Towards Identifying Impacted Users in Cellular Services

Shobha Venkataraman*
shobhav@gmail.com

Jia Wang
AT&T Labs – Research
jiawang@research.att.com

ABSTRACT

An essential step in the customer care routine of cellular service carriers is determining whether an individual user is impacted by on-going service issues. This is traditionally done by monitoring the network and the services. However, user feedback data, generated when users call customer care agents with problems, is a complementary source of data for this purpose. User feedback data is particularly valuable as it provides the user perspective of the service issues. However, this data is extremely noisy, due to range of issues that users have and the diversity of the language used by care agents. In this paper, we present LOTUS, a system that identifies users impacted by a common root cause (such as a network outage) from user feedback. LOTUS is based on novel algorithmic framework that tightly couples co-training and spatial scan statistics. To model the text in the user feedback, LOTUS also incorporates custom-built language models using deep sequence learning. Through experimental analysis on synthetic and live data, we demonstrate the accuracy of LOTUS. LOTUS has been deployed for several months, and has identified the impact over 200 events.

CCS CONCEPTS

• **Computing methodologies** → *Artificial intelligence*.

KEYWORDS

data mining, semi-supervised learning, customer care

ACM Reference Format:
Shobha Venkataraman and Jia Wang. 2019. Towards Identifying Impacted Users in Cellular Services. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3292500.3330711

1 INTRODUCTION

Commercial cellular networks typically have customer service centers set up to handle feedback from users. When a service problem occurs and users are impacted, groups of affected users tend to call into the customer service centers simultaneously. If this data could be mined, it would provide network operators the users' perspective of service-impacting problems. Such user perspective would provide a complementary viewpoint to that of standard service monitoring tools. This could, for example, help service operators understand if there are unexpected issues from the user perspective even if the service is functioning as expected, or what issues users are experiencing under a specific service problem.

In this paper, we study how to identify the users affected by a service-impacting problem from user feedback on nation-wide cellular network in the US with over a hundred million users. We define an *event* to be a service-impacting problem that affects multiple users, typically within a bounded geographic area, e.g., a network outage, a device software bug, third-party server issues. We define the *user impact* of an event to be the users affected by the event. When a user calls in with a service problem, the following workflow is typical: first, the user is directed through an automated system to the appropriate first-tier team. If the user's issue is technical (as opposed to billing, administration, etc.), the first-tier team has the user run some standard technical tests and checks for any known outages. If a technical cause is not easily discovered, the problem is escalated to second-tier technical teams for investigation. Throughout the process, agents write a summary of the user's problems and the troubleshooting steps taken. We define this agent-generated summary to be a *case*. We define the cases pertaining to an event as *event-specific cases*, and the remaining cases as *non-event cases*.

When an event starts and impacted users start calling the customer service center, service operators want to learn of the event as soon as possible. However, each individual agent does not have the global picture: each agent responds only to a relatively tiny number of calls, and agents do not have the time to discuss possible emerging events (and may not even be co-located at the same worksite to have any opportunity to communicate). Thus, with the few cases that an individual agent may be aware of, we want to identify the user impact of the event. Formally, the problem of identifying user impact is the following: *given an event e, a set U of unlabeled cases and a small number of labeled event-specific and non-event cases L, find as many other cases pertaining to e as possible from the set U.* Note that our goal is to identify the individual cases that constitute the user impact, not simply report whether or not there is an event.

Challenges. The problem of understanding user impact is naturally a machine learning problem, but the nature of the user feedback data makes it infeasible to directly apply standard techniques. To explain the challenges involved, we first provide some high-level background about the user feedback data, and then describe why the direct use of standard techniques would not solve our problem.

Data Challenges. The user feedback data comes from the communication between the user and the agent, and there are many points for disparity in language that captures the communication. Some differences come from variations in individual user and device behaviour, while others come from variations in agent behaviour. The scale of the event-specific cases also poses challenges. We describe these briefly here, and defer the details to Section 2:

*Work done while at AT&T Labs – Research.

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330711

(1) User Diversity: There are a wide variety of users, and a problematic event might appear different to different types of users or even in different geographic locations. There is a wide range of device models in use, and device behaviour may also depend on its configuration, operating system, application usage, etc. Moreover, users may not know when their device started misbehaving, and may not accurately report the problem symptoms they observe on their device. We also obtain only an approximate zipcode as an estimate of a complaint's location, and it is unlikely to reflect the exact latitude/longitude of the problem.
(2) Agent Diversity: A major challenge comes from the natural variation of language data: two agents could speak to users with identical problems and still describe them very differently in the case notes. Inconsistent case handling by agents adds additional disparity. Some records contain many details about the problem description, debugging steps taken and results; many others contain only a high-level summary of the conversation with the customer. The text also typically contains spelling mistakes, domain-specific abbreviations and acronyms, which change over time. Moreover, agents may have access to different kinds of information when resolving a case, and so process them differently.
(3) Scale of the Event: There are many reasons (technical and non-technical) that users contact customer service agents, and the vast majority of the cases are not event-specific – an event normally contributes only a small portion of the total volume of cases observed during the time window of the event. Events can also be tiny, on the order of tens of cases, and thus consist of too little data to even learn from.

Methodology Challenges. These data challenges make it impractical to directly apply standard learning techniques to solve our problem. First, due to the diversity of user feedback data, supervised learning is impractical – supervised learning would need a representative sample of the event-specific cases in order to generalize well, and having domain experts manually label sufficient event-specific cases is impractical. Purely unsupervised techniques (e.g., keyword analysis, clustering, heavy-hitter algorithms) also are impractical in our problem, due to both scale and diversity reasons.

Our problem is a natural fit for semi-supervised learning, since an individual agent can provide us with their few known event-specific cases. However, unlike standard semi-supervised learning, we do not always have enough unlabeled data to learn from in an event. Because semi-supervised learning depends on the underlying structure of the data, it requires enough unlabeled data to be able to infer the structure of the different classes. With events as small as few tens of cases, the user feedback data is too diverse to allow a straightforward use of semi-supervised learning algorithms.

Our approach. In this paper, we take a semi-supervised approach to the problem of identifying user impact. We design a custom framework with diverse machine learning and statistical components to address our specific challenges. Our key insight is that event-specific cases have multiple dimensions of similarity between them, some of which are non-redundant.[1] This property allows us to build tight clusters of cases in one dimension (starting with the little labeled data), and then use those clusters to grow clusters in the other dimensions iteratively. Thus, even though the little labeled data is not representative, it provides a starting point for the framework, and from this starting point, we can find a sequence of clusters that constitute the user impact.

[1]Note that due to the presence of non-redundant dimensions of similarity, our problem cannot be solved solely with co-training [3].

Attribute	Values
Need	90
Subneed	475
Subsubneed	1367
Task Resolution	3334
Device Model	4168

(a) Number of unique values for attributes

Problem Description	Values
Call connectivity	8
Call quality	4
Text	8
Data connectivity	6
Data speed	5

(b) Similar values for some problems in the *Subsubneed* field

Figure 1: Properties of Case Attributes

More precisely, our main algorithmic idea is to decompose the problem into two tightly-coupled sub-problems, co-training [3] and spatio-temporal clustering and repeatedly use the results of each to bootstrap the other. We enhance this core algorithmic framework with word vectors and deep sequence learning models; this lets us address the practical challenges that come from the noisy case text, particularly for small events. The unusual language used by the care agents (detailed in Section 2) requires that we custom-build all the language models for our data.

Contributions. We have designed, prototyped and deployed a practical framework that identifies the user impact of the event, called *LOTUS* (for *Location Trouble Scanner*). With measurement analysis of the user feedback, we identify the unique characteristics of our data. On the basis of these characteristics, we develop a novel algorithmic framework to solve our problem, with co-training, spatial scan statistics, word vectors and deep sequence learning. With experimental analysis on real and synthetic data, we show LOTUS is highly accurate, even when provided as few as 10 labeled event-specific cases. LOTUS has been deployed since August 2018 in a field trial for multiple care agent groups and operates in near real-time. It has identified the impact of over 200 events, and has discovered both typical and unusual user perspectives.

We believe that our work offers insights into practical challenges in analyzing user perspective from data recorded at large customer care centers, as well as the workflows necessary to address these challenges. Our learning framework may also be of interest in other problems with multiple non-redundant dimensions of similarity.

2 PROBLEM BACKGROUND

In this section, we first describe our data sources, some of their unusual properties and the resulting challenges. We then present a case study to illustrate some characteristics of our problem.

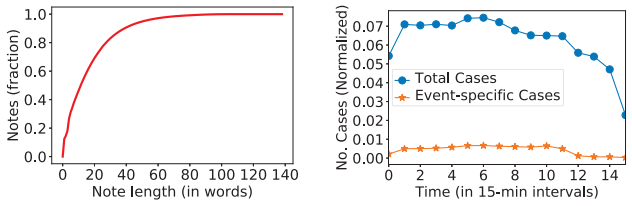
2.1 Data Description

Our data consists primarily of two sources: customer care case data and a user-provided location data. Our analysis in this section is performed on 13.1 million cases collected over an extensive long time duration when users spoke to service representatives of a large mobile network provider. All sensitive and private information was removed before any processing was done on this data.

Customer care case data. When a user experiences a problem or issue and calls the customer care center for assistance, a care case is recorded by the care agent. Each care case includes: (1) a set

Word	Synonym phrases found in case notes
customer	cx, cust, cus, cu, cxci, cci, ccio, cco, cic, ccii, cciu, ci, ccito, cvci, ccoi, cici, ccin, ccui, cxci, ccvi, cfci, cxcalled, custcalled, ccfi xcci, ccim, cdci, ccit
signal	bars, singal, strength, strenth, reception, strengh, siginal, sig, signa, singnal, recption, bar, receiption, sgnal, serviceat, sigl, signl, sgnl, signals
trouble	trbl, troble, touble, truble, troube, troubl, troule, trble, issue, issuses, issu, issueswith, prob, problem, prblems, promblems, problmes, difficulty, probelms

Figure 2: Sample words along with their equivalent phrases in the case notes, illustrating the unusual vocabulary



(a) Case note lengths (b) Case volumes during event

Figure 3: Care case statistics

of *attribute* fields, where agents choose from a set of pre-defined values (2) a free *text* field, where agents may write in a description of the user's problem and the actions they took to resolve it. Both sets of fields are customarily populated in each case. Crucially, for a large number of cases, the case attributes and the case notes both individually contain enough information to deduce the high-level service issue experienced by the user.

The attributes of a case include information pertaining to user identity, user device, agent identity, agent site location and workgroup, the problem the user had (at multiple levels of granularity), the resolution the agent performed, and whether follow-up is required. Figure 1(a) shows the list of attributes that we use in this paper, and the number of unique pre-defined values they each have. The fields *Need*, *Subneed* and *Subsubneed* describe the user's issues at increasingly finer levels of granularity. The field *Task Resolution* describes the action taken by the agent, and the field *Device Model* contains user's device hardware/software model and version.

There are several challenges in analyzing these case fields. First, several of the values for given attribute are similar (e.g., there are multiple names for the same problem, in slightly different language). Figure 1(b) lists the number of similar values for the some commonly reported problems in the *Subsubneed* field. Second, there are also a number of values which are ambiguous, covering a mixture of multiple problems. Third, the case notes contain many domain-specific terms and abbreviations, as well as many spelling errors. To illustrate the range of abbreviations used for just a single word, we give some examples in Figure 2. Such abbreviations are interspersed throughout the notes. Figure 3(a) shows that the agent notes are quite concise, with over 40% of the notes under 10 words. Because these notes are written with their own vocabulary, sentence structure and idiosyncrasies, models trained on regular English language samples will not be accurate on these notes.

Location data. We obtain a zipcode for each user with their case, as an estimate of their most frequent location. In each individual case, this zipcode is unlikely to accurately reflect the exact problem location, but is likely to be within the same metropolitan area. Examining these locations in aggregate is thus likely to give us useful information to assess the user impact of an event. In this work, we examine the aggregation of these locations *county subgroups*, a well-defined entity from the census measurements which subsumes a city or a suburban living area [1].

2.2 A Motivating Case Study

We show a real-world network outage event case study to illustrate the challenges and characteristics of our problem. The outage event lasted several hours during which some users were not able to make calls or send texts into or out of the affected region, or use data while in the affected region. We obtained the ground truth of this event using supplementary information from network logs.

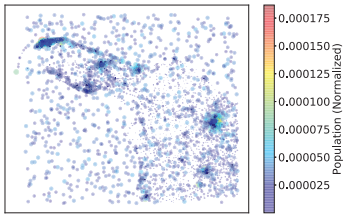
Figure 3(b) shows the normalized time series of the event cases and total cases in 15-minute intervals during the event (for proprietary reasons, the raw numbers are normalized by the total case counts seen in the duration of the event). Note that the event cases form a tiny fraction of the care cases seen routinely during the same time window. On the other hand, the total case volumes exhibit periodicity and predictability – they have a clear diurnal pattern as well as a weekly pattern (no graph shown due to space constraints).

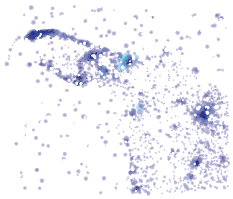
Figure 4 shows the user population, total case counts, and event case counts during the day of the event, at each user-provided zipcode. (For proprietary reasons, the locations have been anonymized while maintaining the location structure of the population; the raw case counts have been normalized by dividing by a number). Note that while the first two maps are very similar, they are substantially different from map for the event cases. In particular, there are quite a few densely populated locations in first two maps that are not present in Figure 4(c). These maps illustrate the spatial structure inherent in the event cases.

Figure 5 lists some sample notes from the event cases. We note there is a diverse set of problems reported by users, including problems with outgoing calls, incoming calls, service quality, no service at all, data connectivity, and voicemail. The notes describe the problems in a variety of writing styles, vocabulary sizes and note lengths. Synonym phrases are heavily used in these notes. Many notes describe multiple problems, even though the case attributes can define only a single problem. Indeed, 98% of the notes from the sample event are distinct (even after text standardization and stemming). Identifying the user impact of the event thus requires identifying cases across all these diverse kinds of cases.

3 LOTUS FRAMEWORK

The main high-level steps of LOTUS are the following:
Step 1: Feature extraction: We construct examples consisting of both case attribute and case text views by extracting features from each of them. To accomplish this step effectively, LOTUS pre-trains word vector models [22]. We describe this step in Section 4.1.
Step 2: Co-training: In this step, we learn to distinguish the event-specific cases from the normal cases on a fixed set of space-time windows. We use co-training to find *separate* classifiers for the text features and attribute features using the labeled and unlabeled data. We also enhance the text classifier by pre-training language models with deep sequence learning. We describe this step in Section 4.2.


0.000175
0.000150
0.000125
0.000100
0.000075
0.000050
0.000025
Population (Normalized)



representation over a character-based representation since many different words in our data (e.g., device names, apps, locations) have similar contexts, and we want to capture that similarity.

In LOTUS, we pre-train words into 300-dimensional real vectors using a few months of historical case notes. We choose 300 dimensions because most words in our vocabulary have far many more frequently-used abbreviations than regular English. We build these vectors using the contextual bag-of-words model with hierarchical softmax, with a window of 4 words before and after the word. We use this short window because agents tend to write concisely, as seen in Figure 3(a). With this short window, many case notes will have comparable text windows for learning word contexts.

4.2 Co-training

We now describe how we learn from the examples constructed with both attribute and text features using co-training.

4.2.1 Co-training Overview. Co-training [3] is an algorithm that performs semi-supervised learning on data that has two redundant sets of features. The key assumptions are: (1) each set of features be *individually* sufficient to represent an accurate classifier on the data, and (2) that the two sets of features are independently generated when given the class label. Section 2.1 explains how the first assumption is met, i.e., the attributes and the text are both individually sufficient to represent an accurate classifier that distinguishes event cases and normal cases. The second assumption implies that the attributes and case notes are chosen independently from semantically equivalent options (i.e., conditioned on the class label); the large number of care agents with minor behavioral differences ensures that this assumption is met in aggregate.

Co-training simultaneously learns a text-only classifier S_t and an attribute-only classifier S_a, and uses the unlabeled data to bootstrap learning by forcing the text-only classifier and the attribute-only classifier to be consistent. To use co-training, we need to specify the two supervised learning algorithms S_a and S_t. Below, we briefly describe our algorithm for learning S_a. Learning S_t is more challenging and we describe it in Sections 4.2.2 and 4.2.3.

For learning S_a, we first observe that a few groups of key attributes can essentially carve out the positive examples from the data, i.e. a few sets of combined values for attributes *Need*, *Subneed*, *Subsubneed* and *Task Resolution*. Thus, accurate classifiers that capture the event in a fixed space-time window can be encoded by DNFs [24], i.e. a disjunction of conjunctive clauses. In LOTUS, we use 3-DNFs, i.e., a DNF such that each conjunctive clause has 3 variables.[2] We choose this classifier because we can then avoid creating features for every possible attribute value; instead, we need to only create features for those values necessary to represent the event cases. We use a standard algorithm for learning DNFs [15] and for completeness, we describe parameters in Appendix A.

4.2.2 Learning Language Models. A major challenge with using semi-supervised learning in LOTUS is that many events are very small. For example, an event in a small town can have just a couple of dozen cases, and their case notes can consist a variety of writing styles. Then, the data in the analysis window is likely insufficient for learning a complex text classifier from scratch. However, in most events, users typically face one of a few technical problems (i.e., voice, data, etc.), and so the case notes, despite their diverse writing styles, are mostly about these few technical problems. If we can pre-train a text classifier for these common technical problems, we can select one or more of these classifiers for S_t, instead of learning a complex text classifier from scratch.

The historical case notes are often sufficiently representative of the diverse language of a technical problem, but it is difficult to label the notes comprehensively through manual effort. However, the case attributes offer an independent description of the complaint, which we can interpret as a noisy label on the case text. Because of the ambiguity and overlap among the attributes, many case attributes can reflect overlapping, similar, or multiple problems (as discussed in Section 2.1). But for our purposes, we do not need to use all the attributes or comprehensively label all the case notes; we only need a data set of case notes sufficiently representative of the technical problem so we can build a good text classifier.

To obtain this data set, we choose a few unambiguous and non-overlapping case attributes for each technical problem, and use their case notes as noisy training data for that particular problem.[3] This way, instead of using all the historical data, we have selected a subsets of the data for supervised learning on each technical problem. These subsets are still noisy, but we find that they are accurate enough to learn a representative language model for each technical problem.[4] Because many case notes are ambiguous and appear with multiple technical problems, we create a manually-labeled validation set of 200 cases for each technical problem.

We use deep learning to train 8 language models: 7 are for technical problems, i.e. *voice, data, text, e-mail, wifi, hotspot, wifi calling*; 1 is for a commonly-used resolution during events, which the agent informs the user of an ongoing outage, and we term this *known outage*. The unusual text of the case notes requires that we train our own language models from scratch, and we cannot use pre-trained off-the-shelf English language models. We use standard models and learning algorithms found appropriate for sequence learning [9, 10]. Specifically, we use a 3-layer network: (1) a 128-node LSTM [12] with standard activations; (2) a 16-node fully connected hidden layer sigmoid activations; (3) a 1-node output layer with sigmoid activation. For completeness, we specify full parameters of the models and their training in Appendix A.

4.2.3 Co-training with Language Models. Finally, we describe how we incorporate these language models into co-training. We first describe co-training algorithm of [3] formally. Let $x = \langle f_a, f_t \rangle$ denote an example constructed from the case, with f_a denoting the set of features from attributes and f_t denoting the set of features from the case text. Let L denote the initial labeled data, and U denote the unlabeled data. Recall S_a and S_t denote the supervised learning algorithms used over the attribute features and text features respectively. The co-training algorithm first uses S_a to learn a classifier $h_{a,1}$ over the attribute features for L, and S_t to learn a classifier $h_{t,1}$

[2] For example, a boolean function *(Need1 ∧ Subneed1 ∧ Subsubneed1) ∨ (Need2 ∧ Subneed2 ∧ Resolution2)* is a 3-DNF. Used as a classifier, it encodes that either the set *[Need1, Subneed1, Subsubneed1]* would all need to be present in the attributes of the case, or the set *[Need2, Subneed2, Resolution2]* would all need to be present.

[3] We note that our goal is not to predict the case attributes, but merely to distinguish between pre-defined groups of selected case attributes.

[4] Note also that the labels obtained from case attributes are different from labels that indicate whether a case is event-specific. The labels we obtain from case information are only useful for building language models, and apply only to a subset of the data.

over the text features and L. It then uses the classifier $h_{a,1}$ and $h_{t,1}$ to label the examples in U. Let $U_{a,1}$ (and likewise, $U_{t,1}$) denote the examples that the classifier $h_{a,1}$ (likewise, $h_{t,1}$) is most confident on. Now, co-training uses the data set $L \cup U_{t,1}$ as the input to S_a to now learn a new classifier $h_{a,2}$, where the labels from $U_{t,1}$ have come from the classifier $h_{t,1}$. It repeats this process until no new examples in U can be labeled.

We modify co-training to use the language models by using the data to select among the available models, rather than training S_t from scratch. Let $M_1, M_2 \ldots$ denote the language models. Formally: at each step i, we create a set $\mathcal{D}_{t,i}$ of valid language models M_j over the data set $L \cup U_{a,i}$ as follows: we include in $\mathcal{D}_{t,i}$ any model M_j with a substantial accuracy (e.g., 10%) on the event-specific cases and a high accuracy on the non-event (e.g., 95%). We then define S_t to be an *OR*-function of all the models selected in $\mathcal{D}_{t,i}$.

Only if there is no language model in $\mathcal{D}_{i,t}$ at any step i will LOTUS learn a text classifier S_t from scratch. In this case, we set S_t to be an ensemble created with the majority vote of perceptron, random forests and stochastic gradient descent. We use an ensemble with a variety of classifiers so LOTUS can easily learn complex hypotheses accurately (parameters in Appendix A).

4.3 Spatio-temporal Localization

After co-training, we have a set of event-specific cases in the current space-time windows under study. The spatio-temporal localization step finds other space-time windows with an unusually high number of cases similar to the co-training output. To do so, we use spatial scan statistics[17] with the SatScan software. It implements a likelihood test using the underlying population counts to compare the observed data with selected baseline statistical models.

We use SatScan in LOTUS with the Poisson model. For the model to apply, we need a few assumptions to hold. First, the probability that an individual user creates a case needs to be much higher during an event than during normal times. Second, each user creates a case independently of other users. Third, the probability that an individual user creates a case is similar across spatial units; thus, the number of cases in a spatial unit is always a function of its population. The first assumption is clearly valid; in fact, so much so that LOTUS works even when the other two assumptions are mildly broken. We also mitigate the third assumption by normalizing the cases in each space-time window with its 10%-trimmed rolling historical average (over 8 weeks).

To use SatScan, we consider a large geographic region (typically a few neighboring US states) and a large time window (e.g., a few days) that are expected to contain the entire event. We estimate the number of event-specific cases in each county subdivision seen every hour in this larger region using the classifiers S_a and S_t from the co-training step. We then apply SatScan and obtain spatio-temporal units with unusually high numbers of event-specific cases. We use these discovered spatio-temporal units as the new set of analysis windows in LOTUS, and we return to the co-training step. Additional parameters are included in Appendix A.

4.4 Supplementary Analysis

After multiple iterations between the co-training and the spatial localization steps, LOTUS identifies a set of event cases. LOTUS finally performs a statistical test to better understand whether an event is specific to a particular device model. For this analysis, we track the historical population sizes of the top 100 device models on an ongoing basis. We compare the device model proportions in the final event cases and their respective historical proportions with a z-test, and report outliers.

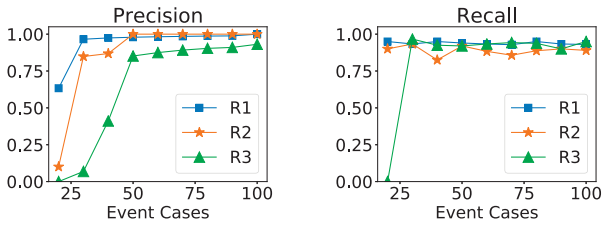


Figure 7: Synthetic Data Results in Regions R_1, R_2 and R_3

5 EXPERIMENTS

We have implemented a prototype of LOTUS that operates in real-time. Our prototype is currently deployed and can can analyze a day's worth of cases (on the order of hundreds of thousands of cases) in around 15-20 minutes on a Linux-based 2.20Ghz high performance system with 32 cores and 1TB RAM. In our evaluation, we first show LOTUS' accuracy on synthetic data (Section 5.1) and real data (Section 5.2), and then describe our deployment (Section 5.3).

5.1 Evaluation with Synthetic Events

Our synthetic experiments compare the accuracy of LOTUS across events of different sizes and over regions of different populations.

5.1.1 Generating Synthetic Events. We generate synthetic data by adding event cases from a known (manually validated) real event with over 2000 cases into background normal data. More precisely, we do the following: (a) we select a region to place the synthetic event, (b) we select an event case at random from the known real event, and (c) we choose a location for the case by selecting a zipcode in the affected region, as a function of the user population. We use as background normal data all cases from a time interval with no major events; this consists of over 357.8 thousand cases.

Spatial Regions. We choose three regions to illustrate LOTUS's accuracy as a function of the background population. The three regions, labeled R_1, R_2 and R_3, have populations in the order of ten thousand, hundred thousand and a million respectively. We restrict the event to be within a 3 mile radius of the selected zipcode (since this is the size of our smallest region).

Experimental Setup. For each region, we vary the number of added event cases from 20 to 100. We select (at random) 10 labeled event cases as positive examples, and 1000 labeled normal cases as negative examples. We give these labeled cases as input to LOTUS along with all the unlabeled cases. Each independent run of LOTUS starts with a different random set of labeled examples.

5.1.2 Synthetic Event Results. Figure 7 shows LOTUS's precision and recall for the three regions across the range of event sizes. We see that in all three cases, the accuracy of LOTUS improves as the number of event cases increases. The event size at which LOTUS achieves acceptable accuracy (i.e., high precision and high recall) depends heavily on the underlying population. Consider the sizes

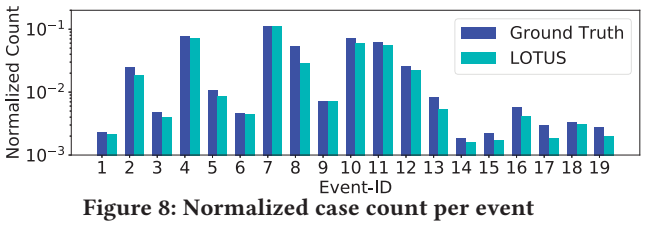


Figure 8: Normalized case count per event

at which LOTUS's precision and recall exceed 75% for each of the regions. For region R_1, this occurs with as few as 30 event cases; for regions R_2 and R_3, it is around 50 event cases.

Note also that the recall is typically high (except for the tiniest events in R_3). However, the precision is low for the smaller event sizes in R_2 and R_3. This is because background data has a few tiny regions with many event-like cases outside the impacted region (in both case types and volumes), and so LOTUS finds them. In practice, as we see in Section 5.2, the events are far more distinguishable.

5.2 Experiments on Live Data

5.2.1 Obtaining Ground Truth. We obtain ground truth for these events by manually labeling each case in the analysis window of the event. We label the cases into five groups: *Non-Event*, *Confirmed*, *Power-cycled*, *Symptomatic*, and *Miscellaneous*. Cases that contain enough information so we see that they are caused by event are labeled as *Confirmed* (e.g., case notes may say the agent's network investigation tools indicated a network outage in the user's location). Cases that contain enough information so we see that they do not relate to the event (e.g., a forgotten password; instructions on using a device or service feature) are labeled *Non-event*.

A case may not contain enough information to be confidently labeled as *Confirmed* or *Non-event*, because the notes are ambiguous and the attributes are coarse-grained. Quite often, particularly at the beginning of an event, agents are not aware of the event, and thus cannot record incoming cases as part of the event. However, even so, the location has an unusually high number of cases with symptoms similar to the *Confirmed* cases around the time of the event. We label these cases as *Symptomatic*. This label is restricted only to the cases in the event's space-time windows, which we obtain from auxiliary information in network logs. Most *Symptomatic* cases are likely caused by the event.

There is often also a rise in cases that are fixed by rebooting the device during the event window. This situation especially occurs when the underlying network outage has been fixed, but the user's device needs to reboot before it can function normally again. When users call in with complaints, they are instructed to first try rebooting their phone. If the event has ended, this reboot fixes the issue. We label these cases *Power-cycled*. Like the *Symptomatic* cases, these cases are also likely caused by the event. Lastly, any cases that do not fall into the above categories are labeled *Miscellaneous*.

5.2.2 Results on Live Data. We now present our results on 19 events with ground truth. The 19 events are caused by network outages of various sizes and occur anywhere from small towns to large metropolitan areas. The outages result in some loss of service, and their causes range from from fiber cuts and power outages to storms that impact cell towers and microcells. We start LOTUS with 10 labeled event cases (i.e., positive samples) for each of the 19 events. With the input event cases, we automatically generate appropriate labeled normal cases (i.e., negative samples) by sampling from historical data, and discarding cases that are similar (in attributes and infrequent keywords) to the labeled event cases.

Event Scale. Figure 8 illustrates the scale of the events. We compare the event cases in the ground truth (the total of the *Confirmed*, *Symptomatic*, and *Power-cycled* cases) with those discovered by LOTUS. For proprietary reasons, the raw number of cases is normalized (by scaling by a number). Note that the number of event cases span over two orders of magnitude. LOTUS discovers both small and large events (and does so accurately, as we show next).

Figure 9 shows the relative number of cases with each label for the ground truth. Specifically, for each event, we normalize the raw case counts of each label by the total labeled ground truth for that event. We see that the *Non-event* cases constitute the vast majority of cases, especially for the small events. For many events, the *Non-event* cases are an order of magnitude bigger than the remaining cases. Of the remaining cases, the *Confirmed* cases are typically the majority. The *Symptomatic* cases are next in size, substantially exceeding the *Power-cycled* cases. The number of *Miscellaneous* cases per event is marginal.

LOTUS Accuracy Results. We first examine LOTUS results on all event cases. Figure 11 shows the recall for each type of event case. For most events, LOTUS discovers over 80% of the *Confirmed* cases. The recall is a little lower for the *Symptomatic* and *Power-cycled* cases: for 14 events, LOTUS discovers over half the *Symptomatic* and *Power-cycled* cases; for 10 of those 14, the recall exceeds 70%.

The only exception to these results is Event-8, where LOTUS discovers 71% of the *Confirmed* cases and 42% of the *Symptomatic* cases as shown in Figure 11. Event-8 is a challenging event for LOTUS because the spatial units do not align completely with the network failures. Event-8 covers a large geographic area with many rural areas and a densely-populated metropolitan area. Event-8 is caused by a few link failures. The metro area is covered by only a few spatial units, but is served by many links, and so the entire area does not lose service. The increase in event-related cases in the metro area is insufficient compared to its population, and so LOTUS does not discover this area during spatio-temporal localization. Indeed, the cases LOTUS missed in Event-8 were all in this metro area.

LOTUS also discovers very few *Non-event* cases. Figure 12 shows that the precision exceeds 95% for most events (i.e., fewer than 5% of the cases LOTUS discovers are *Non-event* cases). The one exception is Event-11, where the precision is 85%. Event-11 was caused by a storm over large geographic area, which affected towers in many spatial units in the area. A number of users in the storm-affected regions called in with cases similar to those of Event-11 (but additional information showed that these cases were *Non-event*). Because the location and the symptoms of these *Non-event* cases matched Event-11, LOTUS reported these *Non-event* cases as well.

Finally, we show the $F1$-measure for LOTUS (and for alternate approaches) in Figure 10. The $F1$-measure for LOTUS is consistently high for all events; it exceeds 75% for 16 of the 19 events.

Comparison with Alternate Approaches. We also compare LOTUS to simpler approaches: vanilla co-training, vanilla SatScan, clustering, and a baseline keyword search currently used by operators. We

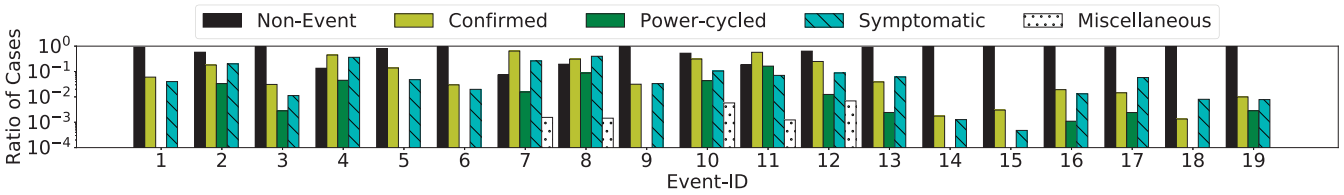


Figure 9: Relative ratios of different types of cases in the ground truth

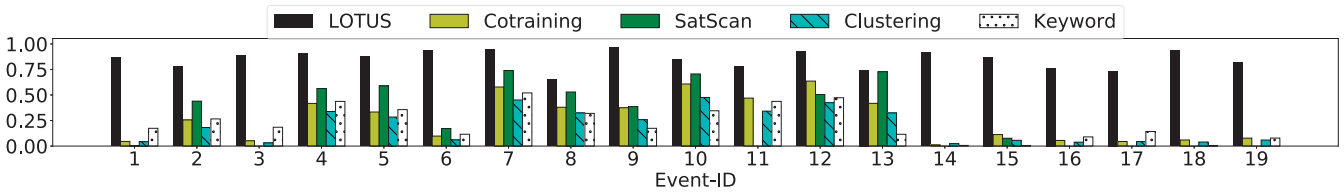


Figure 10: Comparing LOTUS performance with alternate approaches: LOTUS has highest F1-measure on all 19 events

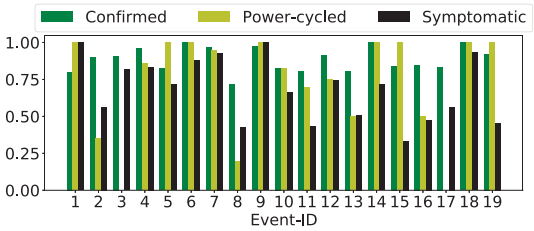


Figure 11: LOTUS accuracy: Recall

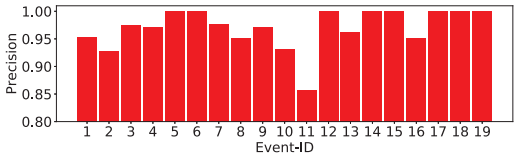


Figure 12: LOTUS Accuracy: Precision

sketch each algorithm briefly here. First, vanilla co-training runs co-training alone on the case data, started with the same labeled data and space-time windows as LOTUS. Next, consider vanilla Satscan. Satscan is unsupervised, so we first simply run SatScan on the same space-time window as LOTUS. Satscan reports space-time clusters which have significantly higher cases than usual. From these clusters, we select every cluster that has an initial labeled event case, and output all cases in that cluster. Third, we compare to a clustering approach. Here, we run k-means clustering on all the cases in the same space-time windows input to LOTUS using the same features (parameters in Appendix A). We report any cluster of cases which contains an initial labeled event case. Finally, we compare to a keyword search. We use common keywords that match how user-observed symptoms and the outage are typically recorded by agents, and report all cases in the analyzed space-time window that match the keywords.

Figure 10 shows the F1-measure of each of these approaches. In all four approaches, for most events, the F1-measure is substantially lower than LOTUS. The only exceptions are Event-10 and Event-13 where the F1-measure of vanilla SatScan approaches LOTUS; this is because these events are mostly isolated spatially, so it is relatively easy for SatScan to identify them accurately. The F1-measure is so low because the precision is often very low (typically under 25% and never exceeds 65%) for all approaches. These results also show that each algorithmic component of LOTUS (i.e, co-training and SatScan) is unable to accurately identify user impact by itself.

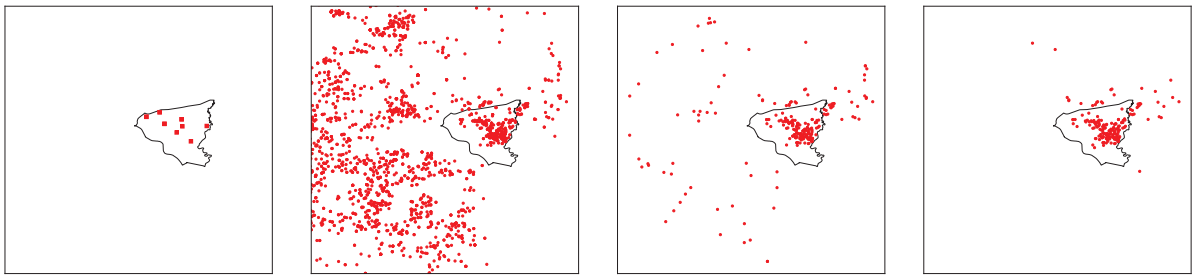
5.3 Deployment

LOTUS has been deployed since August 2018, and has been made available to many care agent teams. LOTUS is triggered through an API that allows the network operators to provide initial labeled cases and to select parameters for analysis. Due to data availability limitations, LOTUS currently operates with an end-to-end delay of 2 hours. LOTUS has assessed the user impact of over 200 events.

Case Studies. We present 2 case studies from the events LOTUS has assessed to illustrate the user impact LOTUS finds.

(1) Unusual User Impact: First, we show how LOTUS provides user perspective that is complementary to the network perspective. A network problem occurred affecting data connectivity in multiple states, and was fixed a few hours later. The network problem resulted in a large increase in data complaints in the affected region. However, even after the problem was fixed, there was a second large increase in data complaints that followed. This second increase was caused by users who were still unable to use data, but could do so once an agent asked them to power-cycle their devices. LOTUS detected this second increase when compared to the normal as well. Thus, even though network had recovered, there were still issues from the user perspective, and LOTUS provided us this insight.

(2) Typical User Impact: Next, we describe an event in which LOTUS discovers user impact that is aligned with the root cause, i.e., tower outages. For Event-5 (from Section 5.2), we obtained network tickets indicating the towers that suffered an outage. The four maps in Figure 13 show the main spatial unit where the event occurs, with location details anonymized for proprietary reasons. Figure 13(a) shows the locations of cell towers where an outage occurred (relative to the anonymized locations). Figure 13(b) shows the locations of all the cases seen in the surrounding geographic area during the analysis window. Figure 13(c) shows the ground truth for the event. Figure 13(d) shows the cases that LOTUS discovered. We see that LOTUS discovers event cases inside the main affected spatial unit; this is where they are concentrated. But it does not discover event cases outside the affected spatial unit. This is typical; an event usually consists of a few concentrated clusters, surrounded by sparser regions. By design, LOTUS discovers the concentrated clusters.



(a) Tower outages (b) All cases (c) Ground truth (d) LOTUS results

Figure 13: Case study: Anonymized locations of tower outages, all cases, ground truth of Event-5 and LOTUS-identified cases.

Lessons Learnt. LOTUS works well when the spatial unit is aligned with the event. This happens most of the time because the events and the spatial units are usually both aligned with the underlying geography. However, if there are multiple simultaneous events, LOTUS does not always distinguish them well. For example, if two events occur with similar symptoms in neighboring locations, LOTUS will report one (combined) event. In our deployment, these situations rarely occurred and did not pose practical issues. The only practical bottleneck for LOTUS was the availability of real-time data. The closer to real-time that LOTUS operated, the more useful LOTUS became.

6 RELATED WORK

The problem of detecting spatially localized patterns in unstructured text has been studied on a variety of data [7, 8, 13, 21, 29]; however, these study macroscopic trends in spatiotemporal patterns, and do not apply to our small-scale events. [11] presents topic "burst" models, but does not apply when the bursty topics appear in large amounts in the background data.

A major line of research explores diagnosis of customer problems by directly using network metrics [2, 18, 26, 28]. The goal of our work is to understand customer problems from *their* perspective rather than the network perspective, and thus, LOTUS is complementary to research that uses network metrics. Another line of work explores customer feedback directly; we describe the most related here. [6] aims to detect service-impacting events in customer tickets, but focuses on detecting large new patterns in test data that do not appear in training data. [14] studies customer feedback to understand characteristics of customer complaints in cellular networks. [19] studies customer tickets together with network metrics to automate diagnosis in IPTV. There is also research on the automatic transcription of customer care calls, followed by NLP to understand the call content [4, 5, 20, 23, 25, 27]; however, in our scenario, we do not have access to the conversation with customer, and so, LOTUS is orthogonal to these works.

7 CONCLUSION

We presented LOTUS, a system for identifying impacted users of a common root cause from noisy user feedback. LOTUS combines several modern machine learning techniques (co-training, spatial scan statistics, word vectors and deep sequence learning) in a novel semi-supervised learning framework. LOTUS is highly accurate and has been deployed for use by customer care agent groups.

REFERENCES

[1] United States Census Bureau, 2018.
[2] P. Barford, J. Kline, D. Plonka, and A. Ron. A signal analysis of network traffic anomalies. In *IMW '02*, 2002.
[3] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of COLT '98*, 1998.
[4] R. J. Byrd, M. S. Neff, W. Teiken, Y. Park, K.-S. F. Cheng, S. C. Gates, and K. Visweswariah. Semi-automated logging of contact center telephone calls. In *CIKM'08*, 2008.
[5] A. Chalamalla, S. Negi, L. V. Subramaniam, and G. Ramakrishnan. Identification of class specific discourse patterns. In *CIKM'08*, 2008.
[6] Y. C. Chen, G. M. Lee, N. Duffield, L. Qiu, and J. Wang. Event detection using customer care calls. In *IEEE INFOCOM '13*, 2013.
[7] J. Eisenstein, B. O'Connor, N. Smith, and E. Xing. A latent variable model for geographic lexical variation. In *EMNLP '10*, 2010.
[8] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. KDD '96*, 1996.
[9] Y. Goldberg. A primer on neural network models for natural language processing. *J. Artif. Int. Res.*, 57(1):345–420, Sept. 2016.
[10] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.
[11] D. He and D. S. Parker. Topic dynamics: An alternative model of bursts in streams of topics. In *KDD '10*, 2010.
[12] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.
[13] L. Hong, A. Ahmed, S. Gurumurthy, A. Smola, and K. Tsioutsiouliklis. Discovering geographical topics in the twitter stream. In *WWW '12*, 2012.
[14] Y. Jin, N. Duffield, A. Gerber, P. Haffner, W. L. Hsu, G. Jacobson, S. Sen, S. Venkataraman, and Z. L. Zhang. Making sense of customer tickets in cellular networks. In *IEEE INFOCOM '11*, 2011.
[15] M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA, USA, 1994.
[16] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.
[17] M. Kulldorff. A spatial scan statistic. *Communications in Statistics - Theory and Methods*, 26(6):1481–1496, 1997.
[18] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *SIGCOMM '04*, 2004.
[19] A. Mahimkar, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and Q. Zhao. Towards automated performance diagnosis in a large IPTV network. In *Proceedings of SIGCOMM '09*, 2009.
[20] J. Mamou, D. Carmel, and R. Hoory. Spoken document retrieval from call-center conversations. In *SIGIR'06*, 2006.
[21] Q. Mei, C. Liu, H. Su, and C. Zhai. A probabilistic approach to spatiotemporal theme pattern mining on weblogs. In *WWW '06*, 2006.
[22] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS '13*, 2013.
[23] G. Mishne, D. Carmel, R. Hoory, A. Roytman, and A. Soffer. Automatic analysis of call-center conversations. In *CIKM'05*, 2005.
[24] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1997.
[25] Y. Park. Automatic call section segmentation for contact-center calls. In *CIKM '07*, 2007.
[26] I. C. Paschalidis and G. Smaragdakis. Spatio-temporal network anomaly detection by assessing deviations of empirical measures. *IEEE/ACM Trans. Netw.*, 17(3), June 2009.
[27] S. Roy and L. V. Subramaniam. Automatic generation of domain models for call centers from noisy transcriptions. In *ACL'06*, 2006.
[28] M. Thottan and C. Ji. Anomaly detection in IP networks. *Trans. Sig. Proc.*, 51(8), Aug. 2003.
[29] Z. Yin, L. Cao, J. Han, C. Zhai, and T. Huang. Geographical topic discovery and comparison. In *WWW '11*, 2011.

A APPENDIX

We now present a detailed discussion of the parameters that we used in our framework and experiments to support reproducibility of results.

A.1 Framework Parameters

In this section, we discuss the various parameters used in different parts of the LOTUS framework.

A.1.1 Word Vector Parameters. We use the `gensim` implementation for training word embeddings. As described in Section 4.1, we build 300-dimensional real-valued vectors using the contextual bag-of-words model with hierarchical softmax. We use a window of 4 words before and after the word. We set a threshold of 50 for the word's minimum frequency of occurrence in the corpus. We train with 8 parallel workers. All other parameters are left to their default values in `gensim`.

A.1.2 Co-training Classifiers. Next, we describe the parameters of different classifiers we use inside the co-training component of LOTUS.

Attribute classifier S_a. As mentioned in Section 4.2.1, we use the standard DNF learning algorithm. This algorithm is designed for data with no noise, and we describe it briefly here. It starts with the entire data, and simply selects a conjunction c that matches the most positive examples in the data without matching any of the negative examples in the data. It adds c to the hypothesis (DNF), and removes the matched positive examples from the data. It repeats this process until all positive examples have a matched conjunction in the hypothesis DNF.

Because our data has a lot of noise, we modify this algorithm slightly to be practically applicable. As mentioned in Section 4.2.1, we use 3-DNFs to represent the attribute classifier; this means our classifier is a disjunction of conjunctions, where each conjunction has 3 attribute values. Due to the noise in the data, we choose conjunctions that account for at least 5% of the (current) positive examples, and less than 1% of the (current) negative examples, and add them to the 3-DNF; we never remove from the DNF. This algorithm works because a small number of conjunctions (typically less than 20) are sufficient to represent most of the event cases.

Deep Learning Language Models. As discussed in Section 4.2.2, we pre-train 8 language models for use inside co-training: 7 for technical problems and one for a common resolution. For these models, we used a 3-layer deep network appropriate for sequence learning, and our design choices for this neural network are based on current best practices. We use `keras` with the Tensorflow backend for the neural networks. We describe here the main parameters, and any additonal ones that we changed from their default values.

Our first layer is a 128-node standard LSTM, as LSTMs are a commonly-used layer for learning from sequences. For each cell of the LSTM, we use the default hyperbolic *tanh* function for the input activation, and the default hard sigmoid function for all other gates. The cells are initialized with Xavier initialization for the kernel, a random orthogonal matrix for the recurrent weights and zeros for the bias weights. We also add in a dropout rate of 0.2 for both the regular and the recurrent weights, in order to ensure robustness. Our second layer is a hidden fully connected layer of 16 nodes, with sigmoid activation functions. Once again, the weights are initialized with Xavier initialization, and we incorporate a dropout rate of 0.2 for robustness. Our final layer is the output layer with one node, which is fully connected to the hidden layer of 16 nodes. Once again, we use a sigmoid function for activation. We use sigmoid functions for all activations in the second layer and the output layer, because they train and converge faster, and as the neural network is only 3 layers deep, it does not suffer from vanishing gradient problems. Our models were trained with the Adam [16] learning algorithm, with a learning rate of 0.0001 and logistic loss. All other parameters were set to their default values in the `keras` library.

Ensemble Text classifier S_t. As discussed in Section 4.2.3, we select from among pre-trained language models when possible, and only if none of the language models are suitable do we construct a classifier from scratch. Here, we use an ensemble classifier constructed through a majority vote of perceptron, random forests, and stochastic gradient descent. For each individual classifier in the ensemble, we consider its label as event if its probability prediction exceeds a threshold of 0.85. We use the implementations in the `scikit-learn`library, and describe below where we modify the default parameter values for each algorithm. We modify the weights of classes for all algorithms: we weight the non-event cases 10 times the event cases, to minimize false positives. We use random forests with 100 trees and a maximum depth of depth of 4, and set a minimum leaf size to be 0.005% examples. We use two instances of stochastic gradient descent with smoothed hinge loss (`modified_huber`), one with L_1 regularizer, and one with L_2 regularizer.

A.1.3 Spatio-temporal Localization. We discuss here the parameters of the input data to SatScan, and the parameters that we set in the running of the SatScan's algorithm.

Input Data for SatScan. The time units we use are hourly windows, since the number of cases exhibits a diurnal pattern (as mentioned in Section 2.1). The spatial units we use are county subgroups [1] (discussed in Section 2.1). We use US Census Bureau [1] Cartographic Boundary Shapefiles to obtain latitude and longitude for each spatial unit (i.e., county subdivision). We obtain distances between spatial units with the standard Vincenty formulae in geospatial libraries.

We apply both S_a and S_t on the cases in each space-time unit, and include in the estimates of event-specific cases any case where at least one classifier has high confidence. We use a threshold of 0.7 for S_t; S_a is binary so does not need thresholds.

SatScan Analysis Parameters. As described in Section 4.3, we use Satscan's discrete Poisson model to model the case count in each spatio-temporal unit (i.e., number of cases in spatio-temporal unit comes from a Poisson distribution, according to a known underlying population at risk). We use the retrospective space-time analysis type. We use a circular window shape to identify the candidate geographical clusters for the spatial scan statistic. We allow SatScan to identify secondary clusters as well. We report only non-overlapping secondary clusters. We set the maximum spatial cluster size to be 20% of the population at risk. We set the number of Monte Carlo simulations to be 999, and we use a p-value of 0.01 as our cutoff. All

remaining parameters are set to their default values in SatScan's spatio-temporal Poisson analysis sample file.

A.1.4 Supplementary Analysis. We use a one-sided p-value of 0.01 as cutoff for the z-test. In addition, to minimize false alerts of device events, we perform the z-test only for models that account for at least 10% of the event cases. In a device event, the affected device models occur in large proportions in the event cases, so even with this additional step, we continue to discover device events, but this way, we do not output models with small deviations from their respective historical proportions.

A.2 Experimental Parameters

We have already discussed the parameters we use for LOTUS experiments in Section 5. We discuss here the parameters we use for alternate experimental approaches that we compare to LOTUS.

All alternate approaches are performed on the data in the same initial spatio-temporal window input to LOTUS (i.e., 1-2 days, 2-4 states). Among the alternate experimental approaches, vanilla co-training and vanilla SatScan use the same parameters as the co-training and SatScan components of LOTUS. The baseline keyword search has no parameters, as it simply selects matching cases from the data under analysis. We present here the parameters for the k-means clustering-based approach.

Parameters for vanilla clustering approach. As discussed in Section 5.2, we start with all the cases in the same spatio-temporal window input to LOTUS. We convert each case into a feature vector using its attributes and text with the same feature extraction module as LOTUS. We then run k-means on the feature vectors for $k = 10, 20, \ldots 100$. We pick the best k using the elbow heuristic. We pick a separate k for each event that we analyze, since some spatio-temporal windows may have higher diversity in cases than others), and use this to report our results in Figure 10 in Section 5.2.