0.36	0.42	0.82	0.66
Conversions	25	51	185	62	0.33	0.29	0.31	0.78	0.65
COS	47	79	194	57	0.37	0.45	0.41	0.71	0.64
CR	51	57	162	53	0.47	0.49	0.48	0.74	0.66
CTR	18	23	244	57	0.44	0.24	0.31	0.91	0.77
Displays	42	32	218	80	0.57	0.34	0.43	0.87	0.70
Margin	56	129	866	55	0.30	0.50	0.38	0.87	0.83
Order Value	42	45	209	73	0.48	0.37	0.42	0.82	0.68
RexT Euro	105	281	674	72	0.27	0.59	0.37	0.71	0.69
RexT Local	37	60	162	63	0.38	0.37	0.38	0.73	0.62
Spend	55	52	208	73	0.51	0.43	0.47	0.80	0.68
TAC	84	154	777	68	0.35	0.55	0.43	0.83	0.80

We also see that we perform poorly across all metrics, but what is the most interesting are the metrics we perform the worst on using the `F1-score` (because `precision` and `recall` are only half the picture): margin and CTR. These metrics were the metrics we flagged the least! This means that not only are we over-sensitive across the board, but we aren't picking up on what is truly an alert. In using the percent change, we are not taking into account the context of the fluctuations over time just a predetermined threshold.

TS Results

For the TS ROC curve, we see the opposite trend as the AS curve. Here, last 60 days is actually the best performing and 7 days is the worst. We expect things like site events to be highly seasonal, so the more data we have, the better the mean describes it. TS users often looked for how far yesterday deviated from the central tendency of the site events graph. This is consistent with our findings here. The mean becomes a better descriptor of the data the more observations it has, because we smooth out the effects of seasonal fluctuations week over week.

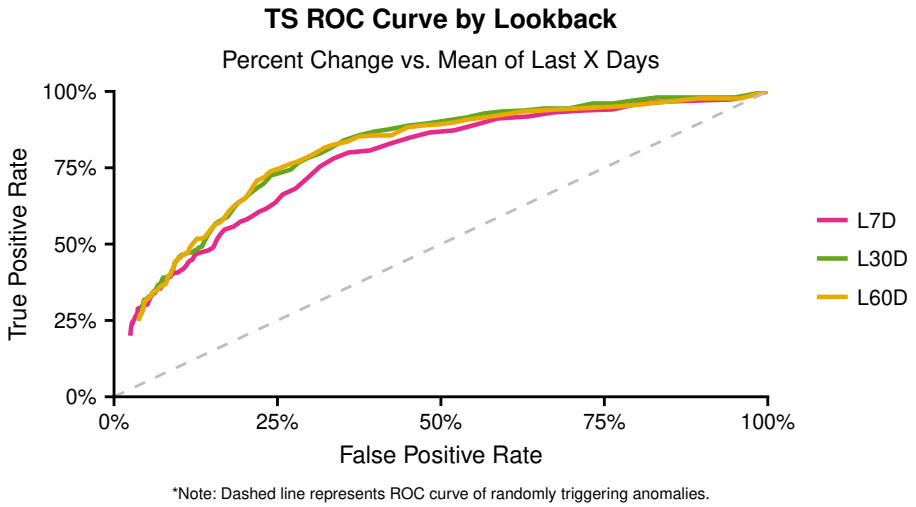


Figure 11: ROC curve for TS alerts using percent change from the mean and different lookback windows

Using the ROC curve data in Figure 11, I have identified the threshold of 36% by choosing a threshold with a false positive rate of 20%. This will be used to check if TS metrics are anomalies when compared to the average of the last 60 days prior. Notice how this threshold is lower than the AS metrics threshold of 67% at the same false positive rate; further evidence that the TS time series behave differently than their AS counterparts.

For the beginning of the month, we triggered alerts for 22% of the clients we looked at. (end = 20%, middle = 14%). Much lower than the AS percentages; site events are expected to fluctuate within a certain range due to seasonality—they aren't as volatile as the AS metrics can be.

Excluding the search tag since that is not super relevant for us, we have way fewer alerts on site level and homepage compared to the other tags. This is probably because those series have much larger volumes (many more people visit the homepage than the other pages), and site level is the aggregate of all the individual pages. We should expect these series to be less volatile, and therefore, have less alerts triggered. Conversely, the pages with the lower volumes, like sales, would be more volatile, and thus, trigger more alerts with this method. There doesn't seem to be any discrimination by site type (remember that not all clients use app tags and not everyone owns a tablet, so we have to take the raw counts with that in mind).

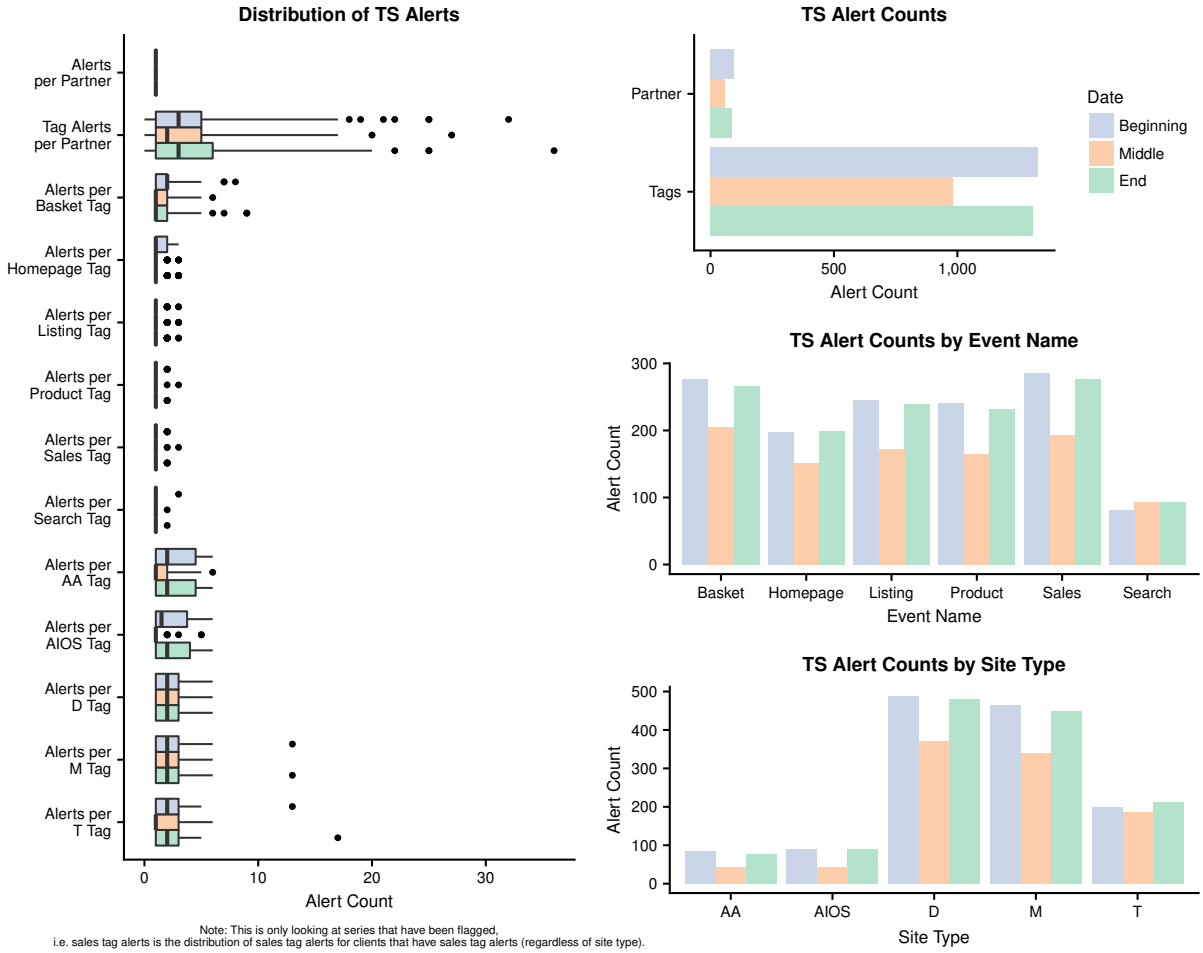


Figure 12: TS alert counts using Percent Change from the Mean

The boxplot in Figure 12 reveals that we have higher variability in the number of homepage and basket tags with alerts compared to the other tags, although it is minimal. For event name tags, the median amount of alerts and, in fact, the whole boxplot in some cases, is just 1 alert; this means that when these tags break, they most likely break on a specific site type. Conversely, the site type tags have variation; for the most part, the median is 2 alerts. Therefore, if we have a desktop tag issue, we would expect it to affect 2 event names tags (homepage and sales, for example). This is quite interesting, because it means that (at least with this model) potential tag issues for a specific site type materialize themselves across more than one event name. We can also observe that, for the most part, partners have below 10 tag alerts at a given time and that the median is lower during the middle of the month. Note that alerts per partner can only ever be 1, because we are showing series that have alerts and the only partner-level alert possible is site-level.

As with the AS results, our metrics are poor across the board. We are flagging such a large number of cases that we hardly have any false negatives, but our false positives are quite high. `F1-score` is very low and accuracy is poor (see Table 16). We can do much better than this. We are performing best on basket and site level, but make no mistake, this performance is mediocre at best. Same with site types, site-level is the best performing, but that is probably getting help from the aggregation of all the events to form it smoothing out the volatility.

Table 16: Prediction vs. Metis for TS alerts using Percent Change from the Mean

Date	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Beginning	72	243	885	34	0.23	0.68	0.34	0.78	0.78
Middle	41	180	894	37	0.19	0.53	0.27	0.83	0.81
End	82	200	814	39	0.29	0.68	0.41	0.80	0.79
TOTAL	195	623	2,593	110	0.24	0.64	0.35	0.81	0.79

Metric	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Basket	37	113	379	11	0.25	0.77	0.37	0.77	0.77
Homepage	25	101	467	14	0.20	0.64	0.30	0.82	0.81
Listing	30	99	354	19	0.23	0.61	0.34	0.78	0.76
Product	36	102	522	23	0.26	0.61	0.37	0.84	0.82
Sales	24	128	439	17	0.16	0.59	0.25	0.77	0.76
Search	22	42	207	18	0.34	0.55	0.42	0.83	0.79
Site Level	21	38	225	8	0.36	0.72	0.48	0.86	0.84

Metric	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
AA	15	42	135	8	0.26	0.65	0.38	0.76	0.75
AIOS	10	34	146	3	0.23	0.77	0.35	0.81	0.81
D	55	229	991	25	0.19	0.69	0.30	0.81	0.80
M	61	194	743	50	0.24	0.55	0.33	0.79	0.77
Site Level	21	38	225	8	0.36	0.72	0.48	0.86	0.84
T	33	86	353	16	0.28	0.67	0.39	0.80	0.79

Territory RexT Results

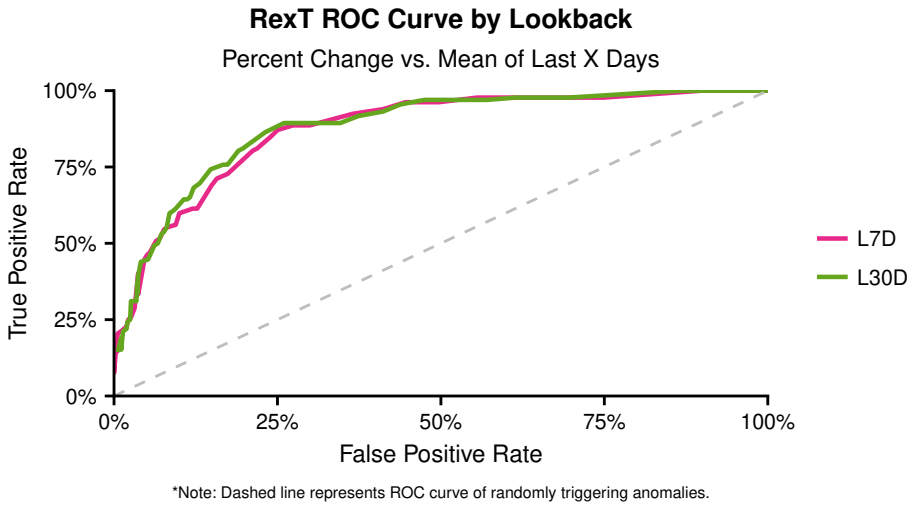


Figure 13: ROC curve for RexT using percent change from the mean and different lookback windows

Here, we only have a choice between 7 days and 30 days, because the RexT alerts only look back 30 days. It

is definitely a close call, but the 30 day window wins out. This data is highly aggregated, and therefore, not as volatile as the other time series we look at; hence, performance over a 30 day period is more indicative of overall patterns than just a week.

Using the ROC curve data in Figure 13, I have identified the threshold of 19% by choosing a threshold with a false positive rate of 20%. This will be used to check if territory RexT has anomalies when compared to the average of the last 30 days prior. Notice how this threshold is lower than the AS metrics threshold of 67% and the TS threshold of 36% at the same false positive rate; further evidence that the territory RexT time series being so aggregated leads to a much lower threshold for anomalous behavior.

Percent change from the mean is most sensitive at the country-level and least sensitive at the region-level. This makes a lot of sense, because we would expect the more granular series to have more volatility just as individual stocks are more volatile than mutual funds. Here, we have a different pattern with time of month than the AS and TS series, we flag the most alerts in the beginning and middle of the month with the end having the least. This model flags Canada, the LATAM countries, and the subregion itself quite a bit; probably due to them being much smaller volume-wise, and thus, more prone to volatility. The boxplot in Figure 14 furthers this notion; we see much larger variablility in the alerts for the LATAM region and its component countries than for North America. Overall, the number of alerts are most variable in the middle of the month.

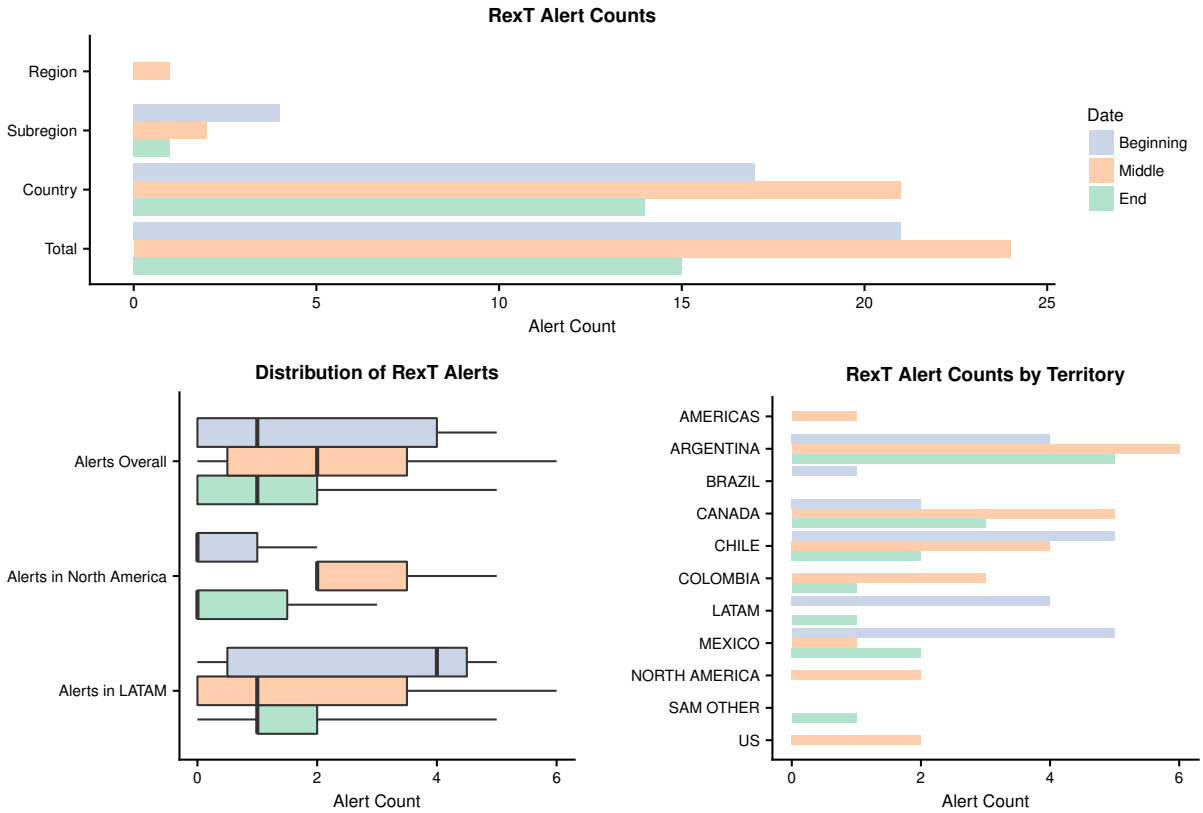


Figure 14: RexT alert counts using Percent Change from the Mean

For some reason, we perform significantly worse for the end of the month—perhaps some seasonality is at play; as shown in Table 19, we flagged very few during this period, but looks like the model wasn't catching things. Other than the poor performance for end-of-month, this model performs *much* better on this data compared to the AS and TS data sets. Though still not something we would want to use in practice, as the scores are still not great, we can see that the more aggregated the data gets, the better this method performs. Finally, some good news: this model performs perfectly for Brazil and pretty well on Mexico; the

other territories are all over the place. Keep in mind that this isn't a huge sample size when we go down to the country granularity, so take this with a grain of salt.

Table 19: Prediction vs. Metis for territory RexT alerts using Percent Change from the Mean

Date	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Beginning	10	11	45	0	0.48	1.00	0.65	0.80	0.83
Middle	11	13	41	1	0.46	0.92	0.61	0.76	0.79
End	3	12	51	0	0.20	1.00	0.33	0.81	0.82
TOTAL	24	36	137	1	0.40	0.96	0.56	0.79	0.81

Territory	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
AMERICAS	0	1	16	1	0.00	0	NaN	0.94	0.89
ARGENTINA	4	11	3	0	0.27	1	0.42	0.21	0.39
BRAZIL	1	0	17	0	1.00	1	1.00	1.00	1.00
CANADA	2	8	8	0	0.20	1	0.33	0.50	0.56
CHILE	6	5	7	0	0.55	1	0.71	0.58	0.72
COLOMBIA	3	1	14	0	0.75	1	0.86	0.93	0.94
LATAM	2	3	13	0	0.40	1	0.57	0.81	0.83
MEXICO	3	5	10	0	0.38	1	0.55	0.67	0.72
NORTH AMERICA	2	0	16	0	1.00	1	1.00	1.00	1.00
SAM OTHER	0	1	17	0	0.00	NA	NA	0.94	0.94
US	1	1	16	0	0.50	1	0.67	0.94	0.94

ROC Curve

With the ROC curve of AS, TS, and RexT results at their optimal lookback windows (Figure 15), we see that this method always performs worse for the AS compared to TS and RexT. This method performs best on aggregated data like that of the territory RexT. At our chosen false positive rate (20%), we can see the true positive rates for each product (intersections with purple line). While we definitely are performing better than randomly triggering alerts (dashed line), we have work to do to get our ROC curve closer to optimal.

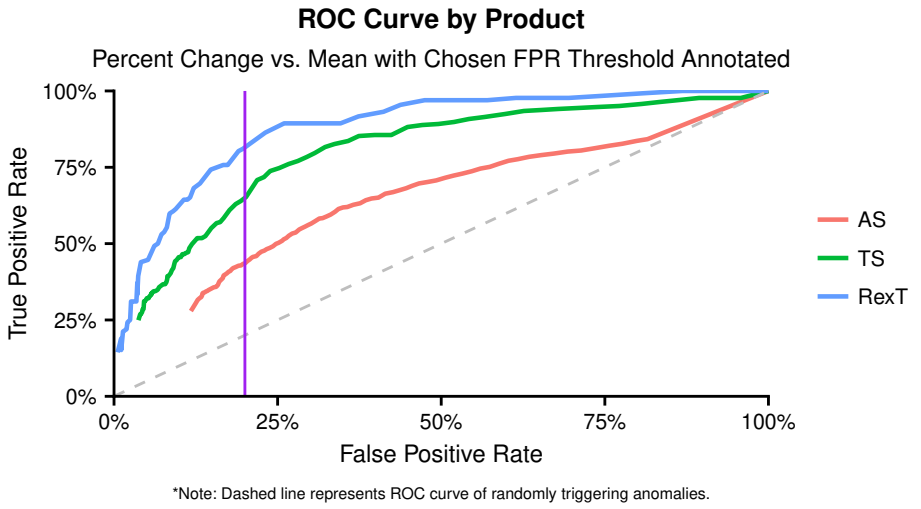


Figure 15: ROC curve for percent change from mean at optimal lookback windows

Alogrithm Complexity

This method is implemented as a vectorized operation, so it will run faster time-wise than the complexity dictates; however, for the complexity analysis we have to check each combination against the threshold which is equivalent to running it through a loop. Average complexity (Θ):

$$\Theta(n) = k * n \tag{8}$$

where:

- n = *number of series to inspect (i.e. total number of clients + total number of campaigns)*
- k = *number of KPIs to inspect (for each series)*

This gives us an overall linear complexity ($\Theta(n)$), since k should always be way less than n, as well as, a tight best-case of $\Omega(n)$ and tight worst-case of $O(n)$. For large n, we would be better off with a faster algorithm if one exists.

Conclusions

As expected, the model had `specificity` of roughly 80% due to our chosen false positive rate. Across AS, TS, and territory RexT, this model performed poorly on `precision`, `recall`, and `F1-score`; our accuracy and ROC curves could definitely be improved. We are successfully flagging a lot of alerts, but our false positives are pretty high. We have very few false negatives, but not because our model is great at capturing expected performance, but rather because we flag so many things as positive that the conditional probability of flagging negative and having it be wrong is low. Furthermore, there is the problem of setting thresholds: if we set common thresholds this won't work for everyone; and, if we have users set it, `SCOPE` has to store those, and the users have to be responsible to update them—not likely to be done judging from past experience. While this method performs decently on the territory RexT data, since it is highly aggregated, it is not fit for granular data or seasonal data like we see with the AS and TS metrics.

Note that, even though this method has proved very sensitive, when `SCOPE` used this in production, our thresholds were even lower than what we selected here! This means that we were even more trigger-happy and our false negative rate was higher. This is very dangerous, because users become desensitized to the alerts when more often than not they are false positives. We have had this issue with `SCOPE`, and once we implemented the current model, while we miss more smaller issues now, users have commented that they now have to make sure they read the alerts instead of directing them to a folder to ignore. It is clear that this method is not optimal, as it doesn't seem to understand the underlying data.

Median vs. Percent Change

For the sake of completeness, I will also look at percent change from median aggregations. This is the same process for the prior model using the mean, just changing the summary metric to the median. The mean is affected by outliers, while the median is robust; therefore, it is better to use the median when you may have outliers in the data you are using to determine whether a current value is abnormal. It should be noted that this model was never used in production—I'm providing it here to be thorough.

$$\%\Delta = \frac{yesterday - median(L7D)}{median(L7D)} \tag{9}$$

As I did with the analysis on the percentage change from the mean, I will use ROC curves to identify the optimal lookback periods for the summary statistic (the median in this case) and pick a threshold with an acceptable false positive rate.

Results

Once again, our first step is to find the optimal lookback window and threshold for AS, TS, and RexT separately. The choice of the median as the summary statistic may also influence which will be best; we can't assume that what worked best for the mean will also suffice for the median. I will follow the process I used for the prior analysis on the mean. For clarity, I will repeat it here. I will graph ROC curves for each series for the 3 lookback windows (L7D, L30D, L60D) and a range of thresholds. From these I will take the lookback window that maximizes the area under the curve. Once we have our lookback window, I will choose a maximum false positive rate which will yield the threshold to use for flagging the alerts. Note that the maximum false positive rate will be roughly $1 - specificity$ when we dive into the results (see Equation 4).

AS Results

With the median, last 60 days still seems to be too much data. However, it is also very close between the 7- and 30-day lookbacks. As with the mean, the 7-day is better than the 30-day. This is most likely due to the fact that a few days of abnormal performance won't affect the median, so the 7-day is still a full picture of what to expect from performance.

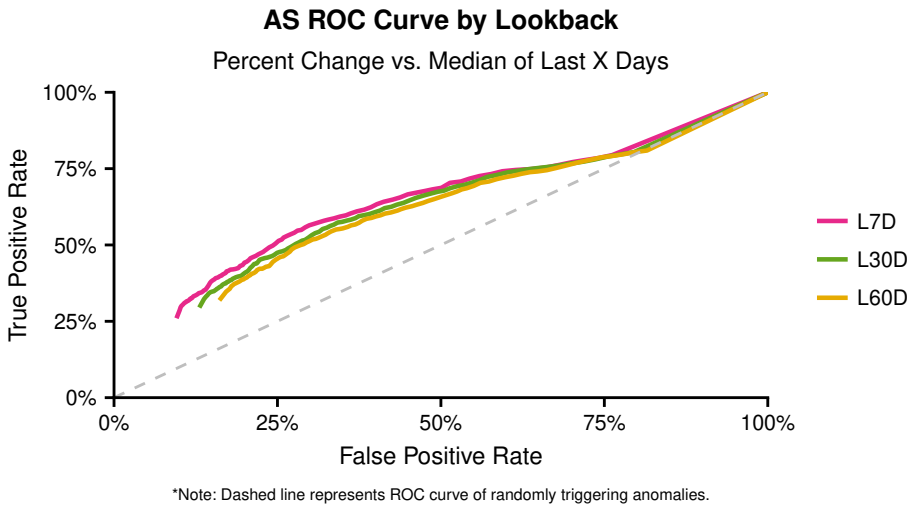


Figure 16: ROC curve for AS alerts using percent change from the median at various lookbacks

It's interesting to note that, as with the mean, our performance converges to that of the method of randomly classifying (diagonal line), but at an lower false positive rate of about 76%. This means that at thresholds of 3% or lower, every positive we trigger has an equal likelihood of being a true positive or a false positive—not at all optimal.

Using the ROC curve data (Figure 16), I have identified the threshold of 57% by choosing a threshold with a false positive rate of 20%. This will be used to check if AS metrics are anomalies when compared to the average of the last 7 days prior.

As with the mean method discussed earlier, the percent change from the median is *super* senstive. For the beginning of the month, we triggered alerts for 57% of the clients we looked at! (end = 56%, middle = 44%). The median also seems to be a more sensitive method across the board compared to the mean, but it is especially more sensitive on the beginning and end of the month. We observe the same pattern with margin and CTR alerts at the client-level as with the mean: very few alerts comparatively. Same is true with the campaign-level analysis. Our volume of alerts is higher, however. The alerts per client under the median (Figure 17) have a similar distribution for the beginning and end of the month compared to the mean, but we actually reduced our median number of alerts middle of month and while increasing the spread for the box in

the boxplot for the beginning and end of the month. Now, 50% of the clients have 2-7 alerts instead of 2-6. For the campaign alerts per client, the 50% sections of our distributions no longer align across time of month. Beginning of the month and end of month are shifted higher in alert count compared to the middle of the month. The median also has less clients in the 150+ alerts section.

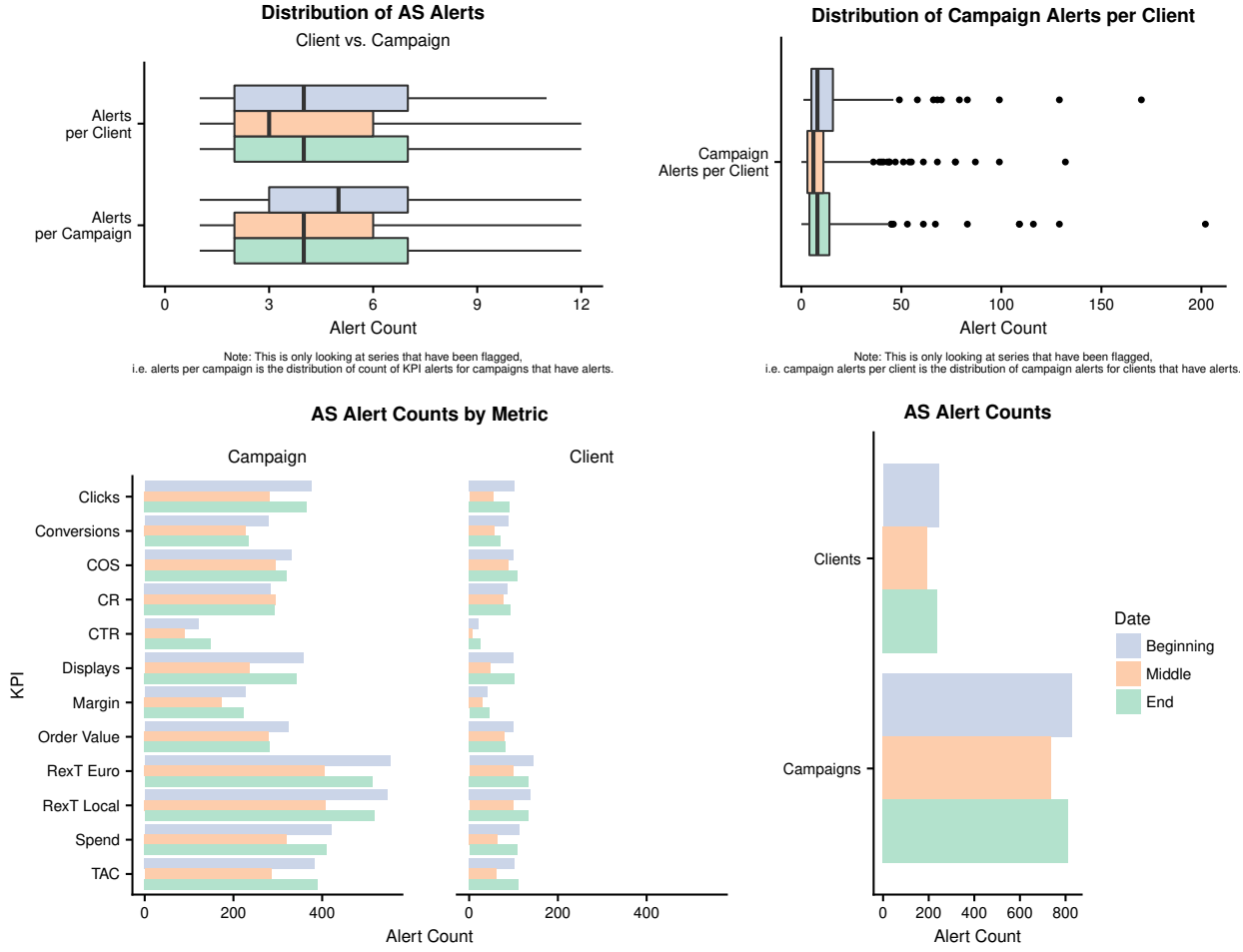


Figure 17: AS alert counts using Percent Change from the Median

As with the mean, our performance is pretty poor, though perhaps slightly better here as we seem to have identified a few extra true negatives and true positives (see Table 21). We still have abysmal `precision`, and our `recall` continues to be mediocre. Our `F1-score` is a far cry from 1. `Accuracy` is sub-80—we shouldn't even consider using this model. The performance is also horrendous by metric. We do the worst on conversions and CTR this time.

Table 21: Prediction vs. Metis for AS alerts using Percent Change from the Median

Date	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Beginning	244	357	1,277	264	0.41	0.48	0.44	0.78	0.71
Middle	170	297	1,474	266	0.36	0.39	0.38	0.83	0.74
End	203	361	1,331	248	0.36	0.45	0.40	0.79	0.72
TOTAL	617	1,015	4,082	778	0.38	0.44	0.41	0.80	0.72

Metric	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Clicks	47	43	192	73	0.52	0.39	0.45	0.82	0.67
Conversions	20	36	200	67	0.36	0.23	0.28	0.85	0.68
COS	49	59	214	55	0.45	0.47	0.46	0.78	0.70
CR	40	43	176	64	0.48	0.38	0.43	0.80	0.67
CTR	13	14	253	62	0.48	0.17	0.25	0.95	0.78
Displays	53	39	211	69	0.58	0.43	0.50	0.84	0.71
Margin	55	97	898	56	0.36	0.50	0.42	0.90	0.86
Order Value	43	44	210	72	0.49	0.37	0.43	0.83	0.69
RexT Euro	107	316	639	70	0.25	0.60	0.36	0.67	0.66
RexT Local	46	67	155	54	0.41	0.46	0.43	0.70	0.62
Spend	58	53	207	70	0.52	0.45	0.49	0.80	0.68
TAC	86	204	727	66	0.30	0.57	0.39	0.78	0.75

TS Results

With the mean, it was much clearer to see that 60 days was a better lookback window than 30 days, however, with the median they are much closer in performance. We will go with the 60 days once again here.

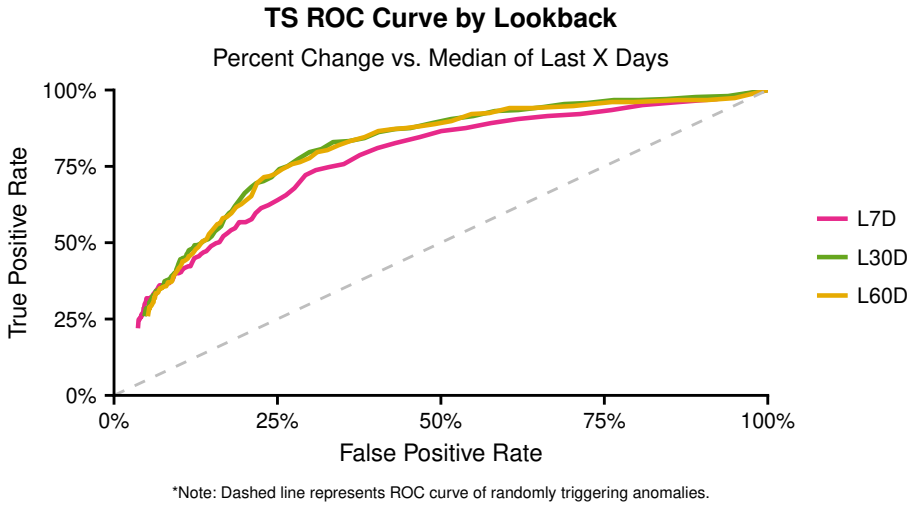


Figure 18: ROC curve for TS alerts using percent change from the median at various lookbacks

Using the ROC curve data (Figure 18), I have identified the threshold of 35% by choosing a threshold with a false positive rate of 20%. This will be used to check if TS metrics are anomalies when compared to the average of the last 60 days prior. Notice how this threshold is lower than the AS metrics threshold of 57% at the same false positive rate; further evidence that the TS time series behave differently than their AS counterparts.

For the beginning of the month, we triggered alerts for 22% of the clients we looked at (end = 20%, middle = 12%). With the TS metrics, the median is less sensitive than the mean. This is probably due to the TS metrics varying according to their seasonality component. We can observe the same pattern at the event name and site type as with the mean. We have fewer alerts on the larger and aggregate series: site-level and homepage, but many more alerts on the lower-traffic pages. There doesn't seem to be any discrimination by site type (remember that not all clients use app and tablet tags, so we have to take the raw counts with that in mind). The distribution of alerts for the median (Figure 19) is pretty consistent with that of the mean; we have smaller variation in some spots, but nothing huge.

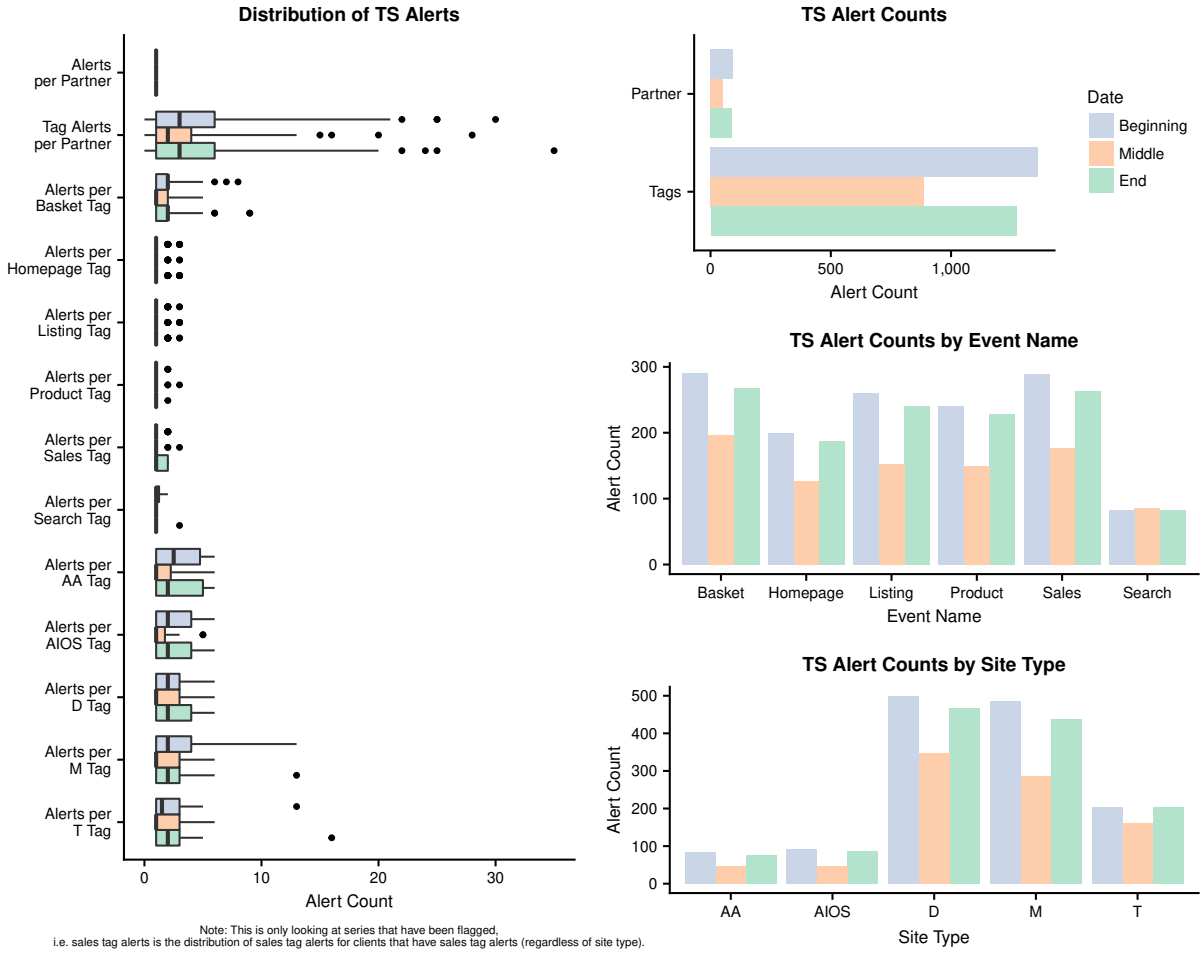


Figure 19: TS alert counts using Percent Change from the Median

Additionally, if we look at a histogram of the site events (Figure 20), we see that it is skewed, meaning that the median is a better measure of central tendency than the mean (which won't lie in the center in this case).

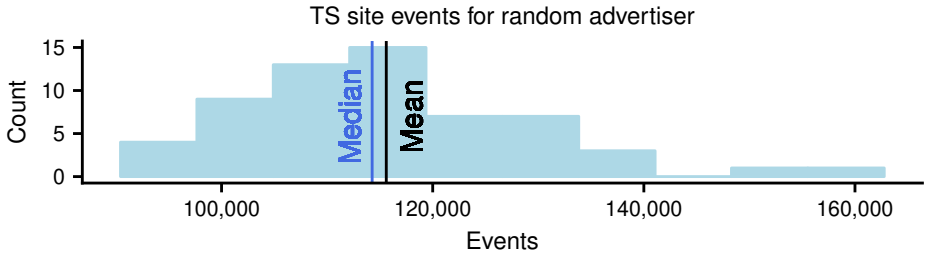


Figure 20: Histogram of site events

As with the AS results, our metrics are poor across the board. As shown in Table 23, we are flagging such a large number of cases that we hardly have any false negatives, but our false positives are quite high. `F1-score` is very low and accuracy is poor. We can do much better than this. We are performing best on basket and site-level, just like with the mean, no improvement here. With site type, site-level is the best performing, but no better than the mean.

Table 23: Prediction vs. Metis for TS alerts using Percent Change from the Median

Date	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Beginning	73	265	863	33	0.22	0.69	0.33	0.77	0.76
Middle	39	165	909	39	0.19	0.50	0.28	0.85	0.82
End	79	198	816	42	0.29	0.65	0.40	0.80	0.79
TOTAL	191	628	2,588	114	0.23	0.63	0.34	0.80	0.79

Metric	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Basket	38	129	363	10	0.23	0.79	0.35	0.74	0.74
Homepage	26	99	469	13	0.21	0.67	0.32	0.83	0.82
Listing	27	100	353	22	0.21	0.55	0.31	0.78	0.76
Product	37	94	530	22	0.28	0.63	0.39	0.85	0.83
Sales	23	125	442	18	0.16	0.56	0.24	0.78	0.76
Search	19	43	206	21	0.31	0.48	0.37	0.83	0.78
Site Level	21	38	225	8	0.36	0.72	0.48	0.86	0.84

Metric	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
AA	14	42	135	9	0.25	0.61	0.35	0.76	0.74
AIOS	10	33	147	3	0.23	0.77	0.36	0.82	0.81
D	55	236	984	25	0.19	0.69	0.30	0.81	0.80
M	58	193	744	53	0.23	0.52	0.32	0.79	0.77
Site Level	21	38	225	8	0.36	0.72	0.48	0.86	0.84
T	33	86	353	16	0.28	0.67	0.39	0.80	0.79

Territory RexT Results

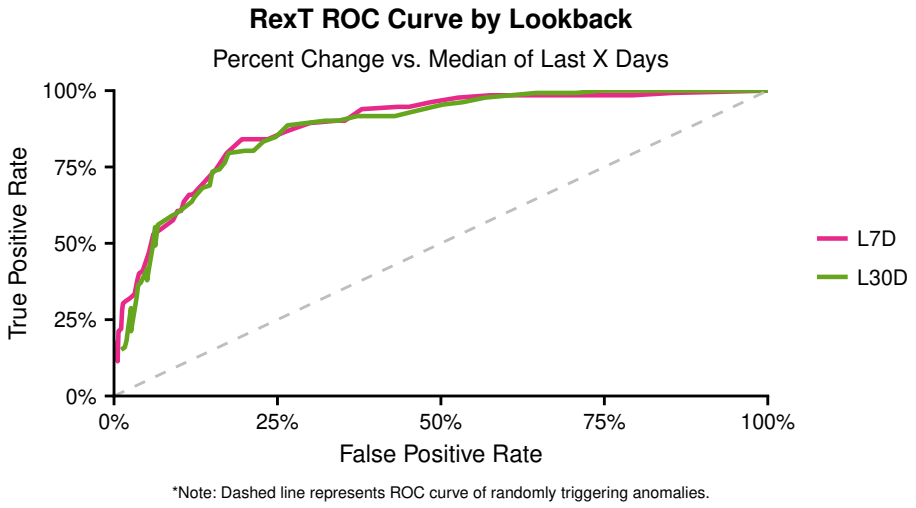


Figure 21: ROC curves for RexT alerts using percent change from the median at various lookbacks

Here, we only have a choice between 7 days and 30 days because the RexT alerts only look back 30 days. As

with the mean, when using the median, 7 days becomes the better lookback window. This is most likely due to recent values affecting the mean, while the median is robust to these changes.

Using the ROC curve data (Figure 21), I have identified the threshold of 19% by choosing a threshold with a false positive rate of 20%. This will be used to check if territory RexT has anomalies when compared to the average of the last 30 days prior. Notice how this threshold is lower than the AS metrics threshold of 57% and the TS threshold of 35% at the same false positive rate; further evidence that the territory RexT time series, being so aggregated, has a much lower threshold for anomalous behavior.

Percent change from the median is most sensitive at the country-level and least sensitive at the region-level; the more granular the series, the higher the volume of alerts. Unlike with the mean, we have the majority of the alerts in the beginning of the month, decreasing as the month goes on. This model flags Canada, the LATAM countries, and the subregion itself quite a bit—probably due to them being much smaller volume-wise, and thus, more prone to volatility. In the case of RexT data, we trigger more alerts using the median compared to the mean; additionally, we have more variation in alert counts (see Figure 22).

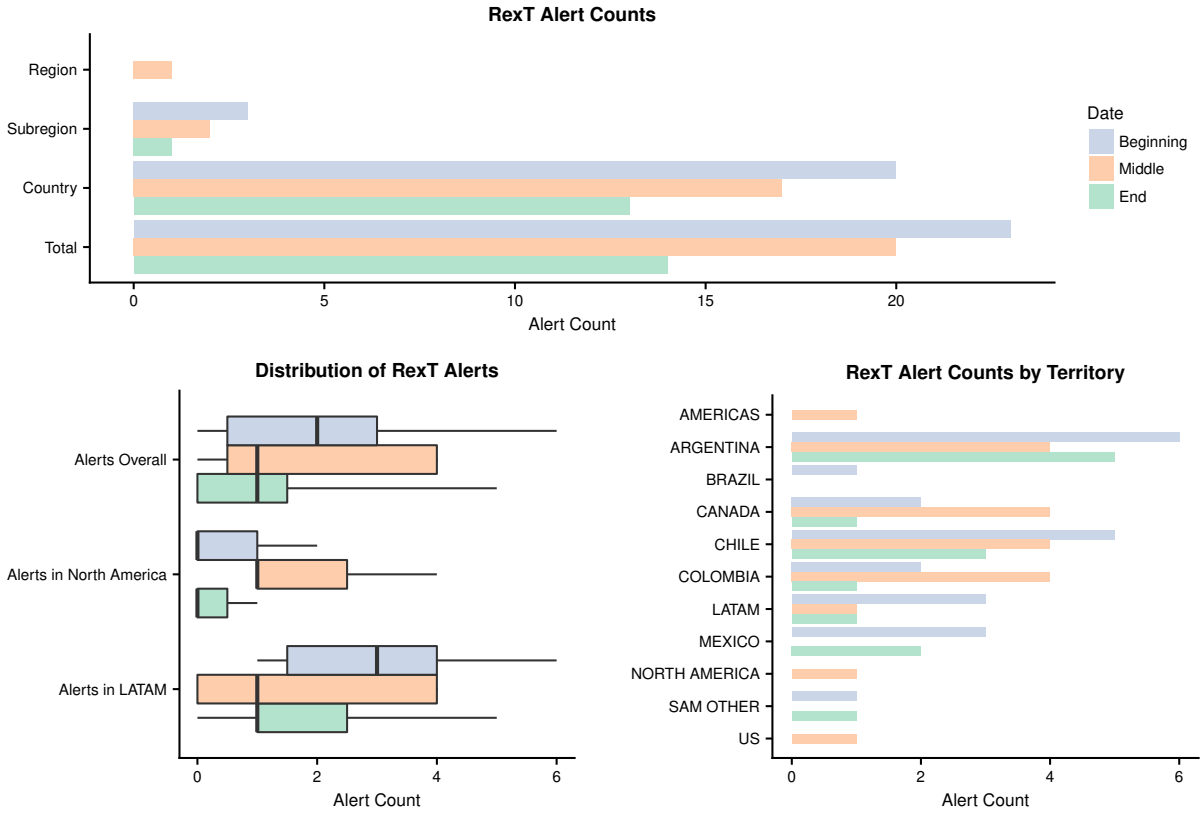


Figure 22: RexT alert counts using Percent Change from the Median

Just like with the mean, we perform significantly worse for the end of the month; we flagged very few during this period, however, it looks like the model wasn't catching things (see Table 26). Other than the poor performance for end-of-month, this model performs *much* better on this data compared to the AS and TS data sets. Though still not something we would want to use in practice, as the scores are still not great, we can see that the more aggregated the data gets, the better this method performs. This model performs perfectly for the Americas and Brazil and pretty well on Mexico; the other territories are all over the place. Notice that only Brazil worked perfectly on the mean and median methods. Overall, the median and mean are pretty equivalent, but the median is slightly better. Keep in mind that this isn't a huge sample size when we go down to the territory granularity, so take this with a grain of salt.

Table 26: Prediction vs. Metis for territory RexT alerts using Percent Change from the Median

Date	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Beginning	10	13	43	0	0.43	1.00	0.61	0.77	0.80
Middle	9	11	43	3	0.45	0.75	0.56	0.80	0.79
End	3	11	52	0	0.21	1.00	0.35	0.83	0.83
TOTAL	22	35	138	3	0.39	0.88	0.54	0.80	0.81

Territory	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
AMERICAS	1	0	17	0	1.00	1.00	1.00	1.00	1.00
ARGENTINA	4	11	3	0	0.27	1.00	0.42	0.21	0.39
BRAZIL	1	0	17	0	1.00	1.00	1.00	1.00	1.00
CANADA	2	5	11	0	0.29	1.00	0.44	0.69	0.72
CHILE	5	7	5	1	0.42	0.83	0.56	0.42	0.56
COLOMBIA	3	4	11	0	0.43	1.00	0.60	0.73	0.78
LATAM	2	3	13	0	0.40	1.00	0.57	0.81	0.83
MEXICO	3	2	13	0	0.60	1.00	0.75	0.87	0.89
NORTH AMERICA	1	0	16	1	1.00	0.50	0.67	1.00	0.94
SAM OTHER	0	2	16	0	0.00	NA	NA	0.89	0.89
US	0	1	16	1	0.00	0.00	NaN	0.94	0.89

ROC Curve

As with the mean, this method performs worst on AS and best on territory RexT (see ROC curves in Figure 23). Percent change metrics won't work for the granularity we need in our alerting system.

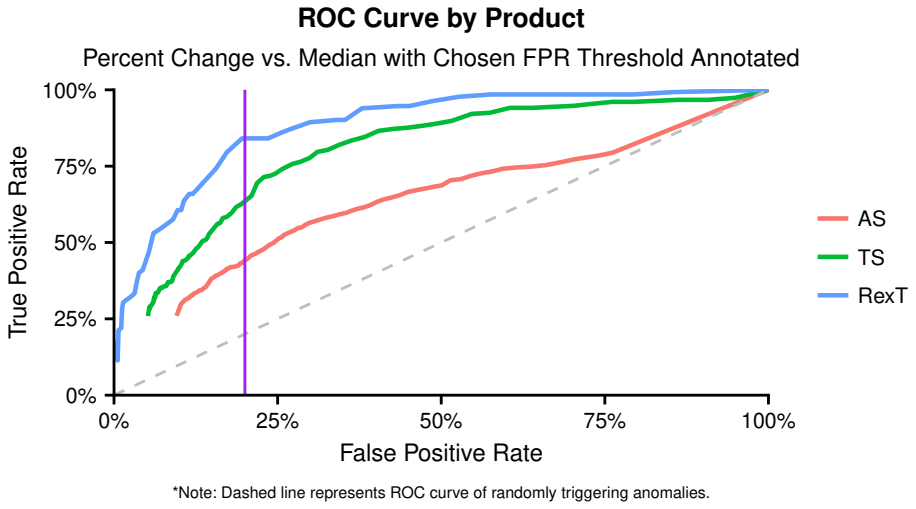


Figure 23: ROC curves for percent change from the median at optimal thresholds

Using the ROC curves in Figure 24, we can see whether the mean or median performs best for each of our data types; the curve with the most area underneath it is better (although not necessarily optimal). For the AS metrics, somewhat surprisingly, the mean is better than the median. There isn't much of a difference for the TS metrics. The median just edges out the mean for the territory RexT data.

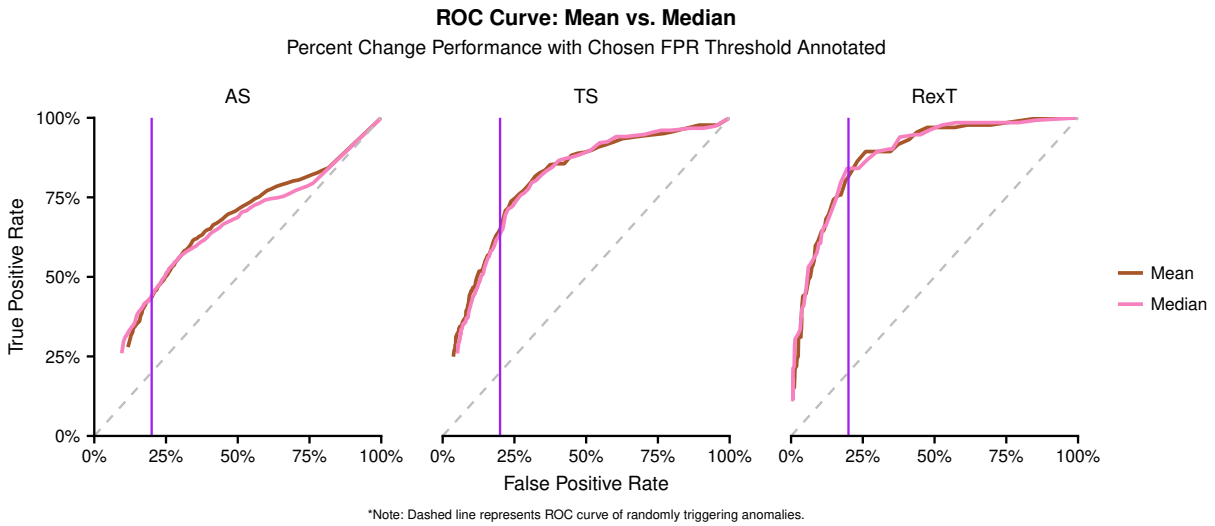


Figure 24: Mean vs. Median ROC curves by SCOPE product

These findings agree with the conclusion that the more aggregated the data is the better off you are using the median and, with more granular data, the mean. The AS data is the most granular and the RexT the most aggregated with the TS in the middle. Also, notice that while I noted in several places, for both the mean and the median, that this method is too sensitive, we could have changed our threshold by lowering our tolerance for false positives. To do so, we move the purple line in the graphs to the left, but observe the huge drop-offs we would have in true positive rates! Our optimal ROC curve would not have such a large drop-off at a relatively large false positive rate; the optimal ROC curve would be at 100% true positive rate at this point and most likely also at lower false positive rates (see Figure 3).

Alogrithm Complexity

As with the mean, this method is implemented as a vectorized operation, so it will run faster time-wise than the complexity dictates; however, for the complexity analysis we have to check each combination against the threshold which is equivalent to running it through a loop. Average complexity (Θ):

$$\Theta(n) = k * n \tag{10}$$

where:

- n = *number of series to inspect (i.e. total number of clients + total number of campaigns)*
- k = *number of KPIs to inspect (for each series)*

This gives us an overall linear complexity ($\Theta(n)$) since k should always be way less than n as well as a tight best-case of $\Omega(n)$ and tight worst-case of $O(n)$. For large n, we would be better off with a faster algorithm if one exists.

Conclusions

As with the mean, this model performed poorly on `precision`, `recall`, and `F1-score`; our accuracy and ROC curves could definitely be improved. We are successfully flagging a lot of alerts, but our false positives are pretty high. We have very few false negatives, but not because our model is great at capturing expected performance, but rather because we flag so many things as positive that the conditional probability of flagging negative and having it be wrong is low. Be it with the mean or median; lookback of 7, 30, or 60 days, percent change metrics won't work for the granularity we need in our alerting system.

Twitter's `AnomalyDetection` Package

Twitter's `AnomalyDetection` R package uses a Seasonal Hybrid ESD (S-H-ESD) which builds upon Rosner's Test (Generalized ESD) to determine outliers. Since Twitter built this algorithm to analyze their highly seasonal tweet data, the first step is a time series decomposition (see Figure 25). Each time series is broken into it's seasonal and trend components with a remainder being the vertical distance between the expected value ($seasonal + trend$) and the original data.

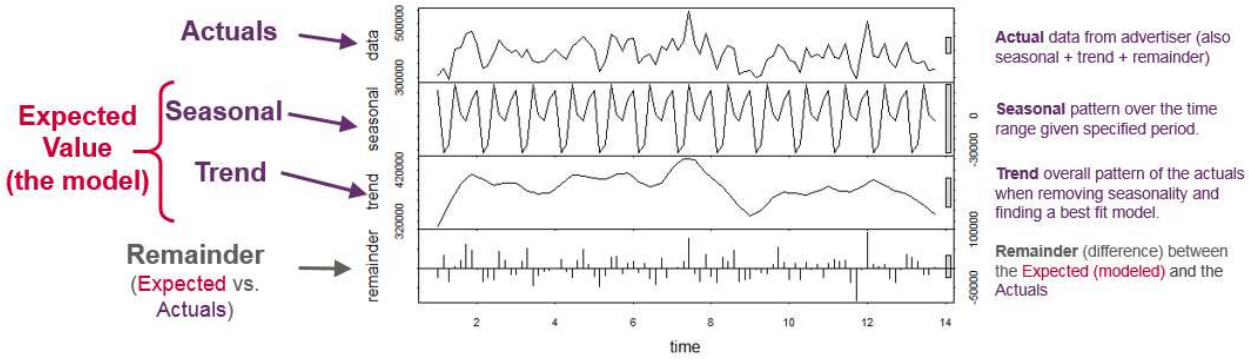


Figure 25: Time series decomposition

The remainder can best be visualized by comparing the actuals to the expected values (see Figure 26). Here, whenever the red and blue lines aren't overlapping, we have a remainder which can be positive or negative. *Note that this is not the data from the prior example.*

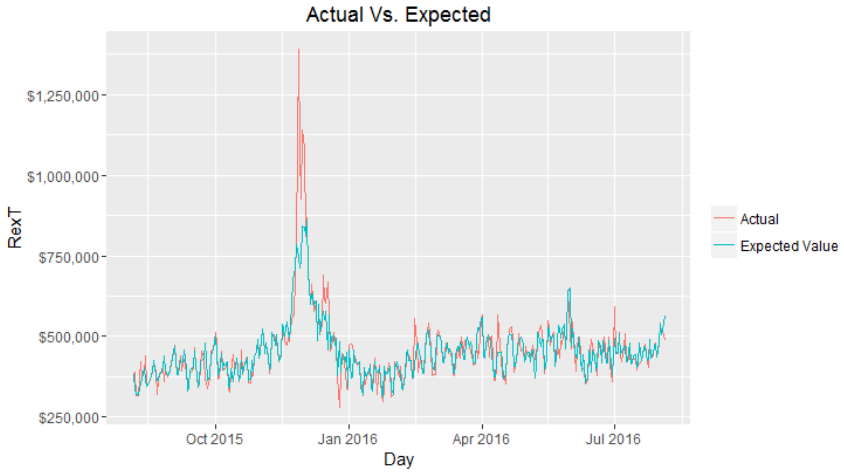
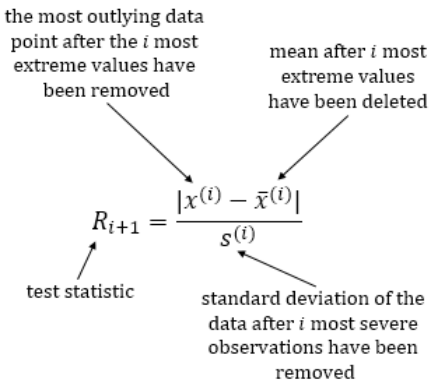


Figure 26: Visualizing the remainder

Rosner's test is then run on the remainders which envolves testing the significance of the test statistic (Equation 11) starting at the values furthest from the mean. This means that if the furthest from the mean is not significant (not considered an outlier), the series has no outliers and the algorithm stops.

To use this test effectively, your data should be roughly normal, and you should have at least 25 observations. For this case study, we will use 95% statistical significance and limit maximum outliers that can be detected in each time series to 1; this doesn't mean it will always flag one, but rather, that we can't have more than 1. Note that the Twitter algorithm requires you to provide a period to determine seasonality; we will look at forcing a 7-day period and picking a custom period for that combo using a Fourier transform to calculate the

strongest seasonality signal.[10]



(11)

This method was developed to work specifically with Twitter's data, however, they themselves acknowledge that since anomaly detection techniques are built around a specific domain, they can rarely be used as-is in other fields.[11] Furthermore, while this is quick for us to implement, we would be better off with a solution customized to our data.

Results

Quite counterintuitively, our data doesn't have a single strong seasonality signal. We have many competing seasonality signals, so this algorithm isn't able to capture seasonality in the way we think of it. Our data quickly finds new normals that buck the seasonality trend: we get incremental budget, the client increases bids for a flash sale, a high-traffic tag gets removed from a few pages of the site, etc. These are all times when our performance changes, and we stay at a new level for a while, making it difficult to determine the true seasonality. Site events data is a little better, but we have multiple levels of seasonality: weekly, monthly, yearly, etc. Despite our lack of a strong seasonality signal, forcing the 7-day period actually performs similar to the custom period, because it is most likely one of the seasonality signals for that time series, although perhaps, not the strongest. The territory RexT data, being highly aggregated, behaves better with the seasonality used in the Twitter model.

AS Results

The 7-day and custom periods disagree on 455 out of 47,076 series (1%). The fixed day period flagged 296 that weren't flagged by the custom period; the custom period flagged 159 series that the 7-day period didn't flag. (Keep in mind that we don't have data from `Metis` for every time series, so we will have more disagreements than data in our table when we match to what `Metis` has collected.)

Among the disagreements, when we look into what was predicted by the custom period, we have nearly equal true and false positives and we are triggering more negatives than positives. Our `precision`, `recall`, and `F1-score` are mediocre. `Specificity` and `accuracy` are dreadful (see Table 28).

Table 28: Prediction vs. Metis for AS alerts where custom and 7-day period disagreed, taking custom period

Date	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Beginning	1	1	2	8	0.5	0.11	0.18	0.67	0.25
Middle	4	4	4	8	0.5	0.33	0.40	0.50	0.40
End	7	3	1	7	0.7	0.50	0.58	0.25	0.44
TOTAL	12	8	7	23	0.6	0.34	0.44	0.47	0.38

[10]Detecting Seasonality using Fourier Transform in R
[11]Introducing practical and robust anomaly detection in a time series

If we take the side of the 7-day period as in Table 29, our `precision` and `recall` improve quite a bit—our `F1-score` improves as a consequence. `Specificity` and `accuracy` are better, but still something we should stay far away from. We have flipped everything here, so now we have a roughly even amount of true negatives and false negatives with more things marked positive in general; this begs the question—what is more important to us: minimizing false positives or false negatives?

Table 29: Prediction vs. Metis for AS alerts where custom and 7-day period disagreed, taking 7-day period

Date	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Beginning	8	2	1	1	0.80	0.89	0.84	0.33	0.75
Middle	8	4	4	4	0.67	0.67	0.67	0.50	0.60
End	7	1	3	7	0.88	0.50	0.64	0.75	0.56
TOTAL	23	7	8	12	0.77	0.66	0.71	0.53	0.62

For the AS metrics using the custom period, we have way more alerts for the beginning and end of the month compared to the middle of the month (see Figure 27). This model doesn't seem to understand seasonality like we do; even with a period custom to that time series, we aren't capturing the true nature of beginning/end of month being expectably more volatile. We are flagging more spend-related metrics beginning of month (spend, RexT—local and Euro); end of month we are flagging more conversion-related metrics (conversions, order value).

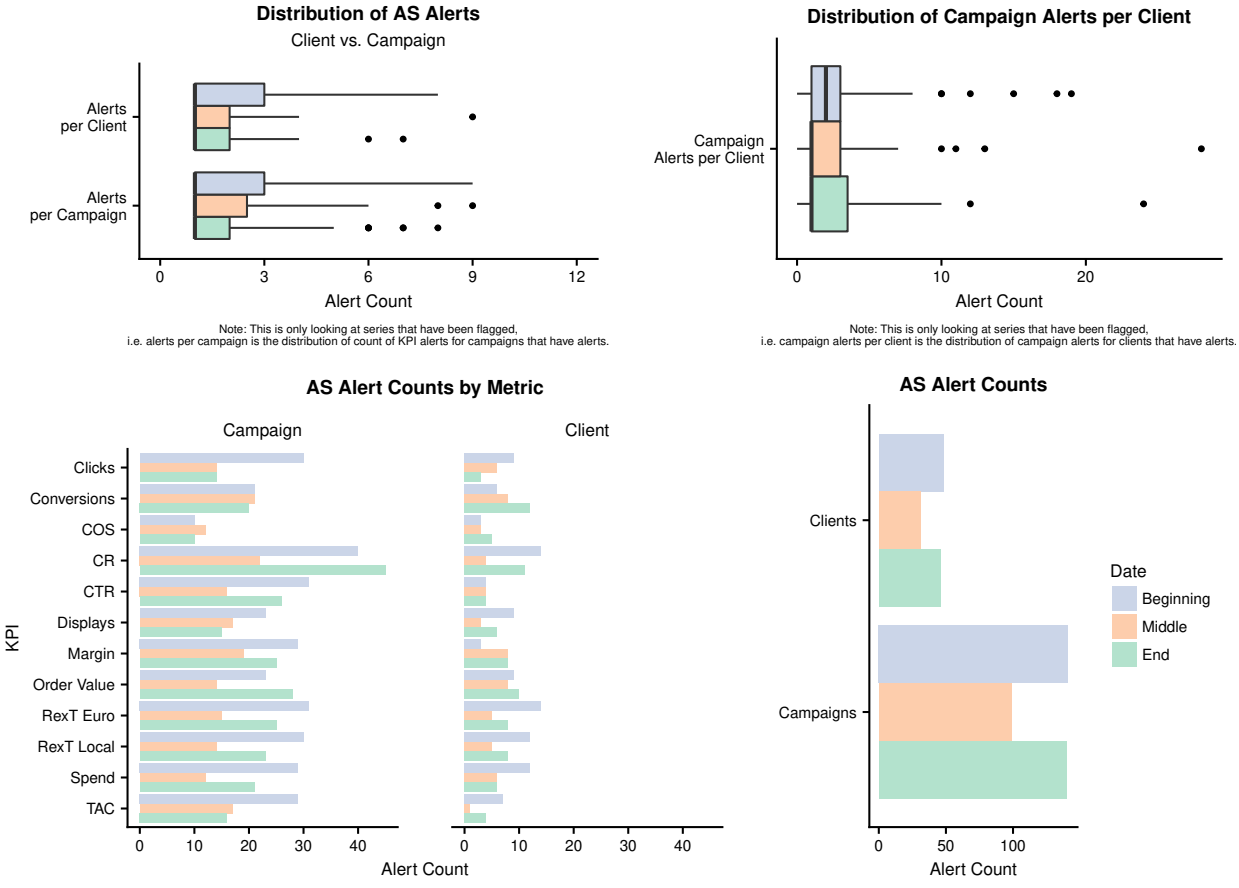


Figure 27: AS alert counts using Twitter AnomalyDetection and Custom Period

At the campaign-level, we have more alerts during the beginning/end of the month for each metric except COS and conversions (which peaks at the beginning of the month) compared to the middle of the month, but we also have many more clicks alerts for the beginning of the month. The model seems to have a different understanding of behavior at the client- vs. campaign-level. There is also a huge uptick in alerts on display volume. For most clients with alerts, we are triggering just 1 alert; the same is true for campaign alerts. Unlike the percent change methods, we see that our campaign alerts per client is, thankfully, well under 200.

Overall, the custom period has good `precision`, but the abysmal `recall` leads to an un-useable `F1-score`. With the custom period, our best metric by `F1-score` (TAC) is still below 0.5. We are looking to get near 1, so it's clear this is not going to work for these metrics. See Table 30 for breakdown.

Table 30: Prediction vs. Metis for AS alerts using Twitter AnomalyDetection with Custom Period

Date	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Beginning	55	5	1,267	335	0.92	0.14	0.24	1.00	0.80
Middle	22	8	1,321	296	0.73	0.07	0.13	0.99	0.82
End	30	8	1,292	309	0.79	0.09	0.16	0.99	0.81
TOTAL	107	21	3,880	940	0.84	0.10	0.18	0.99	0.81

Metric	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Clicks	2	0	161	86	1.00	0.02	0.04	1.00	0.65
Conversions	4	4	188	63	0.50	0.06	0.11	0.98	0.74
COS	2	1	202	84	0.67	0.02	0.04	1.00	0.71
CR	11	0	173	67	1.00	0.14	0.25	1.00	0.73
CTR	5	1	198	56	0.83	0.08	0.15	0.99	0.78
Displays	9	1	181	93	0.90	0.09	0.16	0.99	0.67
Margin	15	8	777	50	0.65	0.23	0.34	0.99	0.93
Order Value	8	0	186	77	1.00	0.09	0.17	1.00	0.72
RexT Euro	21	3	734	104	0.88	0.17	0.28	1.00	0.88
RexT Local	4	0	184	66	1.00	0.06	0.11	1.00	0.74
Spend	4	1	187	102	0.80	0.04	0.07	0.99	0.65
TAC	22	2	709	92	0.92	0.19	0.32	1.00	0.89

Using the fixed 7-day period with the AS data gives us more alerts across the board and larger variation in alert counts (see Figure 28). Here, we see the same pattern with spend metric alerts, although we now see similar alert levels on conversion-related metrics for beginning and end of month (conversions, CR), but we have replaced order value with CR. With the 7-day period at the campaign-level, we continue to see clicks jump out, but now we see that CR is triggered way more at the end of the month (more pronounced than with the custom period). We trigger slightly more campaign alerts per client with the fixed period; however, the overall distribution is pretty equivalent. We also get a slight improvement in `precision`, but unfortunately none in `recall` or `F1-score` (see Table 32). Our performance using `F1-score` is slightly more averaged for all metrics, however, it evens out to the same overall `F1-score` as the custom period—we just have less bad and less good `F1-score`'s for the metrics.

Table 32: Prediction vs. Metis for AS alerts using Twitter AnomalyDetection with Fixed 7-day Period

Date	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Beginning	62	6	1,266	328	0.91	0.16	0.27	1.00	0.80
Middle	26	8	1,321	292	0.76	0.08	0.15	0.99	0.82
End	30	6	1,294	309	0.83	0.09	0.16	1.00	0.81
TOTAL	118	20	3,881	929	0.86	0.11	0.20	0.99	0.81

Metric	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Clicks	3	1	160	85	0.75	0.03	0.07	0.99	0.65
Conversions	6	3	189	61	0.67	0.09	0.16	0.98	0.75
COS	3	0	203	83	1.00	0.03	0.07	1.00	0.71
CR	11	0	173	67	1.00	0.14	0.25	1.00	0.73
CTR	6	1	198	55	0.86	0.10	0.18	0.99	0.78
Displays	6	1	181	96	0.86	0.06	0.11	0.99	0.66
Margin	14	7	778	51	0.67	0.22	0.33	0.99	0.93
Order Value	7	0	186	78	1.00	0.08	0.15	1.00	0.71
RexT Euro	23	5	732	102	0.82	0.18	0.30	0.99	0.88
RexT Local	5	0	184	65	1.00	0.07	0.13	1.00	0.74
Spend	7	0	188	99	1.00	0.07	0.12	1.00	0.66
TAC	27	2	709	87	0.93	0.24	0.38	1.00	0.89

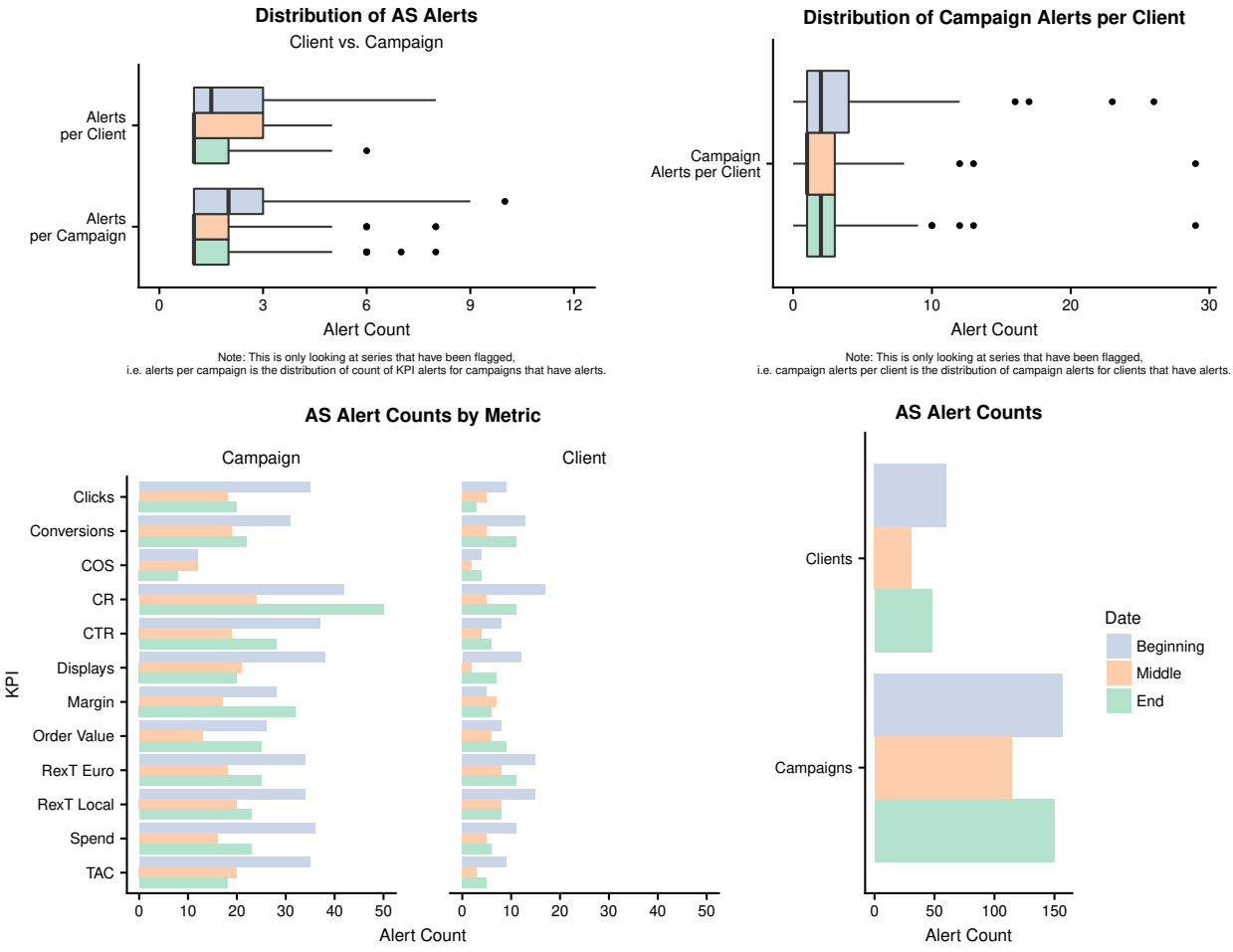


Figure 28: AS alert counts using Twitter AnomalyDetection and Fixed 7-day Period

TS Results

Both the custom period and the fixed 7-day period versions of model yield the same results, so I will only walk through one set this time, but know that they are the same. This is most likely due to the highly seasonal nature of the TS alerts. Many have multiple seasonal trends (7D, 14D, 21D, 30D, 60D, etc.), so

forcing it to be 7D, while it might not be the strongest of the seasonal signals, it is still a good estimate of the true period, meaning that this assumption doesn't harm our results.

We have many more alerts for beginning and end of month, especially, when we look at the tag level. Even though we are taking into account seasonality, this model doesn't seem to recognize that middle of the month tends to behave differently than beginning/end of month. Alerts are spread pretty evenly across event name. However, we see double the amount of desktop `site_type` alerts at the end of the month compared to the other time periods. Not sure why this could be, but interesting to note, nevertheless. Using the boxplots in Figure 29, we see that for nearly every alert count breakout listed, we have a median of one alert triggered; the only exception is the tablet tag. This means that the Twitter model tends to only trigger 1 alert per event name and 1 alert per site type, unlike what we saw for the percent change methods.

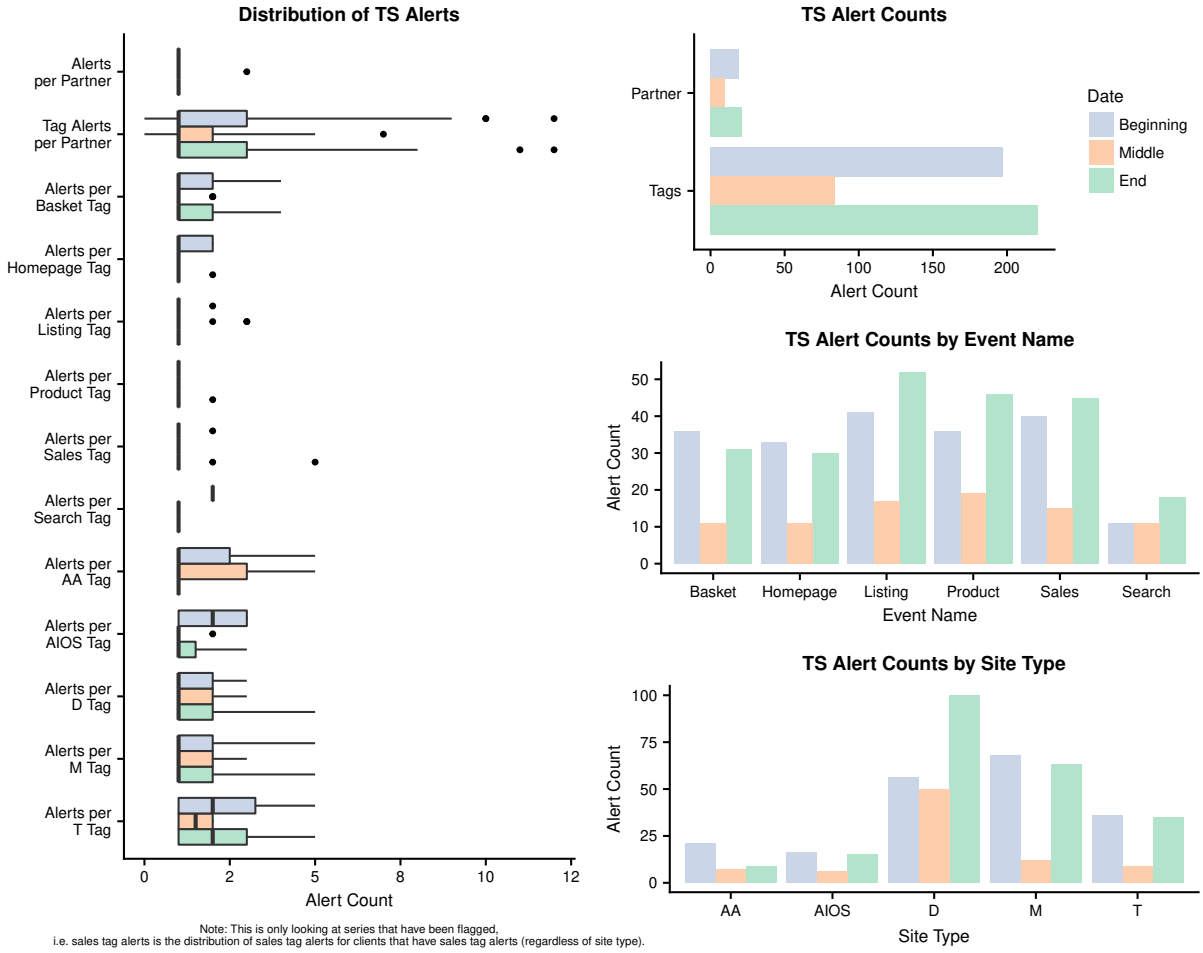


Figure 29: TS alert counts using Twitter AnomalyDetection

Our `F1-score` is horrible for TS metrics as evidenced in Table 34. Even though both the custom and fixed periods agreed on everything, neither agreed with the `SCOPE` userbase. We have poor `precision` and `recall`, to boot, with our worst performance at the middle of the month. While our best performance is at the site level, we have low `F1-score`'s across the board at the `event_name` level. Things are pretty awful at the `site_type` level as well. Site-level is again one of the better performers; this leads me to believe that this model is yet another one that is better on aggregated data.

Table 34: Prediction vs. Metis for TS alerts using Twitter AnomalyDetection with Fixed 7-day Period

Date	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Beginning	28	26	1,121	76	0.52	0.27	0.35	0.98	0.92
Middle	13	10	1,074	66	0.57	0.16	0.25	0.99	0.93
End	25	21	1,030	94	0.54	0.21	0.30	0.98	0.90
TOTAL	66	57	3,225	236	0.54	0.22	0.31	0.98	0.92

Metric	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Basket	8	8	494	41	0.50	0.16	0.25	0.98	0.91
Homepage	9	5	573	30	0.64	0.23	0.34	0.99	0.94
Listing	13	14	442	35	0.48	0.27	0.35	0.97	0.90
Product	13	9	629	45	0.59	0.22	0.32	0.99	0.92
Sales	6	11	572	33	0.35	0.15	0.21	0.98	0.93
Search	7	5	253	32	0.58	0.18	0.27	0.98	0.88
Site Level	10	5	262	20	0.67	0.33	0.44	0.98	0.92

Metric	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
AA	4	6	182	18	0.40	0.18	0.25	0.97	0.89
AIOS	5	5	191	8	0.50	0.38	0.43	0.97	0.94
D	26	20	1,218	54	0.57	0.32	0.41	0.98	0.94
M	10	18	936	99	0.36	0.09	0.15	0.98	0.89
Site Level	10	5	262	20	0.67	0.33	0.44	0.98	0.92
T	11	3	436	37	0.79	0.23	0.35	0.99	0.92

Territory RexT Results

The 7-day and custom periods disagree on 12 out of 198 series (6%). The fixed day period flagged 9 that weren't flagged by the custom period; the custom period flagged 3 series that the 7-day period didn't flag.

The confusion matrix metrics for using the custom period classification on series where the 7-day period and the custom period disagreed on whether it was an alert or not are all pretty low. See Table 37 for the breakdown.

Table 37: Prediction vs. Metis for territory RexT alerts where methods disagreed, taking custom period

Date	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Beginning	1	0	1	1	1.00	0.50	0.67	1.00	0.67
Middle	1	0	1	4	1.00	0.20	0.33	1.00	0.33
End	0	1	2	0	0.00	NA	NA	0.67	0.67
TOTAL	2	1	4	5	0.67	0.29	0.40	0.80	0.50

Taking the 7-day period (in Table 38), the `F1-score` is **much** higher but our `precision`, `specificity`, and `accuracy` are still low. (We flipped the metrics we were looking at when you used the custom period's classifications.) It is clear that by choosing either one over the other that we aren't getting to an optimal, but we are catching more true positives and less false negatives when we use the 7-day period; we just have to deal with the uptick in false positives. This once again brings up the question—what is more important to us: minimizing false positives or false negatives?

Table 38: Prediction vs. Metis for territory RexT alerts where methods disagreed, taking 7-day period

Date	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Beginning	1	1	0	1	0.50	0.50	0.50	0.00	0.33
Middle	4	1	0	1	0.80	0.80	0.80	0.00	0.67
End	0	2	1	0	0.00	NA	NA	0.33	0.33
TOTAL	5	4	1	2	0.56	0.71	0.63	0.20	0.50

Even though we are using seasonality, it seems that with a custom period, we are flagging more alerts on the beginning and end of the month. Using custom period seems to ignore the relationship between the most aggregated series and its component series. For example, since US is such a large portion of North America, we would expect any alerts on the US to have a corresponding alert in North America and vice versa. Other than that, there is no real pattern here. Additionally, Figure 30 shows us that we have the largest variation in the middle of the month with North America regions (North America, US, Canada) while LATAM (LATAM, Brazil, Argentina, Chile, Colombia, Mexico, and SAM Other) is the most variant overall—makes sense because they have smaller and more volatile regions.

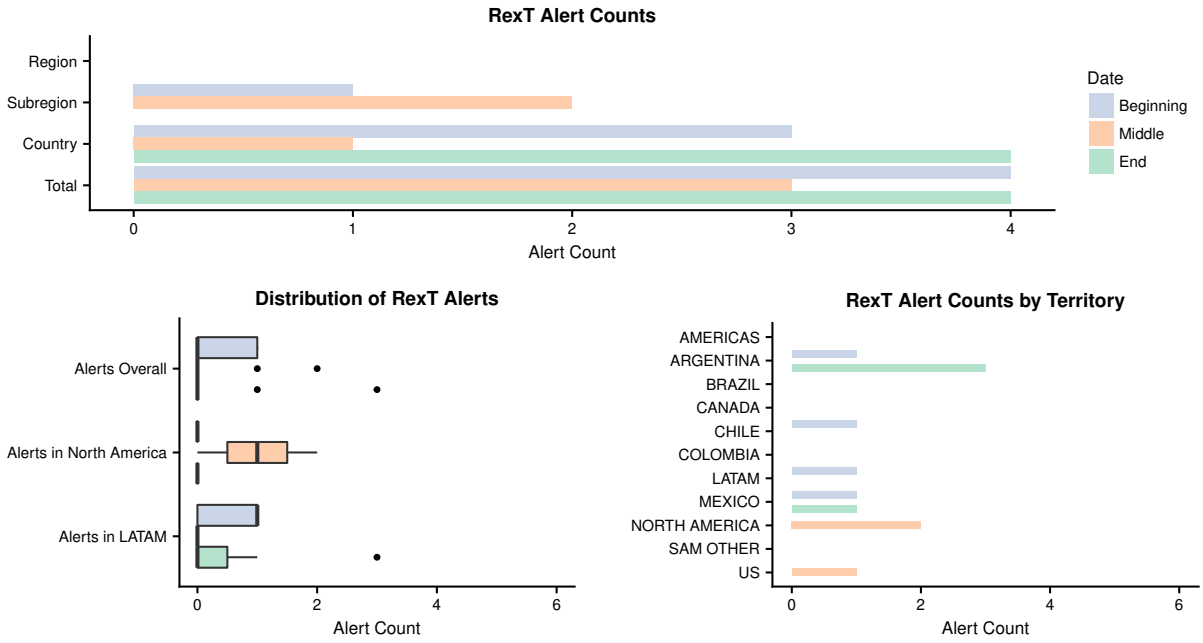


Figure 30: RexT alert counts using Twitter AnomalyDetection with Custom Period

With the custom period (results in Table 39), our `precision` is very high, however our low `recall` gives us a low `F1-score`. `Precision` is perfect for some of the regions when using the custom period, however, very few of those also have good `recall` meaning our `F1-score` is run-of-the-mill.

Table 39: Prediction vs. Metis for territory RexT alerts using Twitter AnomalyDetection with Custom Period

Date	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Beginning	4	0	56	6	1.00	0.40	0.57	1.00	0.91
Middle	3	0	54	9	1.00	0.25	0.40	1.00	0.86
End	3	1	62	0	0.75	1.00	0.86	0.98	0.98
TOTAL	10	1	172	15	0.91	0.40	0.56	0.99	0.92

Territory	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
AMERICAS	0	0	17	1	NA	0.00	NA	1.00	0.94
ARGENTINA	3	1	13	1	0.75	0.75	0.75	0.93	0.89
BRAZIL	0	0	17	1	NA	0.00	NA	1.00	0.94
CANADA	0	0	16	2	NA	0.00	NA	1.00	0.89
CHILE	1	0	12	5	1.00	0.17	0.29	1.00	0.72
COLOMBIA	0	0	15	3	NA	0.00	NA	1.00	0.83
LATAM	1	0	16	1	1.00	0.50	0.67	1.00	0.94
MEXICO	2	0	15	1	1.00	0.67	0.80	1.00	0.94
NORTH AMERICA	2	0	16	0	1.00	1.00	1.00	1.00	1.00
SAM OTHER	0	0	18	0	NA	NA	NA	1.00	1.00
US	1	0	17	0	1.00	1.00	1.00	1.00	1.00

When we look at the fixed period, we actually have more alerts during the middle of month! Here, we seem to have a better correlation between US - North America - Americas alerts, and we have more alerts spread out among the territories. Using the boxplot in Figure 31, we can observe that the fixed period has more variation overall in the number of alerts across the territories compared to the custom period. We now have variation in the Americas and LATAM regions for all dates looked at, but North America has gotten less variant (all the component regions were triggered the same number of times over the same date ranges).

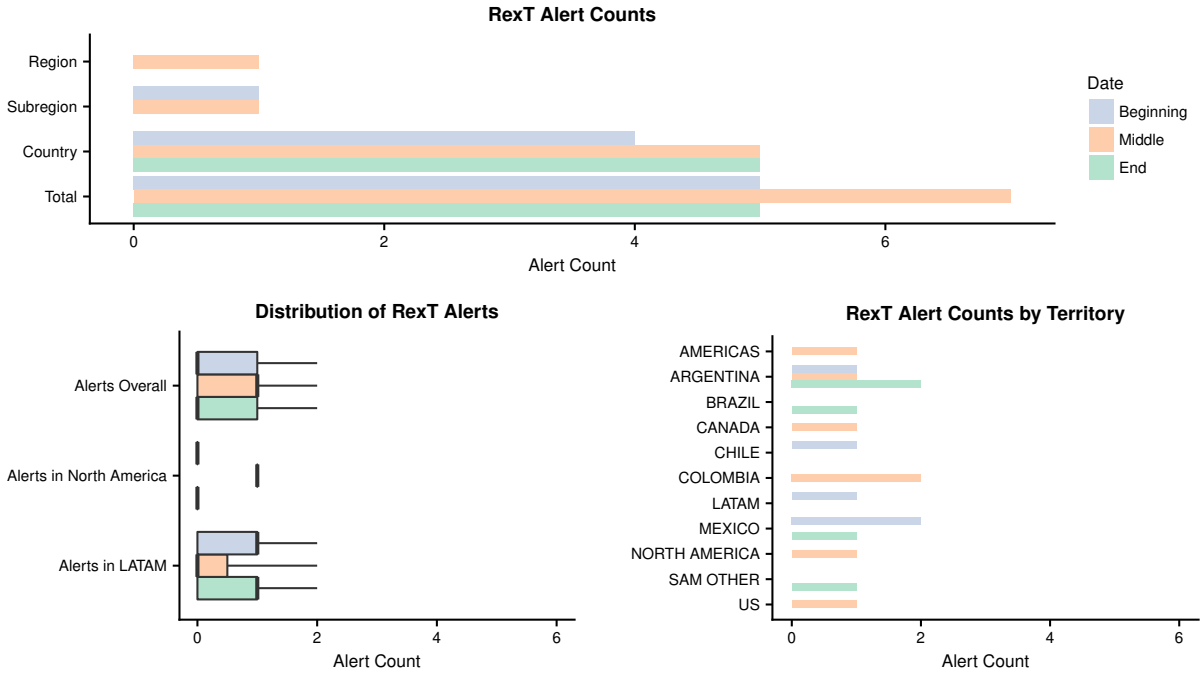


Figure 31: RexT alert counts using Twitter AnomalyDetection with Fixed 7-day Period

As seen in Table 41, we have a higher amount of false positives (and model positives overall) with the fixed period, however, our `F1-score` is slightly better than with the custom period since we improved our `recall` while not lowering our `precision` too much. `Accuracy` and `specificity` are the same for both period methods. Using the 7-day fixed period, we have even more regions with perfect `precision` *and* our `recall` is better; we even have perfect `F1-score` for the US! It's clear that despite our intuition, the fixed-period performs better for region-level data.

Table 41: Prediction vs. Metis for territory RexT alerts using Twitter AnomalyDetection with Fixed Period

Date	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Beginning	4	1	55	6	0.80	0.40	0.53	0.98	0.89
Middle	6	1	53	6	0.86	0.50	0.63	0.98	0.89
End	3	2	61	0	0.60	1.00	0.75	0.97	0.97
TOTAL	13	4	169	12	0.76	0.52	0.62	0.98	0.92

Territory	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
AMERICAS	0	1	16	1	0.00	0.00	NaN	0.94	0.89
ARGENTINA	4	0	14	0	1.00	1.00	1.00	1.00	1.00
BRAZIL	0	1	16	1	0.00	0.00	NaN	0.94	0.89
CANADA	1	0	16	1	1.00	0.50	0.67	1.00	0.94
CHILE	1	0	12	5	1.00	0.17	0.29	1.00	0.72
COLOMBIA	2	0	15	1	1.00	0.67	0.80	1.00	0.94
LATAM	1	0	16	1	1.00	0.50	0.67	1.00	0.94
MEXICO	2	1	14	1	0.67	0.67	0.67	0.93	0.89
NORTH AMERICA	1	0	16	1	1.00	0.50	0.67	1.00	0.94
SAM OTHER	0	1	17	0	0.00	NA	NA	0.94	0.94
US	1	0	17	0	1.00	1.00	1.00	1.00	1.00

ROC Curve

Since this model yields a `TRUE/FALSE` answer to whether a given point is an anomaly, we don't have a threshold we can alter and will provide the ROC "points" instead (Figure 32). It is easy to see how we became less sensitive and more relevant by implementing this model after using percent change from the mean.

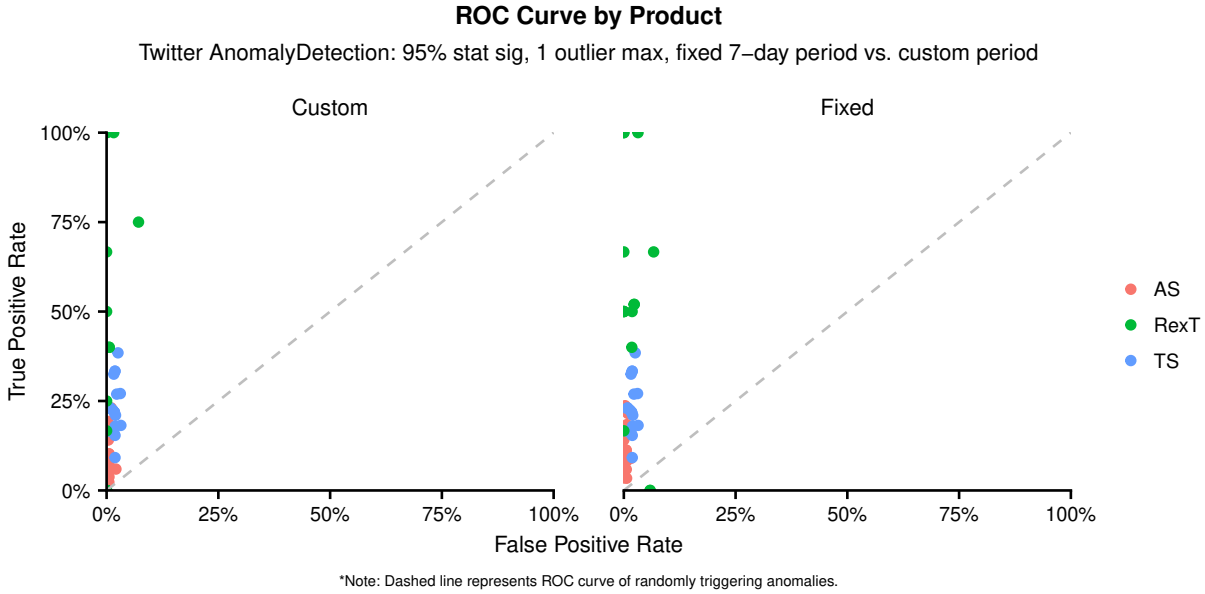


Figure 32: ROC curve by product using Twitter AnomalyDetection comparing custom and 7-day fixed periods

Our false positive rate is well below the rate we chose when selecting our percent change threshold in the prior models. However, with the exception of the RexT alerts we don't seem to be doing much (if at all) better than the percent change models at the same false positive rate. This suggests that none of the models presented thus far (percent change variations or Twitter algorithm variations) are optimal for AS/TS alerts. Once again, we have a model that performs much better on the aggregated data. By taking seasonality into account, we have significantly improved our `precision`, however, our `recall` has suffered, and therefore, this model acheives a *lower* `F1-score` than the percent change models for AS and TS metrics. Somewhat surprisingly, we don't seem to gain anything by using the custom period for each time series. The 7-day seems to work the same for AS and TS and better for the RexT data. The extra calculation needed to determine the period for each series is costly, so it's good to know that it doesn't really make a difference, and we can use a fixed value instead.

Alogrithm Complexity

While the custom period brings this to a 3rd-degree (!) polynomial, it's performance isn't really any better than using the fixed 7-day period (the two methods shared a very large overlap in alerts), so we would be doing ourselves a disservice to use the custom period. As such, we will go over the complexity of the fixed period. Rosner's test will stop searching once the current point we are evaluating turns out not to be an outlier, and since we are only allowing 1 outlier per time series, the algorithm will only check the most devious point; this means, per a specific client and KPI, we have a complexity of 1, so we just need to count the loops needed to get to that point. Therefore, average complexity (Θ) is:

$$\Theta(n) = (k * n)^2 \tag{12}$$

where:

- n = *number of series to inspect (i.e. total number of clients + total number of campaigns)*
- k = *number of KPIs to inspect (for each series)*

This gives us an overall quadratic complexity ($\Theta(n^2)$) since k should always be way less than n as well as a tight best-case of $\Omega(n^2)$ and tight worst-case of $O(n^2)$. For small n, this won't be awful, but for large n, we will quickly eclipse algorithms of lower complexities. It behooves us to use a faster algorithm if we hope to expand `SCOPE`.

Conclusions

In switching from the percent change model to the Twitter algorithm, we reduced our false positives and improved our `precision`. However, since we went for a very low false positive rate, we ended up with poor `recall` (worse than percent change) for AS and TS metrics as we failed to flag the more minor alerts. Due to our lack of complete data when we implemented this—we only knew about true and false positives, but nothing about the negatives—we made the choice to reduce false positives, however, now that we have complete information, we are better off finding the balance between reducing false positives and false negatives; this is part of the reason we are using the `F1-score` as a main metric. Using the `F1-score`, the Twitter method is only better than percent change for territory RexT, where it improves the `precision` and the `F1-score`. We once again have an algorithm that performs best on our most aggregated data, but with a much higher algorithmic complexity that will led to significantly larger compute times when we increase our scope.

Furthermore, this model doesn't seem to understand seasonality like we do. In fact, when left to its own devices, it can't find a single strong seasonality signal. We have many possible signals that have similar strengths; I had to write an extra function to allow the use of a custom period (the strongest among the weak, if you will). This suggests that although our data is seasonal to us; it is, in fact, pretty random, especially when seen through the eyes of a machine. In particular, AS data is very difficult to get a strong seasonality signal out of due to fluctuations in performance and budgets being common over the 60 day period; TS often has several, but none being the dominant by far; territory RexT is not as erratic as the AS data, but it's still

less obvious what the period is truely. The Twitter algorithm is designed for highly-seasonal, very granular data, seconds (or milliseconds) apart. We have granular data days apart, and this mismatch appears to be preventing this model from being helpful for us.

Note that when this model was in use, it was thought to be way too sensitive, as well—although, a huge improvement over the percent change. Changing to this model was enough to convince AS team leaders that we could release `SCOPE` to their whole team and move off of the beta-testers. When we used this model, we were allowing more outliers to be present in the time series, so we were capturing local and global outliers (I'm not strictly calling them min or max because the data gets transformed, so it wasn't the min/max when looking at the graph of the series), whereas here, we just capture the global; we, thus, have many fewer alerts than we triggered using Twitter's `AnomalyDetection` package in production. Also, note that users often reported that this was too sensitive; however, when collecting data with `Metis`, I noticed that users were marking as alerts series that didn't seem too extreme (stuff they complained was being flagged because the model was too sensitive before); this begs the question: did users complain that the model was too sensitive because of the old email layout providing tons of information per alert or because the actual alert was irrelevant?

Current Model and Other Rules-based Models

Now that we understand the motivation to reduce sensitivity, we are going to dive into the current model being used from May 16, 2017 - June 8, 2018 (the time of finalizing this case study and my last day here so I can't speak to the future usage) along with any other rules-based models that are found while researching this paper. Although the purpose of this case study is to look through models we have used and then find a machine learning alternative, I wanted to have a section for other rules-based models that we have not used, yet could be interesting for this study and to help establish the baseline performance we are looking to beat with the machine learning alternatives.

Chi-Square + Tukey Fence + 10% More Extreme Check

SCOPE moved to this model on May 16, 2017 after calculating `precision` for this method and the Twitter method we looked at earlier. We were looking for something to drastically reduce our sensitivity while still capturing extreme values as we built the `Metis` web app to collect labelled data and eventually enable us to pursue a machine learning method of identifying accounts with issues. It is important to note that while we picked the model that maximized `precision`, since we had no insight into false negatives, we were unable to calculate `recall` at the time, and therefore will most likely have increased our false negatives due to lack of penalization, as with the `F1-score`. Here, I will be evaluating this model with all the data we need to calculate these metrics, and thus, will have a more accurate view of its performance.

Chi-Square

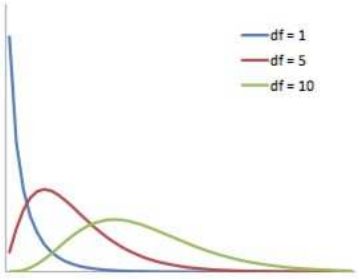


Figure 33: Chi-square distribution at various degrees of freedom.

First, a test statistic is calculated for each time series to be checked for anomalies (i.e. each metric for each client has its own test statistic). In doing so, we put the data on standard scale (this prevents the magnitude of the changes from causing alerts even if when looking at the whole series the change doesn't seem large). The test statistic is defined as:

$$\frac{x - mean(x)^2}{var(x)} \tag{13}$$

Then, I calculate the critical value using the Chi-square distribution (Figure 33) at a given statistical significance threshold (depending on metric—see Table 43). If the value of the test statistic is greater than this (meaning it is outside what we would expect to observe), we trigger an alert.

Table 43: Statistical significance levels used for Chi-Square test.

Metric	Stat Sig
COS	99.95%
CR	99.90%
CTR	99.90%
Events (Site/Tag)	97.50%
RexT (Client/Euro)	99.75%
RexT (Territory)	97.50%
Spend	99.75%
Other (Margin, Conversions, etc.)	95.00%

Tukey Fence

In addition to the Chi-Square test, we check that the value is outside the bounds of the Tukey fence, with lower bound of $Q_1 - 3 * IQR$ and upper bound of $Q_3 + 3 * IQR$ where Q_1 is the first quartile, Q_3 is the third quartile and $IQR = Q_3 - Q_1$ (see Figure 34). If yesterday's value is above the upper bound or below the lower bound, it is an alert.

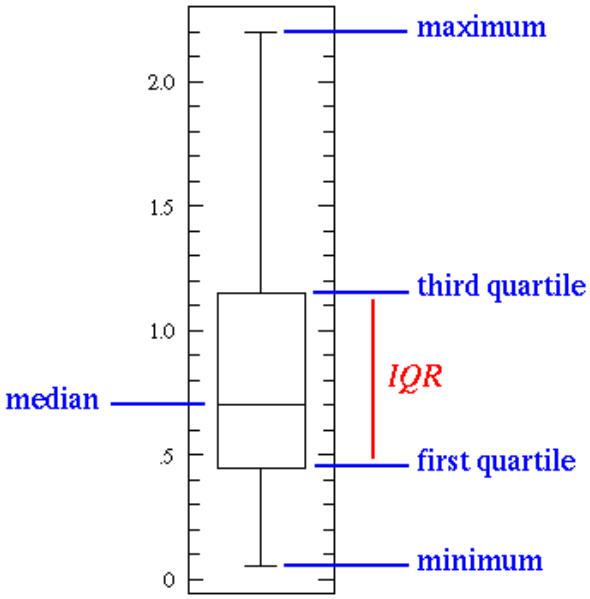


Figure 34: Boxplot showing how values of the Tukey fence are calculated.

Value Has Gotten More Extreme

I also require that yesterday's value is 10% more extreme than the day before, meaning that if we are below the lower bound on the Tukey fence, we check that yesterday's value is less than or equal to 90% of the day before's (it has decreased further); if we are above the upper bound on the Tukey fence, we check that yesterday's value is greater than or equal to 110% of the day before's (it has increased further).

Results

Before going into the results on each of the `SCOPE` products, we should note that this model lacks a seasonality component. As such, we may be wrongly classifiying series when seasonality explains the fluctuation. We have feedback from several users for each product, but they all look for different patterns as it relates to their job philosophy; this makes finding an ideal model difficult. When dealing with time series data, not using a seasonality component seems foolish, however, we need to make sure we use it properly otherwise it doesn't perform better than an approach such as this one.

Furthermore, this model was *purposely* designed to be less sensitive than its predecessors. We were looking to quickly lower the amount of irrelevant alerts so that the flagged alerts would carry more weight with `SCOPE` users. Naturally, since we maximized the only metric we could calculate at the time (`precision`), we have introduced many more false negatives; hence, the low `recall` overall.

We can expect there to be differences between the territory RexT series and the AS/TS series due to volume. We have much more data to work with for the AS/TS series and those are naturally more volatile, while the territory RexT series, being aggregates, will smooth out the underlying issues and shrink the variance. This leads us to have very few "alerts" to look at.

AS Results

Since our model doesn't take seasonality into account, we trigger more alerts during the beginning and end of the month compared to the middle; however, our false negatives are spread evenly across the dates studied. Some of these "alerts" can be explained by seasonality patterns, and, therefore, get marked as false positives. We have become much less sensitive and increased our false negatives; this gives us poor `recall` and consequently a low `F1-score`. We see that, for the most part, if a campaign or client has an alert, it is just for one metric. We seem to trigger many more alerts for beginning and end of month compared to middle of the month on spend and performance-related metrics. This is due to our model not using seasonality. Many advertisers have monthly budgets, if they pace too hot in the beginning of the month, they have to pull back and may run out at the end. These spikes and dips in spend can trigger alerts across several metrics. This trend is even more apparent when looking at the campaign-level alerts.

Using the boxplots in Figure 35, we can see that 50% of the time we trigger 1-3 alerts per campaign and 1-2 alerts per client (this is 1-3 for beginning of the month). The median number of alerts is higher per campaign than per client due to the higher volatility at the campaign-level. It's interesting to note that, for this method, we have more volatility during the middle of the month compared to the end of the month. Looking at the campaign alerts per client, we see a huge departure from what we saw in the earlier methods: we have no clients with triple digit campaign alert counts! For our clients with alerts, we tend to trigger more during the beginning and end of the month, but we still expect 1-2 alerts. At the campaign-level, we tend to get 1 more alert during the beginning of the month compared to the middle and end, but we still tend to stay at 3 or less alerts per campaign.

Looking at the daily-level (Table 44), we see mediocre `precision` and `accuracy`, along with very low `recall` and `F1-score`. We don't have any pattern of seasonality here, since this model is blind to it. When we take a look at model performance on the metric level, we see pretty dismal `recall` and `F1-score` across the board; we have a relatively large amount of false negatives—a byproduct of clamping down on model sensitivity. For the most part, `precision` and `accuracy` are good at the metric level, but it is evident that we can improve on this performance.

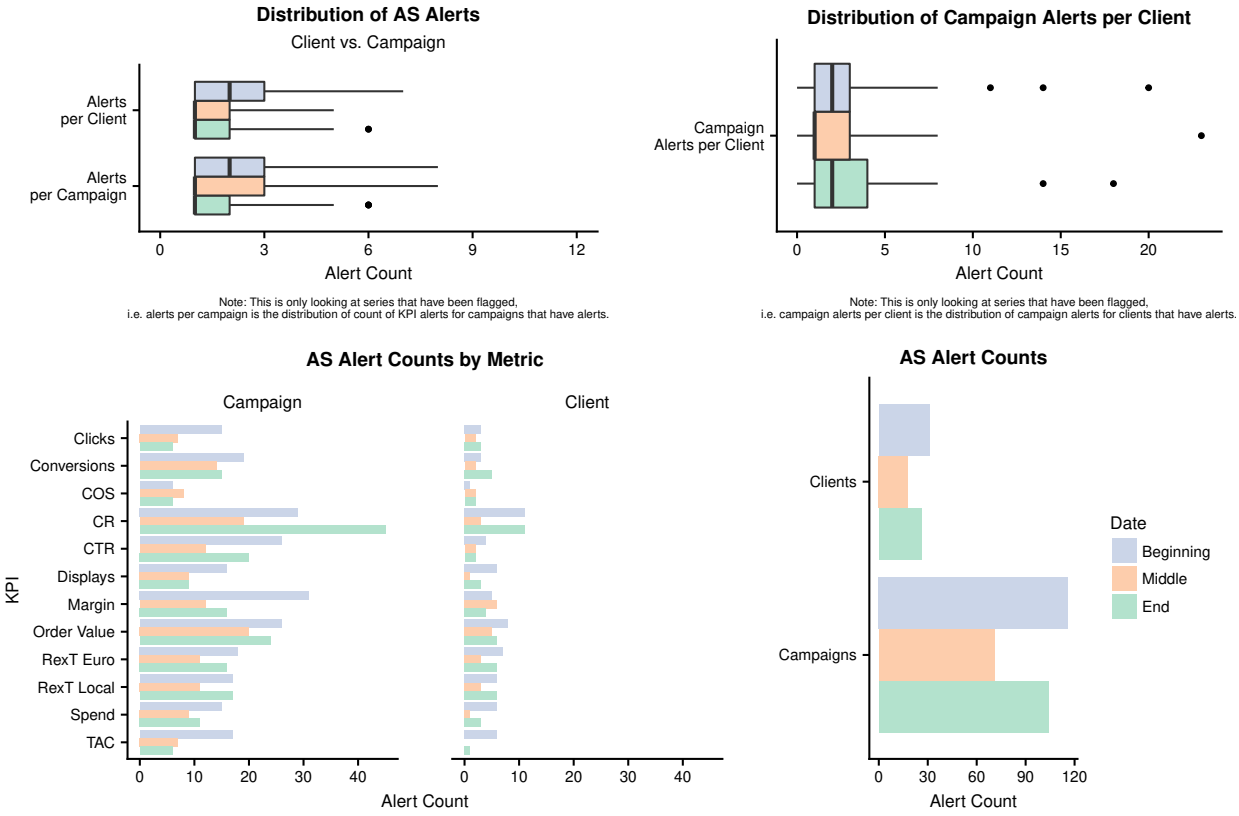


Figure 35: AS alert counts using Chi-Square Tukey Fence More Extreme Method

Table 44: Prediction vs. Metis for AS alerts using Chi-Square Tukey Fence More Extreme Method

Date	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Beginning	33	10	1,443	416	0.77	0.07	0.13	0.99	0.78
Middle	16	5	1,545	361	0.76	0.04	0.08	1.00	0.81
End	23	2	1,494	372	0.92	0.06	0.11	1.00	0.80
TOTAL	72	17	4,482	1,149	0.81	0.06	0.11	1.00	0.80

Metric	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Clicks	1	0	198	103	1.00	0.01	0.02	1.00	0.66
Conversions	2	1	213	75	0.67	0.03	0.05	1.00	0.74
COS	0	1	237	95	0.00	0.00	NaN	1.00	0.71
CR	7	0	196	84	1.00	0.08	0.14	1.00	0.71
CTR	6	1	232	62	0.86	0.09	0.16	1.00	0.79
Displays	3	2	214	109	0.60	0.03	0.05	0.99	0.66
Margin	16	5	885	72	0.76	0.18	0.29	0.99	0.92
Order Value	4	1	219	96	0.80	0.04	0.08	1.00	0.70
RexT Euro	16	2	844	135	0.89	0.11	0.19	1.00	0.86
RexT Local	3	2	201	82	0.60	0.04	0.07	0.99	0.71
Spend	5	0	224	112	1.00	0.04	0.08	1.00	0.67
TAC	9	2	819	124	0.82	0.07	0.12	1.00	0.87

TS Results

We trigger more alerts beginning and end of month as expected with seasonal changes in site events both at the monthly and the weekly level. Most clients with alerts, have only 1 tag with an alert; although, the maximum alerts experienced can be quite high when there are issues on multiple tags. Once again, we see more alerts during the beginning and end of the month compared to the middle. Seasonality strongly influences site events and our model's naive assumption to not use seasonality fails to account for this. Using the boxplots in Figure 36, we can observe that we have more variation on alert counts at the site type than at the event name. Event name tags don't have much varation; however, when we have a tablet tag alert, 50% of the time in the beginning of the month, we have between 1 and 4 alerts—quite different from when we looked at percent change methods. There is also much higher variation of the android tag alerts in the beginning of the month compared to the other time periods.

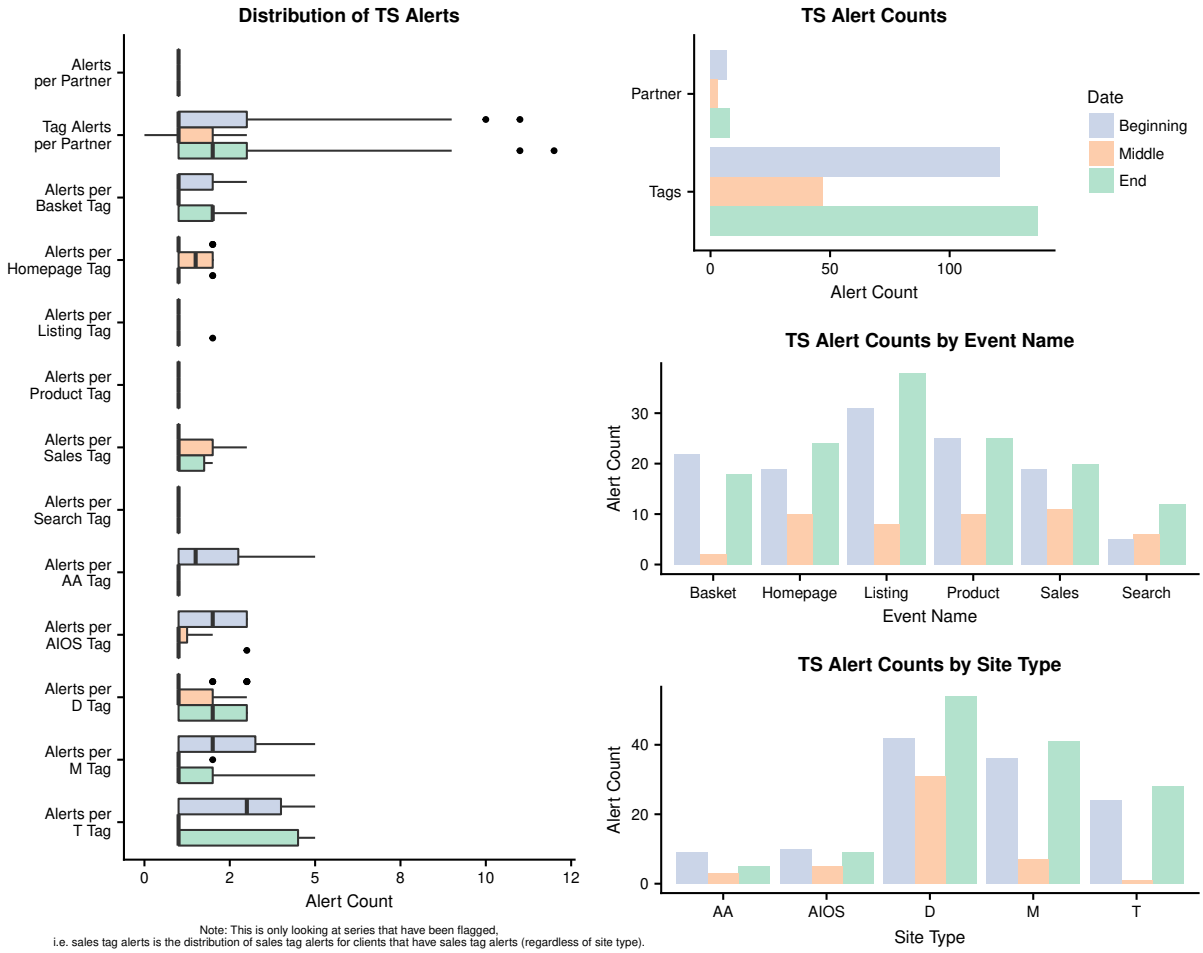


Figure 36: TS alert counts using Chi-Square Tukey Fence More Extreme Method

However, somewhat surprisingly, our false negatives are spread pretty evenly across the month (see Table 46). This suggests that the users want to see these alerts even though they are most likely due to seasonality. Performance across event name seems pretty similar. The users don't know which tag they are looking at, so we should expect this. Same as with event name, we don't see differences across site type aside from volume.

Table 46: Prediction vs. Metis for TS alerts using Chi-Square Tukey Fence More Extreme Method

Date	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Beginning	25	10	1,137	79	0.71	0.24	0.36	0.99	0.93
Middle	9	3	1,081	70	0.75	0.11	0.20	1.00	0.94
End	27	2	1,049	92	0.93	0.23	0.36	1.00	0.92
TOTAL	61	15	3,267	241	0.80	0.20	0.32	1.00	0.93

Metric	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Basket	6	2	500	43	0.75	0.12	0.21	1.00	0.92
Homepage	8	3	575	31	0.73	0.21	0.32	0.99	0.94
Listing	15	5	451	33	0.75	0.31	0.44	0.99	0.92
Product	13	3	635	45	0.81	0.22	0.35	1.00	0.93
Sales	7	1	582	32	0.88	0.18	0.30	1.00	0.95
Search	7	0	258	32	1.00	0.18	0.30	1.00	0.89
Site Level	5	1	266	25	0.83	0.17	0.28	1.00	0.91

Metric	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
AA	3	1	187	19	0.75	0.14	0.23	0.99	0.90
AIOS	4	2	194	9	0.67	0.31	0.42	0.99	0.95
D	26	7	1,231	54	0.79	0.32	0.46	0.99	0.95
M	14	2	952	95	0.88	0.13	0.22	1.00	0.91
Site Level	5	1	266	25	0.83	0.17	0.28	1.00	0.91
T	9	2	437	39	0.82	0.19	0.31	1.00	0.92

Territory RexT Results

Since the territory RexT levels are aggregations of clients, we can expect them to be more stable time-series. However, those monitoring this are likely to want "alerts" on variations that might not seem alert-worthy were they client-specific because the volume is greater here and there might be a multi-client or market-level issue starting. This makes the model's job much harder. It has to be more sensitive at this level, and this method's one-size fits all approach doesn't do us any favors. We don't wrongly classify any positives because these are large changes that users agreed with, unfortunately, we miss quite a few alerts (false negatives). We have essentially maximized our `precision` at the expense of `recall` and thus have a pretty dismal `F1-score`. It's obvious we need a model that adjusts its sensitivity to the metric and not just a significance level. It is also likely that our Tukey fence is too wide for this high-level of an aggregation.

We don't have enough alerts triggered over the dates selected to really look for patterns in the data. We see that the only alerts flagged are the lowest-level (country) proving that the more aggregated we get, the less our model flags. This is even more evident when we look at which country was triggered: Chile; this is one of the smallest territories that we looked at. Looking at the boxplot in Figure 37, we see that most territories don't have alerts at any time (North America regions), and that when we do have variation, 75% of the alert counts are within 1 alert of each other with LATAM having the higher variation.

When we compare the model's predictions to what users classified (results shown in Table 49), we see that while our `accuracy` and `precision` are good, we have awful `recall` and `F1-score`. Most of our false negatives were in the middle of the month. Since we don't take seasonality into account in this model, this could just be due to users deciding spikes/drops on the beginning/end of the month aren't alerts because of when it happened; as such, both the model and user would classify it as a negative. Again, we don't have any pattern at the territory level; our false negatives are spread across the data.

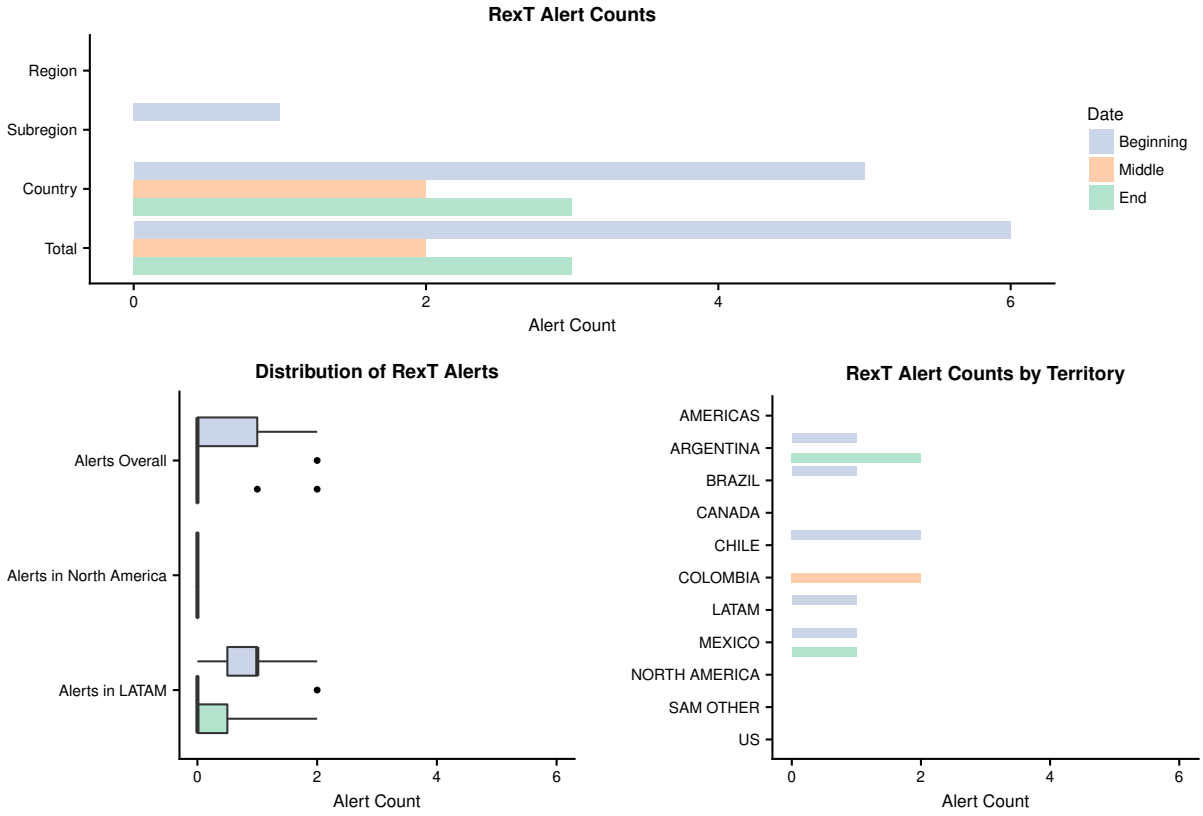


Figure 37: RexT alert counts using Chi-Square Tukey Fence More Extreme Method

Table 49: Prediction vs. Metis for territory RexT alerts using Chi-Square Tukey Fence More Extreme Method

Date	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Beginning	6	0	56	4	1	0.60	0.75	1	0.94
Middle	2	0	54	10	1	0.17	0.29	1	0.85
End	3	0	63	0	1	1.00	1.00	1	1.00
TOTAL	11	0	173	14	1	0.44	0.61	1	0.93

Territory	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
AMERICAS	0	0	17	1	NA	0.00	NA	1	0.94
ARGENTINA	3	0	14	1	1	0.75	0.86	1	0.94
BRAZIL	1	0	17	0	1	1.00	1.00	1	1.00
CANADA	0	0	16	2	NA	0.00	NA	1	0.89
CHILE	2	0	12	4	1	0.33	0.50	1	0.78
COLOMBIA	2	0	15	1	1	0.67	0.80	1	0.94
LATAM	1	0	16	1	1	0.50	0.67	1	0.94
MEXICO	2	0	15	1	1	0.67	0.80	1	0.94
NORTH AMERICA	0	0	16	2	NA	0.00	NA	1	0.89
SAM OTHER	0	0	18	0	NA	NA	NA	1	1.00
US	0	0	17	1	NA	0.00	NA	1	0.94

ROC Curve

Since this model yields a `TRUE/FALSE` answer to whether a given point is an anomaly, we don't have a threshold we can alter and will provide the ROC "points" instead (Figure 38). Once again, it is easy to see that we accomplished our goal of becoming less sensitive. All of the points are very close to the y-axis meaning they had near-zero false positive rates. However, we can see that we aren't doing a great job of catching all the positives; our true positive rate is sub-50%. We can also observe that, in general, we are performing better than deciding randomly (i.e. flipping a coin for each series)—the dashed line. We have succeeded in lowering our false positive rate while increasing our true positive rate compared to the percent change models' ROC curves at similiar points, and all our points are further to the left (lower false positve rate) than the right-most points of the Twitter models (meaning this is less sensitive), however, we still need a much higher true positive rate to be optimal.

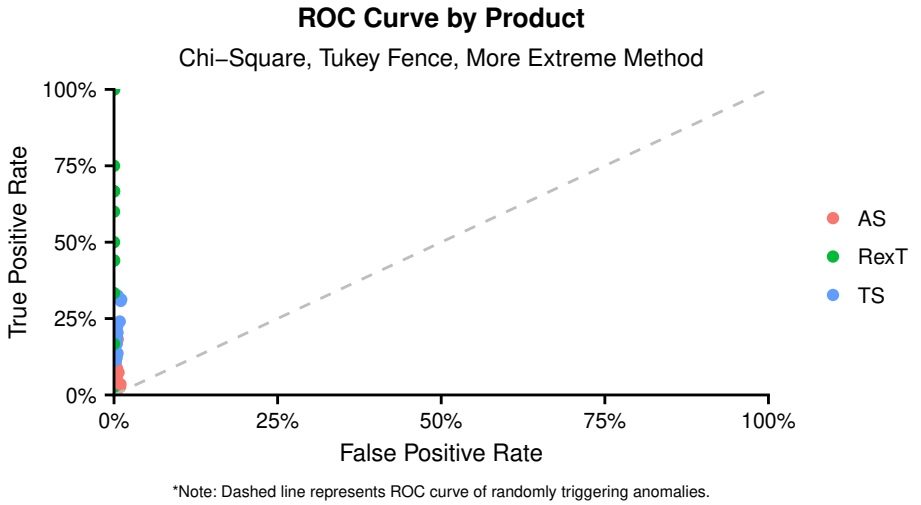


Figure 38: ROC curve for Chi-square Tukey Fence More Extreme method

Alogrithm Complexity

This process can take quite long to calculate each alert and requires the use of the R package `data.table` to run in production in a reasonable amount of time. This will only get worse the more metrics we add. Average complexity (Θ):

$$\Theta(n) = (k * n)^2 \tag{14}$$

where:

- n = *number of series to inspect (i.e. total number of clients + total number of campaigns)*
- k = *number of KPIs to inspect (for each series)*

This gives us an overall quadratic complexity ($\Theta(n^2)$) since k should always be way less than n as well as a tight best-case of $\Omega(n^2)$ and tight worst-case of $O(n^2)$. Note that this is method much slower than the mean/median methods. For large n, we would be much better off with a faster algorithm if one exists.

Conclusions

In switching from the Twitter model, we improved our `precision`, but lowered our `F1-score`'s; we were valuing the reduction in false positives more than that of false negatives, since we only had access to the number of false positives. As expected, the model had very high `specificity` for all tasks due to the high

probability that a given series won't be an anomaly. This analysis was performed using a multiplier of 3 for the Tukey fence; however, if we lower it to 1.5, we practically double our alerts and improve our `F1-score` at the expense of `precision`. Since it's clear we are still missing something with this approach, we use 3 for the multiplier in production to just flag the most obvious outliers. This model also has very high `precision` across the board, and we are successfully alerting for large-deviation alerts. However, false negatives abound; in reducing sensitivity and increasing alert relevance, we inadvertently reduced the proportion of alerts we correctly identify—`recall` (Equation 2). Hence, the lower-deviation alerts are now going undetected. This can be seen in our ROC curve points: while our false positive rate is minimal, our true positive rate isn't too high, meaning we have room for improvement. Furthermore, we are missing a seasonality component, so we unnecessarily flag series whose cause can be traced to seasonality. Even though, some users still want to know this, ideally, we should take this into account in our model. In conclusion, since we maximized `precision` when selecting a less sensitive model, we obtained a very low `recall` and `F1-score` across the board—we are missing alerts, and we need to reincoporate seasonality somehow.

Western Electric Rules

Statistical Process Control is a practice often used in manufacturing as quality control "alerting" for products. Figure 39 shows the normal distribution which many natural processes follow; we can observe where the mean and standard deviations from the mean fall, as well as the percentage of observations that fall in those areas. This motivates the use of these ranges to flag "abnormal" behavior. Control charts (see Figure 40) are constructed, and the outputs are graphed along with the mean for the sample and some boundaries: most commonly the upper control limit (UCL) and the lower control limit (LCL). Rules, such as distance in standard deviations from the mean and patterns, are then used to determine if the process has gone out-of-control.[12]

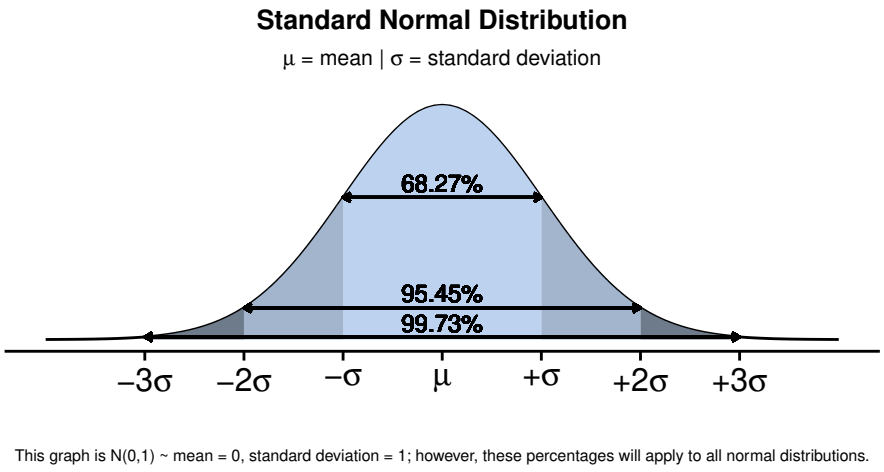


Figure 39: Normal curve showing standard deviations from the mean

In 1956, Western Electric published some of the rules it used for quality control which take into account patterns and use the mean and standard deviation to catch out-of-control or non-random occurrences.[13] I will use their rules along with a few additional from the *Nelson rules*[14] which build upon these, giving us 8 rules.

These rules will be ordered from most serious to least serious for our purposes, and the algorithm will flag the most flagrant and move onto the next series (it won't flag the same series for multiple violations). From there, I will look into the performance of all the rules and different combinations of the rules to determine

[12] Statistical process control
[13] Western Electric rules
[14] Nelson rules

which rules work best and compare the "best" version to the other algorithms we looked at. Below you will find an explanation of each of the rules in the order used for the analysis along with some control charts illustrating what each rule will flag (Figure 40).

- **Rule 1** checks if the last point is outside the control limits, which are defined as $\mu \pm 3\sigma$ where μ (mu) is the mean and σ (sigma) is the standard deviation. This means that the last point was well beyond what we would expect to see for a given distribution. For instance, if we assume a normal distribution for our data, we expect 99.7% of our data points to be within 3 standard deviations of the mean, so we definitely want to know about that 0.15% that fall outside on a given side of the mean ($\frac{0.3}{2}$).
- **Rule 2** checks if 2 out of 3 last points fall beyond ± 2 standard deviations on the same side of the centerline (the mean). This is a less extreme version of rule 1; here, we require more points, but they don't have to be as far from the mean. Violation of this rule means the variation in the series is increasing. Again if we are using a normal distribution, we are looking for points that fall outside of where 95% of them lie. Note that we are only looking for points on one side of the mean; this means we are looking for the 2.5% of points that fall outside our desired range (to a given side of the mean).
- **Rule 3** checks if 4 out of 5 last points fall beyond ± 1 standard deviation on the same side of the centerline. Violation of this rule as with rule 2 indicates that the variation of the series is increasing, but to a lesser degree and possibly not as rapidly as with rule 2 (should performance continue this way). When looking at the normal, we expect 68% of the points to not violate this rule; therefore, we are requiring even more points than with rule 2 to fulfill this requirement. Since we are only looking for points to one side of the mean, so we are looking at 16% of points that would be in this area.
- The **trend rule** checks if 6 consecutive points going up or down. This happens when we are ramping up or perfomance is slowly declining over time; the latter is definitely something we would want to know about. It is important to note that this rule is not taking account the magnitude of the increase or decrease point to point. This means that we might find ourselves in a situation where the value drops (or rises) day after day by an insignificant amount (like 1 click if we are talking hundreds or thousands of clicks a day); the trend rule would trigger an alert in that case which would obviously be a false positive. However, one would assume that changes have larger magnitudes and the confluence of circumstances to have several days of declining/inclining performance along with miniscule differences in values day-over-day to be rare, so hopefully this won't be an issue for our purposes. Of course we could put a limit on the magnitude of the change, but that introduces a parameter that will vary by time series and be subjective.
- The **mixture rule** checks if 8 consecutive points outside of ± 1 standard deviation on either side of the mean. This is a sign that things are starting to get erratic and stray from the mean. The mixture rule is a variation on rule 3. Here, we require more points, and they have to be consecutive, because now we are more concerned with the spread to *either side* of the mean. This is where we expect 32% of the points in the normal distribution to be.
- The **stratification rule** checks if 15 consecutive datapoints within ± 1 standard deviation on either side of the mean. This may not seem like an alert in our context, since the mean is becoming a more accurate descriptor of where we could expect future performance to lie, but this means that the underlying distribution is changing—something we might want to know about. Using the normal, we expect 68% of the points to fall in the "stratification" range; therefore, seeing 15 consecutive points in this region, may indicate that our assumptions about the data aren't true anymore. This is probably more of an issue when you have to estimate the values and stick with them for a while; for `SCOPE`, these values will be calculated from the sample every time we run which is why this rule is ranked near the bottom.
- **Rule 4** checks if 9 consecutive points fall on the same side of the mean. This would mean that the average is no longer a good descriptor of future performance, since the last several points were all above or below it and not spread. In this case, we aren't necessarily concerned with the distance from the mean in standard deviations, but the most recent points all being above or below the mean when we would expect them to straddle the mean. Again this is probably more of an issue when you can't calculate the summary statistics (mean and standard deviation) each time, so we rank it low in

importance.

- The **noise rule** checks for constant oscillation in a series—if 14 points in a row alternate in direction, increasing then decreasing or vice versa. As the name suggests, series that get flagged for this alert, may not have anything meaningful for us to interpret—it may just be noise. For our purposes, we have many erratic series or even normal ones that oscillate, because they won't always be the same value every day. We could easily trigger this too many times, so we rank this last.
- The "No Violations" chart won't flag any of the rules above, but it doesn't necessarily mean that we don't have any alert-worthy data.

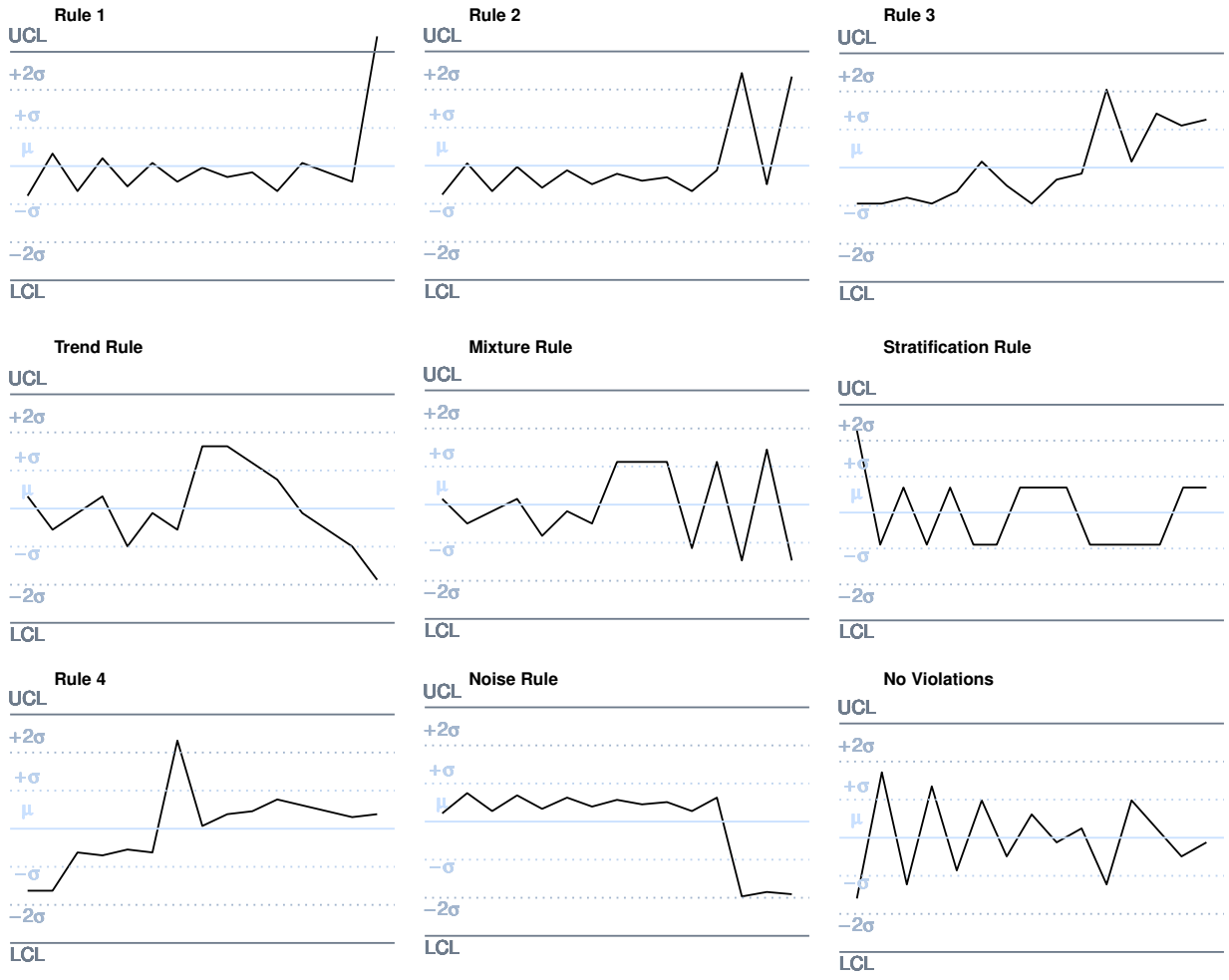


Figure 40: Western Electric Rule Violation Examples

Note that, in Figure 40, each graph has different data meaning UCL, LCL, etc. won't be the same across graphs, so they won't line up in the grid.

Results

Before diving into how this method performed on the data, I want to look holistically at the distributions of the rules violated to see which rules are most sensitive and gather intuition of their strengths and weaknesses in the context of our data. I will determine if we can exclude any rules on a per-product (AS, TS, RexT) basis.

Remember that we have ranked our rules in order of what constitutes a serious issue to less serious ones. That order is: rule 1, rule 2, rule 3, trend, mixture, stratification, rule 4, and noise. The algorithm will *only* flag the most serious violation it finds; for example, if rule 1 and the trend rule would both be triggered, *only* rule 1 will be reported. This means that for each time series we know what was the most serious violation, but not all the violations. Likewise, for something to be flagged as violating rule 2, it can't violate rule 1. One would expect that the most serious violations (the ones the algorithm checks for first) should have fewer violations overall than the looser rules.

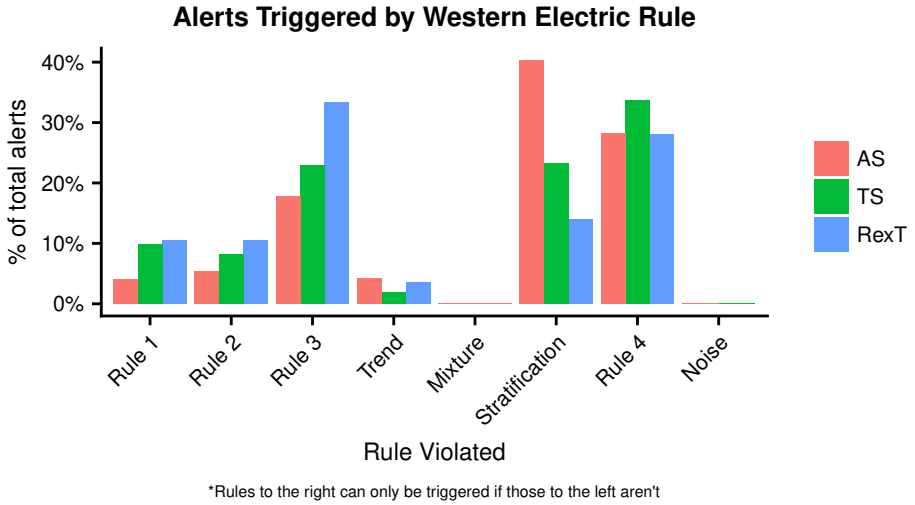


Figure 41: Alerts flagged at each rule in the Western Electric rules cascade

Looking at the percent of total alerts that was triggered at each stage of the algorithm in Figure 41, we see that rule 1 and rule 2 are roughly equivalent for AS and RexT while we actually have more rule 1 alerts than rule 2 for TS. From this graph, we learn that much fewer AS metric time series experience values outside of 3 standard deviations from the mean compared to TS and RexT which are roughly equal. Going to rule 3, we see large jumps in the percentage of total alerts; a third of the alerts for RexT violate rule 3. Again, we expect rule 3 to have more violations than rule 1 and rule 2 because it is less strict. The trend rule triggers much smaller share of the alerts, however, the violations for the TS time series is even smaller than the AS and RexT. This is most likely due to TS time series being relatively stable; we wouldn't expect 6 points in a row all increasing or all decreasing—normal for AS or RexT series due to ramping up, things being turned off or territory level seasonality (i.e. holiday seasons).

We essentially have no alerts using the mixture rule; this could mean one of two things: either all the erraticism also violates one of the 4 previously checked rules, or we don't have series with so many points outside of one standard deviation. With the stratification rule, we see the largest population of alerts for the AS and large volumes for TS and RexT; this means that we actually see lots of series (especially for AS metrics) where the last quarter of the dates (15/60 days) we take into account when checking for anomalies are within one standard deviation of the mean. This makes sense because we would expect our data to go through phases, i.e. increase/decrease for a few days, find new level, stay for a while, increase/decrease, etc. We also see large amounts of alerts triggered for rule 4; the TS alerts have the highest percentage here—possibly due to monthly or yearly seasonality where the series could acheive a higher- or lower-level than the mean dictates, and due to the overall stability of the series, stay there. Somewhat surprisingly, we hardly have any violations of the noise rule; this is most likely due to the fact that while we do have noise in our series, they do not perfectly alternate increases and decreases for 2 weeks straight.

It is also interesting to note that for RexT data, rules 3 and 4 trigger over half of all the alerts. For AS series, stratification and rule 4 account for nearly 70% alerts; TS series have almost 80% of the alerts across rule 3, stratification, and rule 4. This begs the question: given that these alerts are the less serious ones, should we

really have such a large percentage of our alerts caught here? (meaning most of the flagged alerts are of the less serious—and likely false positive—kind). Furthermore, for AS (campaign + client) and TS (site- and tag-level) series we are triggering over 5 thousand series as alerts each! It's pretty clear, right off the bat, that we can't use all these rules for our purposes.

In order to provide an initial insight on which rules to stop using, we can try to find a point in where, after evaluating a certain number of rules to reach that point, we have reduced our alerts to a reasonable percentage of the original number.

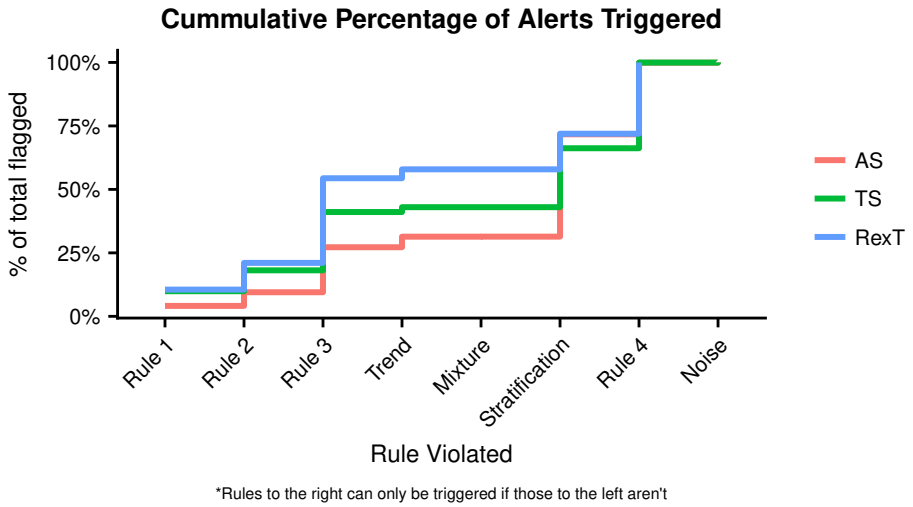


Figure 42: Cummulative percentage of alerts triggered at each stage of the Western Electric rules cascade

By looking at the cummulative percentage of the alerts triggered (Figure 42), we can see as we check each rule, how much of the alerts have already been triggered. For example, we can observe that after triggering the alerts for rule 2, we have triggered about 10% of the AS alerts (triggered at rule 1 + triggered from rule 2), 18% of the TS alerts, and 21% of the RexT alerts. We can also see the huge jump across the board from the stratification rule here (along with rule 3 and 4) as well.

Note that any analysis from here on out as to the effectiveness of each rule has a bias. I have ranked the rules in importance for our purposes initially, and we are only flagging less important rule violations if the most important ones haven't been flagged beforehand. This means that when we evaluate say the stratification rule, we aren't evaluating the performance of the rule on it's own, but it's relevance in that spot in the hierarchy. It's crucial to keep this in mind, however, we can take into account the volume of alerts that are coming from that alert, along with the fact that the volume is only a subset of what it could have been without the hierachy. If a rule doesn't perform well in this context we will likely want to remove it, but what we are actually looking to do is determine if there are rules that absolutely need to be removed (i.e. whatever they do flag is most irrelevant) or the rule after which all additional rules don't really contribute anything to the task at hand. It's possible that these cutoff points are different for AS, TS, and RexT as we saw their distributions of alerts across the rules was different at each step (although following a high-level pattern of increasing sharply for certain rules).

Since we have essentially no alerts for mixture and noise, we can start by removing those rules from the algorithm. Noise was the last rule to check, so we can assume a series that would have flagged there (if any) was flagged by one of the more important rule; and therefore, the noise rule is either redundant or irrelevant for the data. Mixture was in the middle of the rule list, however, we can assume that almost all of the alerts that would have been flagged by this rule as a stand-alone test have been caught in the more important (and stricter) rules.

Preliminary Analysis of Rules

Given that this method as a whole is triggering way too many alerts using all the rules, we clearly need to find the cutoff point. In order to look at where the cutoff point should be, I will check the performance using multiple rules. Note that we can't do rule 1 and rule 3 without thinking about the series that could have been flagged by both rule 2 and 3—this will skew the numbers; instead, I will look at which rule should be the last rule we check and ignore alerts triggered on rules following that one.

As such, I will first look to eliminate the second half of the checks (mixture, stratification, rule 4, and noise) of which we only have to look further into stratification and rule 4, since mixture and noise have been ruled irrelevant for our data. The stratification rule—being the most sensitive—is the best rule to evaluate first. As expected, the performance of the stratification rule is atrocious for the AS data (see Table 51); keep in mind that this alert is triggered when performance is pretty similar day-after-day for a 2 week period which is actually a good sign for us. The performance is even more horrid for the TS data with a lower percentage of the "alerts" being true positives. For the RexT data points, we have comparatively few true positives, but we do have false negatives. It's clear this rule isn't helping us with this data.

Table 51: Performance stopping after stratification rule

Series	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
AS	369	1,298	3,201	852	0.22	0.30	0.26	0.71	0.62
TS	142	623	2,659	160	0.19	0.47	0.27	0.81	0.78
RexT	12	29	144	13	0.29	0.48	0.36	0.83	0.79

Table 52 shows the performance of stopping after running rule 4. Even though this rule is run near the end, we trigger many alerts and most of them are false positives. It's safe to say we can cut this from our algorithm as well. Remember, rule 4 checked if several consecutive points fall on the same side of the mean. This is important if we care about the overall pattern of our data changing slightly, but not really necessary for our purposes.

Table 52: Performance of stopping after rule 4

Series	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
AS	464	1,780	2,719	757	0.21	0.38	0.27	0.60	0.56
TS	179	990	2,292	123	0.15	0.59	0.24	0.70	0.69
RexT	14	43	130	11	0.25	0.56	0.34	0.75	0.73

Now, for the first half of the rules. It was pretty obvious we needed to trim the last two rules from our algorithm based on the amount of alerts they triggered so late in the process—there couldn't possibly that many issues! However, it's not as easy to determine what to trim from the first half of the rules: we once again are faced with the issue of trying to capture more alerts or focusing on providing the most relevant alerts. We probably don't want more than half of our alerts to be false positives, but how much should we tolerate? Here, I'm going to see how many rules we can use to maximize our `F1-score` without sending our false positives through the roof.

Rule 1 will be used regardless and will serve as the baseline to determine if adding another rule helps or hurts this method's performance. As with most of the methods we looked at thus far, rule 1 performs best on the RexT data although `precision` is way better than `recall` (see Table 53). Notice that the false negatives for the AS data are outstandingly high! AS alerts clearly aren't always as obvious as being far from the mean. The TS data has the worst `precision` of all, but not the worst `F1-score` because it doesn't have nearly as many false negatives as the AS data.

Table 53: Performance of stopping after rule 1

Series	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
AS	98	19	4,480	1,123	0.84	0.08	0.15	1.00	0.80
TS	71	40	3,242	231	0.64	0.24	0.34	0.99	0.92
RexT	6	0	173	19	1.00	0.24	0.39	1.00	0.90

If we add in rule 2, our `precision` on RexT data takes a huge hit, as seen in Table 54! We correctly identify another alert, however, we completely destroy our `precision` and end with a slightly lower `F1-score`. We do see slight improvement in `F1-score` for the TS data; however, now our false positives outnumber our true positives for TS data. This is a dangerous situation to be in—if more than half of the alerts we send out are not actually alerts, people won't pay attention to them anymore and will miss the true alerts. AS data also sees a small increase in `F1-score`, but we also significantly increase our false positives. The ratio of false positives to true positives isn't quite as bad as the TS one, however, it may be difficult to stand behind using this model in production.

Table 54: Performance stopping after rule 2

Series	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
AS	149	106	4,393	1,072	0.58	0.12	0.20	0.98	0.79
TS	94	130	3,152	208	0.42	0.31	0.36	0.96	0.91
RexT	7	5	168	18	0.58	0.28	0.38	0.97	0.88

By adding in rule 3 (Table 55), we now have all data sets with more false positives than true positives (more than double for TS data). Although we get some improvement in our `F1-score` for AS data, and our TS and RexT `F1-score`'s stay roughly the same, this is clearly unacceptable. Therefore, for the remainder of the analysis on this method, I will use only rule 1.

Table 55: Performance stopping after rule 3

Series	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
AS	226	377	4,122	995	0.37	0.19	0.25	0.92	0.76
TS	128	356	2,926	174	0.26	0.42	0.33	0.89	0.85
RexT	11	20	153	14	0.35	0.44	0.39	0.88	0.83

Should we go another step forward and add the trend rule (see Table 56), we observe marginal improvements over stopping at rule 3, however, they aren't enough to beat stopping at rule 1.

Table 56: Performance stopping after trend rule

Series	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
AS	251	437	4,062	970	0.36	0.21	0.26	0.90	0.75
TS	136	370	2,912	166	0.27	0.45	0.34	0.89	0.85
RexT	12	21	152	13	0.36	0.48	0.41	0.88	0.83

Note how each time we added a rule (besides rule 1), we added more false positives than true positives. This means that we can't really use a method like this for our data. What determines an alert, unfortunately, can't be captured be a rule-based system as we have seen with all the models thus far. There are many more intricasies than these systems can account for.

AS

We see significantly less alerts during the middle of the month compared to the beginning and end of the month. This is in line with our intuition that performance is very different for these times; we can experienece spikes and dips that are actually expected. Since this method catches large variations from the average, we should expect more alerts during the beginning and the end of the month. The same pattern can be observed on a by metric basis with RexT, CR, and order value having larger disparities. Note that we have also seen this pattern before with other methods. Using the boxplots in Figure 43, we see significantly less variation during the end of the month for alerts per client and overall 50% of clients have between 1 and 3 alerts. There is actually less variation in alerts per campaign with this method; 50% of the time we have between 1 and 2 for the middle and end of the month. Campaign alerts per client are between 1 and 4 50% of the time with all clients having less than 30 campaign alerts.

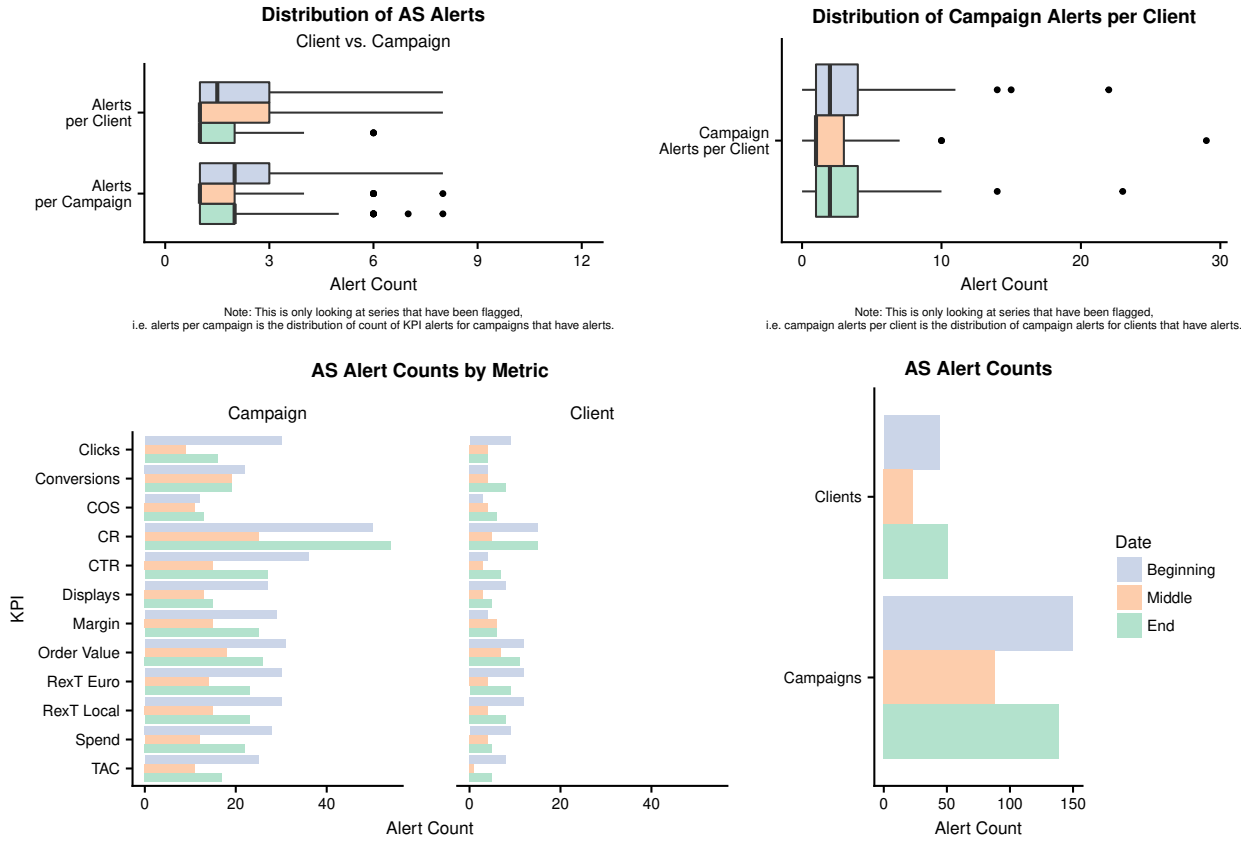


Figure 43: AS alert counts using Western Electric Rule 1

In Table 57, the data collected from `Metis` is compared to what was flagged by Western Electric Rule 1. For middle of the month, we are hardly triggering any false positives, however, we are missing plenty of actual alerts. This gives us the worst overall performance. Beginning and end of the month have better performance despite, having more alerts (and false positives). The performance as a whole though isn't great. We are performing much better for RexT Euro and margin than our overall score, although not great. The worst performances are on clicks and COS. There are quite a few metrics with perfect `precision`, but this many times is because we are missing alerts (false negatives).

Table 57: Prediction vs. Metis for AS alerts using Western Electric Rule 1

Date	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Beginning	47	9	1,444	402	0.84	0.10	0.19	0.99	0.78
Middle	19	4	1,546	358	0.83	0.05	0.10	1.00	0.81
End	32	6	1,490	363	0.84	0.08	0.15	1.00	0.80
TOTAL	98	19	4,480	1,123	0.84	0.08	0.15	1.00	0.80

Metric	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Clicks	2	0	198	102	1.00	0.02	0.04	1.00	0.66
Conversions	4	1	213	73	0.80	0.05	0.10	1.00	0.75
COS	2	1	237	93	0.67	0.02	0.04	1.00	0.72
CR	10	0	196	81	1.00	0.11	0.20	1.00	0.72
CTR	7	1	232	61	0.88	0.10	0.18	1.00	0.79
Displays	7	2	214	105	0.78	0.06	0.12	0.99	0.67
Margin	14	6	884	74	0.70	0.16	0.26	0.99	0.92
Order Value	6	0	220	94	1.00	0.06	0.11	1.00	0.71
RexT Euro	22	4	842	129	0.85	0.15	0.25	1.00	0.87
RexT Local	5	1	202	80	0.83	0.06	0.11	1.00	0.72
Spend	5	0	224	112	1.00	0.04	0.08	1.00	0.67
TAC	14	3	818	119	0.82	0.11	0.19	1.00	0.87

TS

We observe the same trend with the TS data as with the AS data: middle of the month triggers comparatively fewer alerts. This disparity is visible both with the site level alerts and the tag level alerts. Other than this pattern also being present in the overall numbers, there isn't anything of note when looking at the event name and site type breakouts. With this method, we have hardly any variation in alert counts for event names with alerts, but variation of 1-4 in the most extreme variations of site type alerts (Figure 44). For example, the tablet tag during the end of the month has 1-5 alerts for 50% of the clients.

Quite suprisingly (and perhaps beyond explanation), we perform much better for beginning of the month compared to the other times (see Table 59). The performance isn't something we can use in production, however, it is one of the best we have seen so far and is better than the performance on the AS data.

This method performs best on site-level, listing, and sales tag and the worst on homepage. There most likely isn't any underlying reason, just the fact that it is a small amount of alerts triggered in total for each event name. The desktop site type performs the best along with AIOS, however, it is a very small sample size. Take this with a grain of salt, though, because the site types with decent sample sizes are pretty mediocre in performance.

Table 59: Prediction vs. Metis for TS alerts using Western Electric Rule 1

Date	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Beginning	32	23	1,124	72	0.58	0.31	0.40	0.98	0.92
Middle	9	4	1,080	70	0.69	0.11	0.20	1.00	0.94
End	30	13	1,038	89	0.70	0.25	0.37	0.99	0.91
TOTAL	71	40	3,242	231	0.64	0.24	0.34	0.99	0.92

Metric	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Basket	10	6	496	39	0.62	0.20	0.31	0.99	0.92
Homepage	9	9	569	30	0.50	0.23	0.32	0.98	0.94
Listing	14	5	451	34	0.74	0.29	0.42	0.99	0.92
Product	13	8	630	45	0.62	0.22	0.33	0.99	0.92
Sales	9	8	575	30	0.53	0.23	0.32	0.99	0.94
Search	8	2	256	31	0.80	0.21	0.33	0.99	0.89
Site Level	8	2	265	22	0.80	0.27	0.40	0.99	0.92

Metric	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
AA	6	9	179	16	0.40	0.27	0.32	0.95	0.88
AIOS	4	3	193	9	0.57	0.31	0.40	0.98	0.94
D	29	15	1,223	51	0.66	0.36	0.47	0.99	0.95
M	14	7	947	95	0.67	0.13	0.22	0.99	0.90
Site Level	8	2	265	22	0.80	0.27	0.40	0.99	0.92
T	10	4	435	38	0.71	0.21	0.32	0.99	0.91

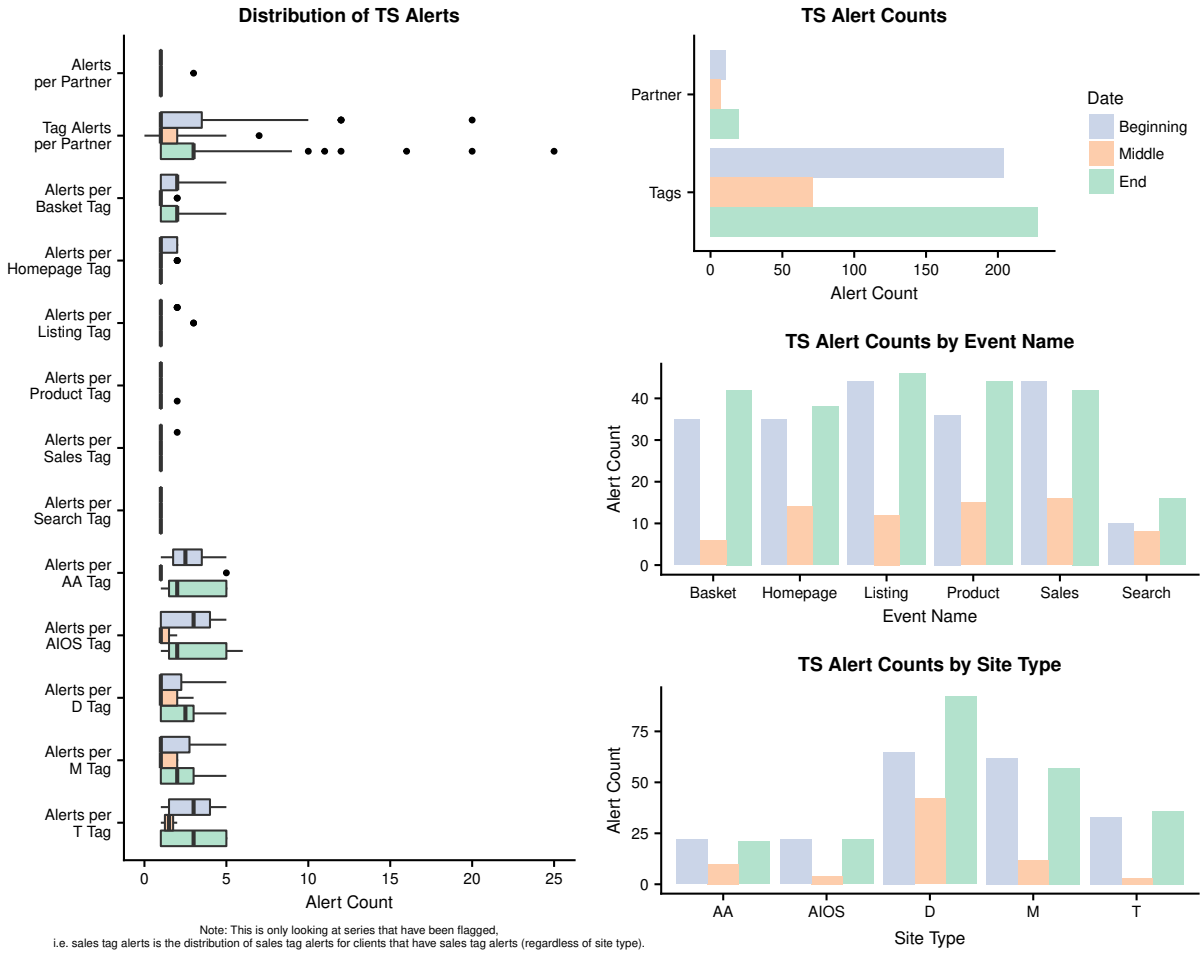


Figure 44: TS alert counts using Western Electric Rule 1

Territory RexT

As we have seen with prior methods presented in this paper for anomaly detection, we have more alerts the more granular we get (country has more than subregion which has more than region). When looking at the territory breakdown, we still see Argentina with the most alerts which is consistent with the prior anomaly detection methods discussed. With the RexT data, we have hardly any variation at all; we only have 2 sections where 50% (or more) of the distribution isn't just at 0 (see Figure 45).

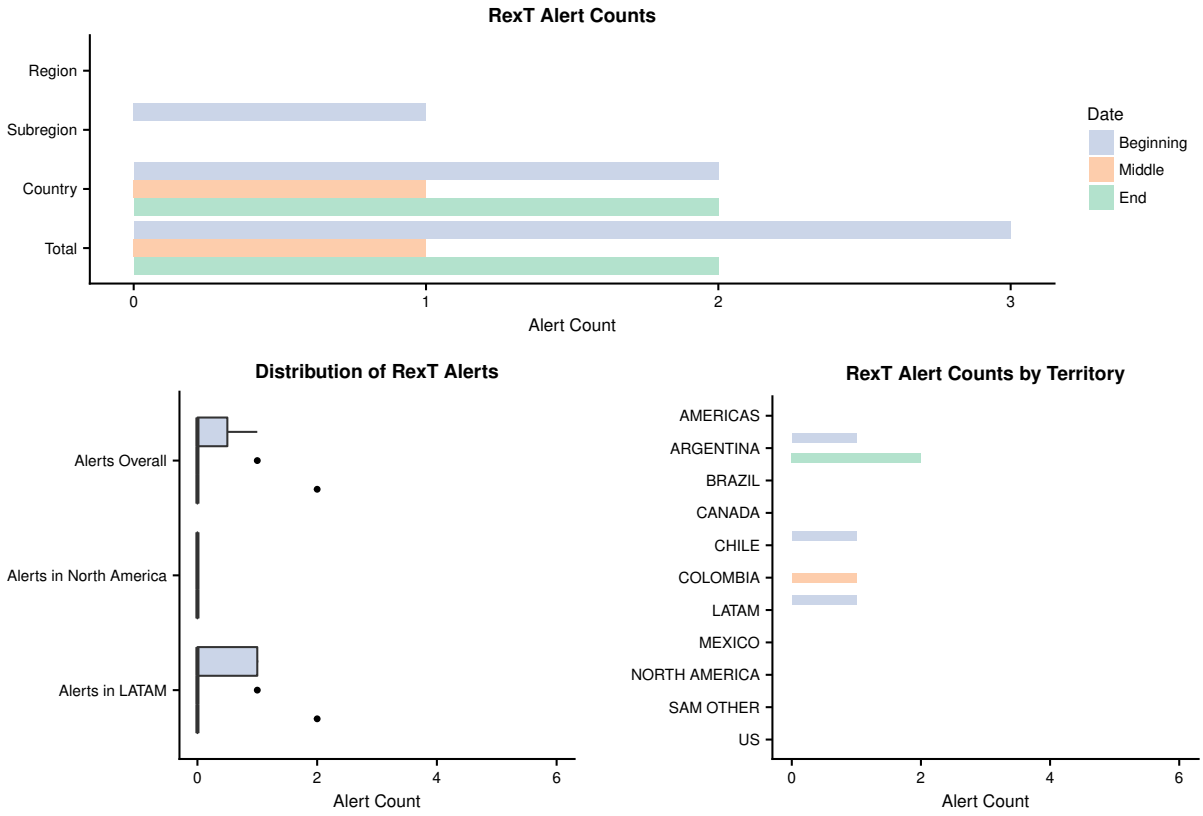


Figure 45: RexT alert counts using Western Electric Rule 1

As usual, we perform the best on RexT data as shown in Table 62; although TS data is a close second. We also observe the same disparity between time of month as with the AS and TS series. Our performance, here, is at its best during the end of month and worst during the middle. We have generally low `precision` and high `recall` for possibly the first time when looking at the territory level; however, we don't really perform well on any territory. This method doesn't work too well for RexT data.

Table 62: Prediction vs. Metis for territory RexT alerts using Western Electric Rule 1

Date	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
Beginning	3	0	56	7	1	0.30	0.46	1	0.89
Middle	1	0	54	11	1	0.08	0.15	1	0.83
End	2	0	63	1	1	0.67	0.80	1	0.98
TOTAL	6	0	173	19	1	0.24	0.39	1	0.90

Territory	TP	FP	TN	FN	Precision	Recall	F1-score	Specificity	Accuracy
AMERICAS	0	0	17	1	NA	0.00	NA	1	0.94
ARGENTINA	3	0	14	1	1	0.75	0.86	1	0.94
BRAZIL	0	0	17	1	NA	0.00	NA	1	0.94
CANADA	0	0	16	2	NA	0.00	NA	1	0.89
CHILE	1	0	12	5	1	0.17	0.29	1	0.72
COLOMBIA	1	0	15	2	1	0.33	0.50	1	0.89
LATAM	1	0	16	1	1	0.50	0.67	1	0.94
MEXICO	0	0	15	3	NA	0.00	NA	1	0.83
NORTH AMERICA	0	0	16	2	NA	0.00	NA	1	0.89
SAM OTHER	0	0	18	0	NA	NA	NA	1	1.00
US	0	0	17	1	NA	0.00	NA	1	0.94

ROC Curve

Since this model yields a `TRUE/FALSE` answer to whether a given point is an anomaly, we don't have a threshold we can alter and will provide the ROC "points" instead (Figure 46). As usual, we see that RexT appears to have the best performance followed by TS and AS, respectively. This is most likely due to the aggregated nature of the RexT time series; we wouldn't expect such a large change to happen to cause one day to be more than three standard deviations from the mean—it is a bit extreme—giving us few false positives. However, this extreme behavior is much more likely in TS, and especially, AS data. As we saw above, this method performs best on the RexT data with consistently higher true positive rates at low false positive rates.

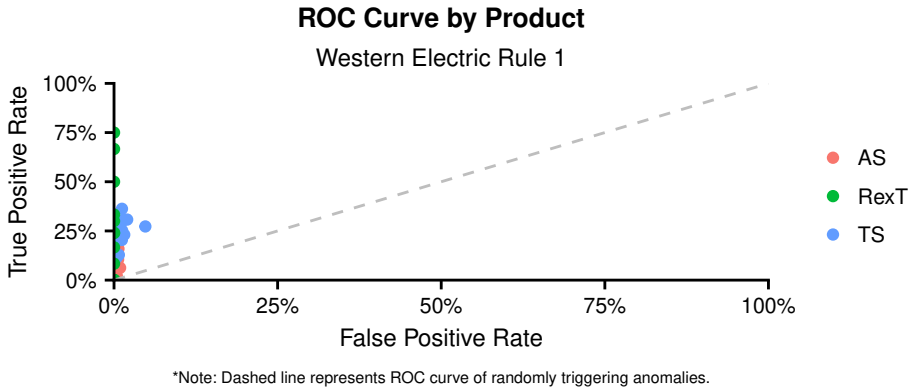


Figure 46: ROC curve using Western Electric rule 1

Alogrithm Complexity

This process can take quite long to calculate each alert and requires the use of the R package `data.table` to run in production in a reasonable amount of time. This will only get worse the more metrics we add. Average complexity (Θ):

$$\Theta(n) = (k * n)^2 \tag{15}$$

where:

- n = *number of series to inspect (i.e. total number of clients + total number of campaigns)*
- k = *number of KPIs to inspect (for each series)*

This gives us an overall quadratic complexity ($\Theta(n^2)$) since k should always be way less than n as well as a tight best-case of $\Omega(n^2)$ and tight worst-case of $O(n^2)$. Note that this is much slower than the mean/median methods. For large n, we would be much better off with a faster algorithm if one exists.

Conclusions

Initially, this method had a lot of promise—rules of varying severity that have been used for years to identify time series that are out-of-control. However, they don't seem to work very well with all our data. Many of the rules were overly sensitive and quite suprisingly values outside three standard deviations from the mean aren't always seen as anomalies. This method worked best on RexT and TS, but each method we have tried so far works better (although definitely not great) for one data type. This gives us a "best" rules-based method for each; our goal is to find a method that works well for all our data. The failure of this method further motivates the finding that a rules-based approach is not truly understanding our data, and we need to use machine learning where we will have more fine-tuned control.

Unsupervised Machine Learning Models

Unsupervised models are useful when we don't have any information on whether our model output is correct (i.e. we don't have labelled data); in this case, we would not know what was an alert and what wasn't—we just have the time series data. These models won't use the labelled data to inform their predictions, however, since we have that data, we can use it to evaluate them for comparision to the supervised learning models presented in the next section. We can expect supervised machine learning models to perform better than unsupervised ones, nonetheless, to be thorough, I went through some potential unsupervised algorithms for comparison. Here, we have to make sure our training data is both representative of the data set as a whole and separate from the test set (unseen by the model during training) used to grade the models on.

I tried the Isolation Forest, Local Outlier Factor, and One-class SVM algorithms on the data. All performed pretty poorly–worse than the current model–so in the interest of brevity (read trying to complete this before my last day), I won't include them here. If you are interested in how these performed, you can take a look at the Jupyter notebook with the name of the method. There is also some dimensionality reduction with PCA (principal component analysis) for those who are interested.

Supervised Machine Learning Models

Since we have the labelled data from `Metis`, we can use supervised machine learning models. With supervised learning, the model can use labelled data to make better judgements; we are no longer guessing as we were with the unsupervised methods discussed previously. These models should perform better than their unsupervised counterparts due to the additional information. As with unsupervised learning, we need our test and train data sets to avoid overfitting of the model which leads to poor ability of the model to generalize to new (unseen) cases.

I tried the following algorithms: Ada Boost Classifier, Decision Tree, Extra Trees Classifer, Gradient Boosted Decision Trees, K Nearest Neighbors, Logistic Regression, Multi-layer Perceptron (Neural Network), Naive Bayes, Passive Aggressive Classifier, Random Forest, Ridge Classifier, Stochastic Gradient Descent Classifier, and Support Vector Machine. After an initial trial with each method, I determined if further investigation was worth it; I will only document here the methods that passed that test and show promise for further investigation. Feel free to check out the Jupyter notebooks for the other methods.

Note that since we will always be separating our data into a training and a testing set, the stats reported here-on-out will only be on the testing set meaning that the number of series tested will be significantly less than in the rules sections.

Logistic Regression

Logistic regression despite its name is a method of classification. It allows us to predict which class an observation belongs to (think: alert or not alert). We can also use it to predict the probability it is an alert by defining a threshold above which it is an alert. For this model, the best performing set of features was: the last 25 data points, dummy-encoded KPI's, exponential weights for TS, linear weights for AS, and including summary statistics (mean, variance, min, max). Everything was min-max scaled ($value = \frac{value - min(x)}{max(x) - min(x)}$).

Note that this method will be performed in Python using Scikit-learn and the Jupyter notebook is available in the repo.

Results

Overall, we achieve a `F1-score` of 0.74 for AS, 0.89 for TS, and 0.96 for RexT; however, when we separate this out by class (alert vs. not-alert), we see that our performance is being carried by the majority class (not alert). Isolating the alert class, our `F1-score` for AS is 0.21 (!), 0.18 for TS (!), and 0.83 for RexT. Figure 47 shows the confusion matrix on the test set using thresholds of 80% for AS, 70% for TS, and 50% for RexT.

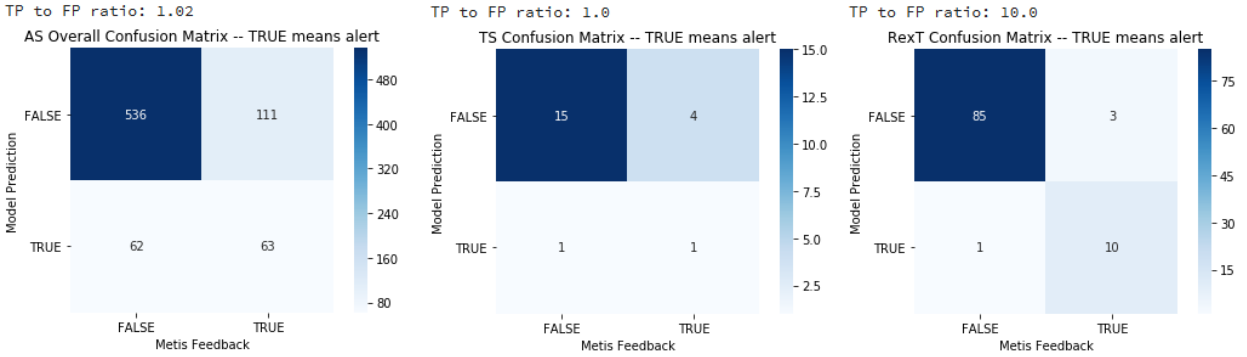


Figure 47: Logistic regression confusion matrix

ROC Curve

Note that although we have a dip in the ROC curve for TS (see Figure 48), it is in the range of false positive rate greater than 20% which is already way more than we can tolerate. Essentially, we only care about maximizing the area under the ROC curve from 0% FPR to 20% FPR.

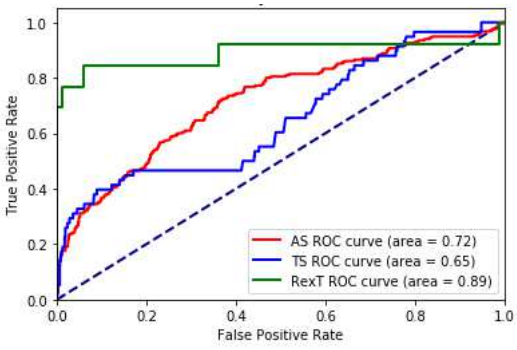


Figure 48: Logistic regression ROC curve

Alogrithm Complexity

See `Scikit-learn` documentation.

Conclusions

This performed pretty well on the RexT data, but that is most likely due to the relatively few data points we have. The performance is pretty weak for AS and TS data; perhaps with more data to average the opinions of for those series, we can obtain better results, but for now, we should look to other methods.

K-Nearest Neighbors (KNN)

As its name implies, k-nearest neighbors finds the k most similar observations to each point (using distance–like Euclidean or Manhattan) and classifies each point based on what its neighbors are classified as. Since this method uses distance, you *must* make sure to scale your data. For this model, the best performing set of features was: the last 25 data points min-max scaled and dummy-encoded KPI's.

Note that this method will be performed in Python using Scikit-learn and the Jupyter notebook is available in the repo.

Results

Overall, we achieve a `F1-score` of 0.79 for AS, 0.90 for TS, and 0.84 for RexT; however, when we separate this out by class (alert vs. not-alert), we see that our performance is being carried by the majority class (not alert). Isolating the alert class, our `F1-score` for AS is 0.44, 0.26 for TS (!), and 0.25 for RexT (!). Figure 49 shows the confusion matrix on the test set using thresholds of 35% for AS, 60% for TS, and 40% for RexT.

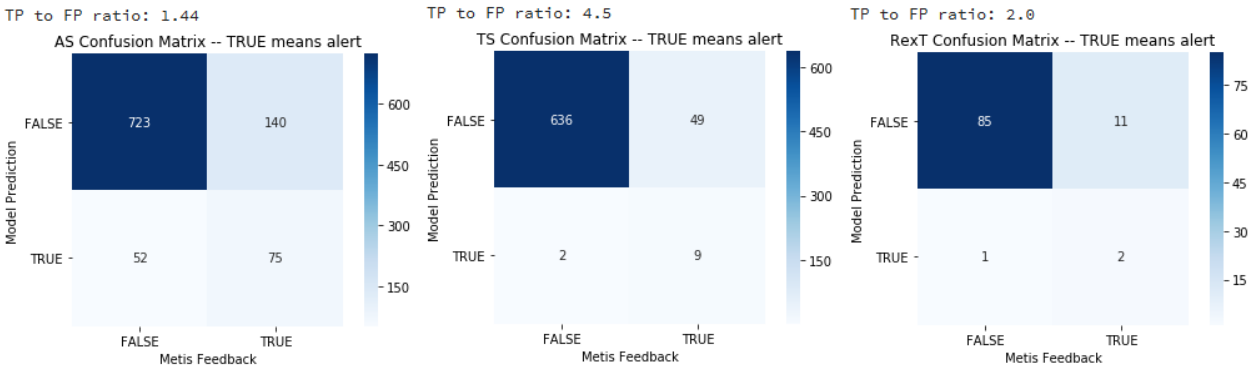


Figure 49: KNN confusion matrix

ROC Curve

As seen in Figure 50, this method was one of the best for the TS dataset even though we see that it could definitely be a lot better. The ROC curve is slower to rise than ideal and even around 20% FPR we are just getting close to 80% TPR. Unfortunately, this means that if we want a good true positive to false positive ratio, we have to allow lots of false negatives which doesn't work for our use case. This is one of the worst methods for the AS and RexT data, altough given that the RexT data is relatively sparse, we will take that with a grain of salt.

Alogrithm Complexity

See `Scikit-learn` documentation.

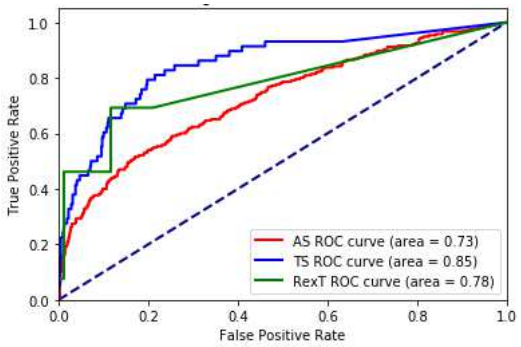


Figure 50: KNN ROC curve

Conclusions

There may be some hope for developing this further for the TS alerts, but there isn't much promise for the other datasets.

Random Forest

Random forest is a bagging method (**b**ootstrap-**agg**regat**ing**); this means we build many weak estimators (decision trees in the case of a random forest) by randomly sampling *with* replacement (hence, the bootstrap part of the name). Once we have all the trees built, we average them (the aggregating part of the name). This method can turn a weak estimator into a strong(er) one. For this model, the best performing set of features was: the last 25 data points min-max scaled and dummy-encoded KPI's.

Note that this method will be performed in Python using Scikit-learn and the Jupyter notebook is available in the repo.

Results

Overall, we achieve a `F1-score` of 0.79 for AS, 0.91 for TS, and 0.96 for RexT; however, when we separate this out by class (alert vs. not-alert), we see that our performance is being carried by the majority class (not alert). Isolating the alert class, our `F1-score` for AS is 0.47, 0.37 for TS, and 0.83 for RexT. Figure 51 shows the confusion matrix on the test set using thresholds of 40% for AS, 35% for TS, and 10% for RexT. These might sound like low thresholds, but this is something we can use to our advantage: we would have lots of leeway in fine-tuning which alerts we send and in the emails and which should be in the dashboard only (alerts that aren't as serious).

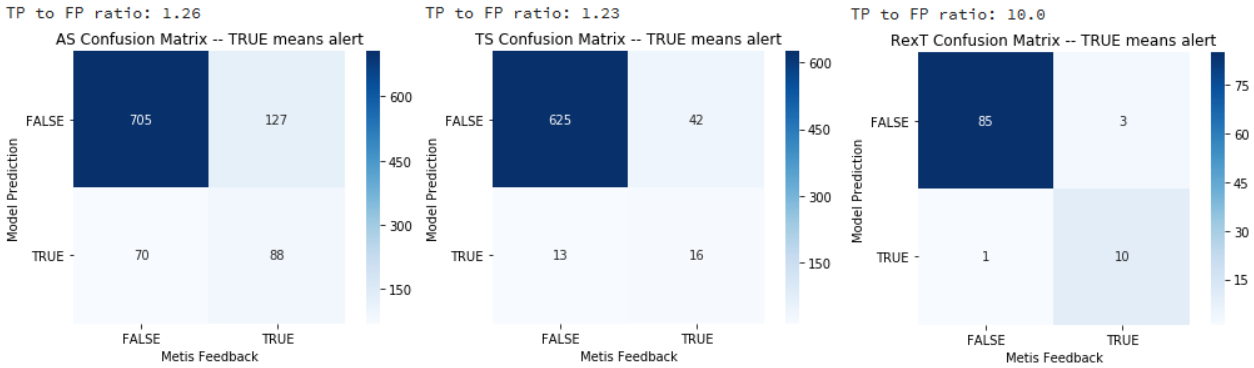


Figure 51: Random Forest confusion matrix

ROC Curve

In Figure 52, we see the ROC curve for the TS alerts increasingly rapidly; however, since this dataset has the fewest alerts (about 8%), we still aren't optimal. This method would not work as-is for AS, but once again this model is fine for the aggregated RexT data.

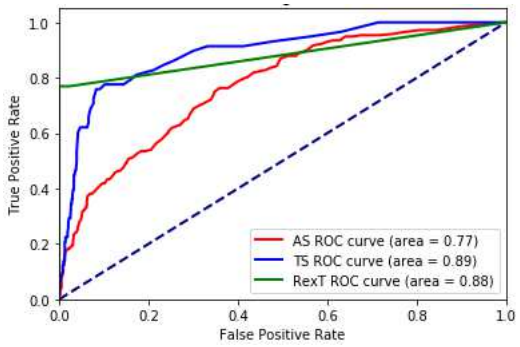


Figure 52: Random Forest ROC curve

Alogrithm Complexity

See `Scikit-learn` documentation.

Conclusions

This performed pretty well on the RexT data, but that is most likely due to the relatively few data points we have. The performance is pretty weak for AS and TS data (despite the promising ROC curve); perhaps with more data to average the opinions for those series, we can obtain better results, however, for now, we should look to other methods.

Gradient Boosted Decision Trees (GBDT)

This method, like the random forest, takes many weak estimators and makes a stronger one. However, boosting and bagging carry out this process differently. Gradient boosting iterates on the estimators where bagging will keep making new ones. The gradient boosting decision tree algorithm trains a decision tree on the data and then checks what it classified wrong; with this data, it weighs those incorrectly classified observations higher and then trains again. This process repeats with the idea that the resulting model gets stronger and stronger (but you have to avoid overfitting). For this model, the best performing set of features was: the last 25 data points min-max scaled using linear weights and dummy-encoded KPI's.

Note that this method will be performed in Python using Scikit-learn and the Jupyter notebook is available in the repo.

Results

Overall, we achieve a `F1-score` of 0.80 for AS, 0.91 for TS, and 0.96 for RexT; however, when we separate this out by class (alert vs. not-alert), we see that our performance is being carried by the majority class (not alert). Isolating the alert class, our `F1-score` for AS is 0.46, 0.39, and 0.82 for RexT. Figure 53 shows the confusion matrix on the test set using thresholds of 50% for AS, 50% for TS, and 0.1% for RexT.

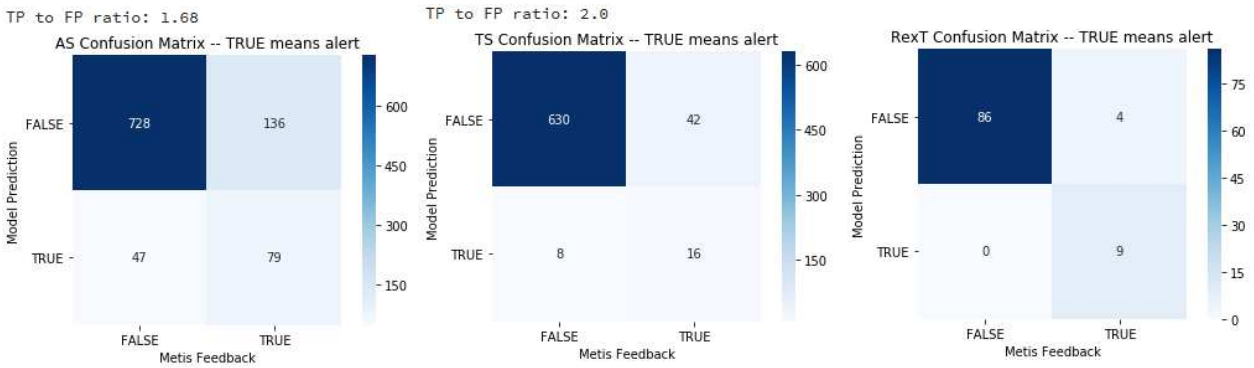


Figure 53: GBDT confusion matrix

ROC Curve

As seen in Figure 54, this method was one of the best for our datasets even though we see that it could definitely be a lot better. ROC curves are, unfortunately, still rather low in our target FPR range (0-10%).

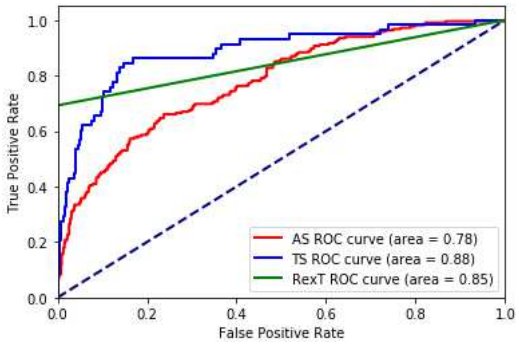


Figure 54: GBDT ROC curve

Alogrithm Complexity

See `Scikit-learn` documentation.

Conclusions

This method is very commonly the top-performer in a variety of use-cases because it "learns" from its mistakes in classification. The only issue here is that our data is quite noisy. This method should be revisited when we have enough data so that most time series have been classified a few times over and the classification opinions averaged.

Multi-Layer Perceptron (MLP)

A multi-layer perceptron is a feedforward neural network of at least 3 layers of nodes. Input is fed in one end and each layer fits the data giving weights to each of its nodes. Then the weights are passed to the next layer for another fitting and weight calculation of those nodes. This keeps repeating until the last layer outputs the result by combining the ending nodes. If this sounds like a black box, that's because it is! For this model, the best performing set of features was: the last 25 data points min-max scaled and dummy-encoded KPI's.

Note that this method will be performed in Python using Scikit-learn and the Jupyter notebook is available in the repo.

Results

Overall, we achieve a `F1-score` of 0.76 for AS, 0.89 for TS, and 0.83 for RexT; however, when we separate this out by class (alert vs. not-alert), we see that our performance is being carried by the majority class (not alert). Isolating the alert class, our `F1-score` for AS is 0.46, 0.44 for TS, and 0.22 for RexT (!). Figure 55 shows the confusion matrix on the test set using thresholds of 60% for AS, 40% for TS, and 7% for RexT.

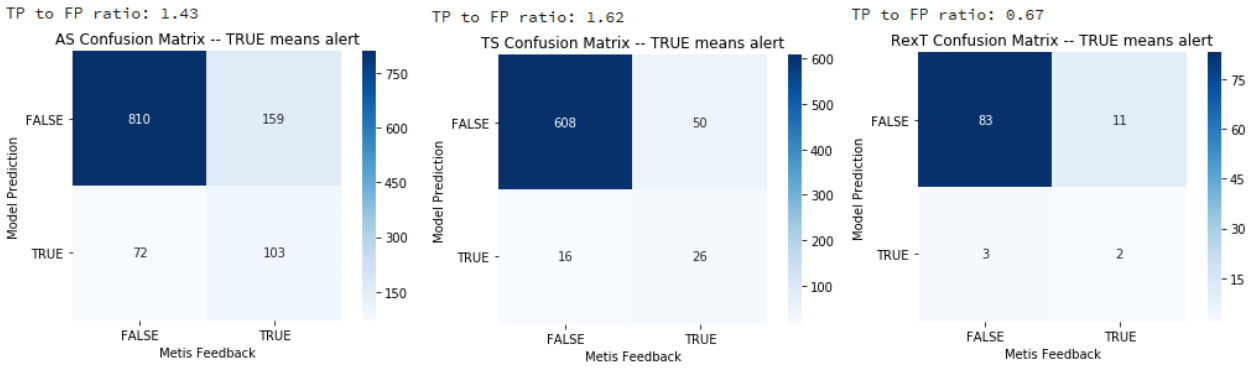


Figure 55: MLP confusion matrix

ROC Curve

What I find most interesting about this method is how much trouble it has with the RexT data, which pretty much every other method had an easy time with (see Figure 56). Not only is it bad, but for much of the ROC curve it is worse than randomly guessing!

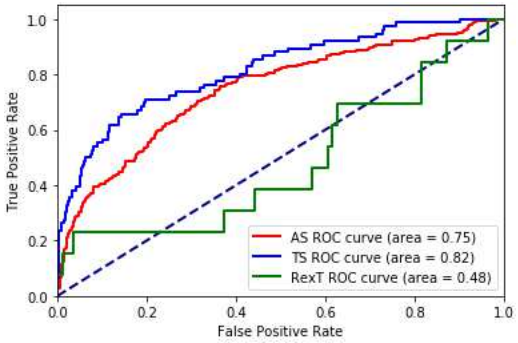


Figure 56: MLP ROC curve

Alogrithm Complexity

See `Scikit-learn` documentation.

Conclusions

MLP wasn't anything special for this dataset, but the reason I wanted to showcase it was to draw attention to how well it performs on the AS data that we disqualified from being eligible for the training set (because we needed 25 days of data and it had some number below that). In Figure 57, we finally see a ROC curve that we have been trying to achieve (the AS curve). Here, at a 60% threshold, we have over a 5:1 true positive to false positive ratio, and we catch way more alerts than we miss (true positives vs. false negatives). For this reason, I briefly toyed with the idea of training the model as presented here, and then manipulating the

non-disqualified data, so that it would be disqualified. It is most likely due to the noisiness of the data that this happened, but it's interesting nonetheless.

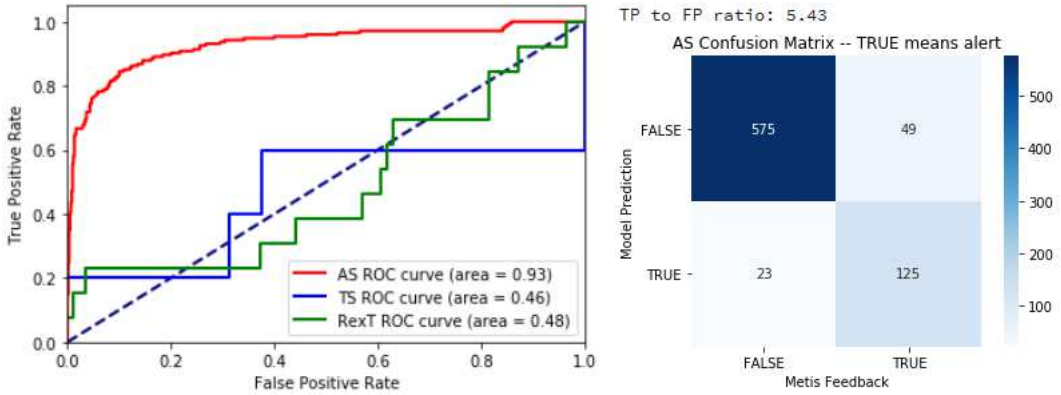


Figure 57: MLP performance on disqualified AS data

Conclusion

The selection of a machine learning model has proven to be a tough task, partly because of the difference of user opinions and not having collected enough opinions on the same series. Since I am not able to definitively name a better model at this point, I leave the reader with possible avenues of continued research and pointers on what to look for when judging future models. Gradient Boosted Decision Trees (GBDT) showed a lot of promise and often performs well in these situations of imbalanced classes. Although the neural network also showed promise, GBDT has the benefit of being more easily interpreted; with neural networks you are getting a black box which isn't always the best solution (i.e. if you need to explain the inner workings to others). For further investigation, I suggest:

1. Collect more data from `Metis` and re-test results of top methods.
2. Try adding a feature of sliding window for the mean (you can try overlapping windows and non-overlapping ones).
3. Try combining logistic regression with GBDT and separately with Random Forest. *See this article for reference.*
4. Additional feature engineering.

When evaluating your models, make sure your true positives are more than your false positives by a good amount. I was using a ratio of 2:1, but this is more subjective; false positives are very costly with the emails, so I was targeting 1 out of every 3 at most being a false positive. Another thing I was looking for was a high true positive rate with a low false positive rate; this is because we also need to control our false negatives–it's not very helpful if we have a good true positive to false positive ratio, but numerous false negatives because those are alerts we are failing to catch(!).