Unveiling Clusters of Events for Alert and Incident Management in Large-Scale Enterprise IT

Derek Lin
Pivotal Software, Inc.
3495 Deer Creek Rd
Palo Alto, CA, USA
dlin@gopivotal.com

Rashmi Raghu
Pivotal Software, Inc.
3495 Deer Creek Rd
Palo Alto, CA, USA
rraghu@gopivotal.com

Vivek Ramamurthy
Pivotal Software, Inc.
3495 Deer Creek Rd
Palo Alto, CA, USA
vramamurthy@gopivotal.com

Jin Yu
Pivotal Software, Inc.
570 Bourke street
Melbourne, VIC, Australia
jyu@gopivotal.com

Regunathan Radhakrishnan
Pivotal Software, Inc.
3495 Deer Creek Rd
Palo Alto, CA, USA
rradhakrishnan1@gopivotal.com

Joseph Fernandez
Visa Inc
900 Metro Center Blvd
Foster City, CA, USA
lazferna@visa.com

ABSTRACT

Large enterprise IT (Information Technology) infrastructure components generate large volumes of alerts and incident tickets. These are manually screened, but it is otherwise difficult to extract information automatically from them to gain insights in order to improve operational efficiency. We propose a framework to cluster alerts and incident tickets based on the text in them, using unsupervised machine learning. This would be a step towards eliminating manual classification of the alerts and incidents, which is very labor intense and costly. Our framework can handle the semi-structured text in alerts generated by IT infrastructure components such as storage devices, network devices, servers etc., as well as the unstructured text in incident tickets created manually by operations support personnel. After text pre-processing and application of appropriate distance metrics, we apply different graph-theoretic approaches to cluster the alerts and incident tickets, based on their semi-structured and unstructured text respectively. For automated interpretation and read-ability on semi-structured text clusters, we propose a method to visualize clusters that preserves the structure and human-readability of the text data as compared to traditional word clouds where the text structure is not preserved; for unstructured text clusters, we find a simple way to define prototypes of clusters for easy interpretation. This framework for clustering and visualization will enable enterprises to prioritize the issues in their IT infrastructure and improve the reliability and availability of their services.

KDD'14, August 24–27, 2014, New York, NY, USA.

http://dx.doi.org/10.1145/2623330.2623360 .

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: [Data Mining]; I.5.3 [**Pattern Recognition**]: [Clustering]

Keywords

Hierarchical clustering; Connected Components; Graph cut; Complete Linkage; kd-tree; Non-Negative Matrix Factorization; Tickets Analysis; Alerts and Incidents management

1. INTRODUCTION

Organizations of all sizes struggle with a common problem in Infrastructure and Operational Management. Thousands of automated alerts with semi-structured text are generated every day from hundreds of infrastructure tools. Similarly, thousands of incident tickets (or incidents) with manually entered unstructured text are created daily by support personnel. These alerts and incidents are centrally collected (eg. IBM Netcool [5]); however, maintenance of these alerts and incidents is costly and labor intensive, and requires detailed analysis from subject matter experts. The classic problem enterprises repeatedly encounter is the identification of important alerts and incidents, on which attention can be focussed. We desire an alert and incident discovery and classification process that can be automated and is data agnostic. It benefits the downstream processing steps by helping to reduce false positive alerts [16] or improving operational efficiency to answer questions such as *What are the topics of alerts and incidents? what are the topics or clusters that generate the most volumes of alerts? What is the mean-time-to-repair for a cluster?*

Toward addressing this need, we propose two distinct frameworks: one framework to cluster alerts with semi-structured text, and another framework to cluster incidents with unstructured text. The text within alert and incident data are each preprocessed using token normalization and stop-word removal - two common approaches used in text mining. The text within each alert or incident is now represented as a bag of words.

To cluster alerts comprised of semi-structured text, we first choose a distance metric, then apply a top-down clus-

tering approach using connected components and graph-cut. Related work in [6] used Levenshtein edit distance and DBSCAN for text clustering; similarly [14] used Levenshtein distance for simple clustering based on dissimilarity. Resulting clusters are then visualized with a novel visualization technique for automated interpretation.

Compared to the clustering of alerts, clustering of incidents is more computationally demanding. Alerts are generated by technology components (a database, or an application) and are already broadly tagged by their component names. Alerts generated from a technology component are clustered independently from those from another technology component. Clustering of all alerts within a technology component is manageable, especially after the token normalization step which reduces the space of alerts substantially. Clustering of incidents, however, is more computationally challenging as there is no high-level grouping of incidents. To cluster incidents, we develop an approach based on matrix factorization and KD-tree to create a preliminary grouping of incidents before applying complete linkage clustering per group and later performing cluster merging in a post-processing step.

Based on the clustering output, we compute several interesting business intelligence metrics or key performance indicators per cluster and visualize them. In this paper, we also address the need for automated interpretation of alert clusters. For alert cluster interpretation, we propose a novel visualization that preserves the structure and human-readability of the text data as compared to traditional word clouds where the text structure is not preserved. For incident cluster interpretation, we simply find prototypes of clusters. KPI visualization and cluster interpretation would help enterprises prioritize the issues in their IT infrastructure and improve the reliability and availability of their services.

The rest of the paper is organized as follows. In the following two sections, we will describe the details of the framework. Then, we present the results of these clustering approaches that provide insights into the top issues in the IT infrastructure.

2. HIERARCHICAL CLUSTERING OF SEMI-STRUCTURED ALERTS TEXT BASED ON CONNECTED COMPONENTS DETECTION AND GRAPH-CUT

Figure 1 illustrates the 5 high level steps in the proposed framework to cluster semi-structured alerts text.

- Step 1: Token normalization of alert and incident summary texts.
- Step 2: Stopword removal.
- Step 3: Pairwise distance metric computation to compare any two summaries.
- Step 4: Hierarchical clustering based on computed pairwise distance matrix.
- Step 5: Post-processing and Visualization.

In the following sub-sections we will describe these steps in detail.

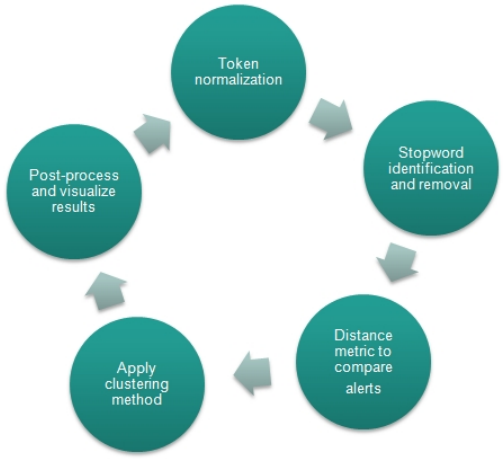

Figure 1: Steps in the clustering framework

2.1 Token Normalization of Alerts Text

Tokens are individual words within alerts. For the purpose of clustering, we are only interested in the textual structural information in alerts. Tokens exhibiting high variability such as IP address, date and time are either removed or replaced by another common string. For instance, two alerts A and B may report the same issue at specific IP address (e.g Alert A: service is not running at 10.2.34.56 and Alert B: service is not running at 10.12.2.54). For the purpose of clustering these two alerts based on the issue they refer to (a particular service not running), the specific IP address (e.g 10.12.2.54) in the alert string is not important. By replacing the IP address with a string "*ipaddr*", both of these alerts end up containing the same 'normalized' text. Which particular tokens to normalize, depends on the problem domain. In our enterprise data set, we performed the following token normalizations to bring together alerts that originally appeared to be very different.

- Replace various date formats with the text "date"
- Replace various time formats with the text "time"
- Replace various URLs with the text "url"
- Replace various file paths with the text "filepath"
- Replace various server (node) names with "nodename"
- Remove application names that provided no insight on the type of an alert
- Replace numbers with ♯, and punctuations with whitespace
- Remove trailing whitespace

Token normalization substantially reduced the alert space and eliminated a lot of noise in the data. It was now a lot easier to detect structural patterns within the normalized alerts and to apply suitable distance metrics for clustering.

2.2 Stopword Removal

Stopwords are words that commonly occur in all of the alert groups. These words do not help in clustering alerts as they are present in a large proportion of the alerts. For instance, every alert originating from a certain service component contains the word: "service". In order to cluster alerts coming out of this component based on its content, this word ("service") is not helpful and hence can be considered as a stopword and removed.

In order to identify the stopwords, we perform the following steps:

- Split the alerts text into unigrams (words or tokens)
- For each distinct unigram, compute its total count across all alerts, as well as the total number of distinct token normalized alerts that it appears in.
- Sort the unigrams in descending order of total count across all alerts, and then in descending order of total number of distinct token normalized alerts that they appear in. Choose the top "k" unigrams from this sorted list as the stopwords to be removed. "k", is chosen by visual inspection, using domain knowledge, to remove only words that are not useful for clustering.

2.3 Distance Metric to Compare Alerts

Before we can perform any clustering, we need to be able to compare any two alerts where each alert is represented as a bag of words . We propose to use Jaccard distance as a metric to compare any two alerts. In order to compute Jaccard distance, each alert is represented as a bag of words. Let us denote one such set from alert A as A and another set of words from alert B as B. Then, Jaccard distance is defined as below:

$$JaccardDistance(A, B) = 1 - (\frac{|A \cap B|}{|A \cup B|}) \quad (1)$$

Jaccard distance is a number between 0 and 1 and is not sensitive to the position of the matching word in the two alerts that are being compared.

Given N alerts that need to be clustered, we create at NxN distance matrix using the Jaccard distance metric to compare any two alerts.

It is possible to apply other distance metrics such as Rank-Biased Overlap [17], which takes into account the position of the match when comparing two alerts.

2.4 Clustering Alerts based on Distance Matrix

In the previous section, we described how we create a NxN distance matrix that summarizes how the alerts relate to each other. Given the NxN distance matrix, various clustering methods potentially apply, but each with different assumptions and advantages. For example, hierarchical clustering with minimax linkage [1] tended to produce over-fragmented clusters in our experiments by assuming that similar events within a cluster are within a certain radius distance from the centroid. Not assuming any cluster shape is a criterion in our clustering method choice. We propose using a graph-theoretic approach that uses connected component detection to generate initial clusters before applying the graph-cut algorithm to further refine the clusters.

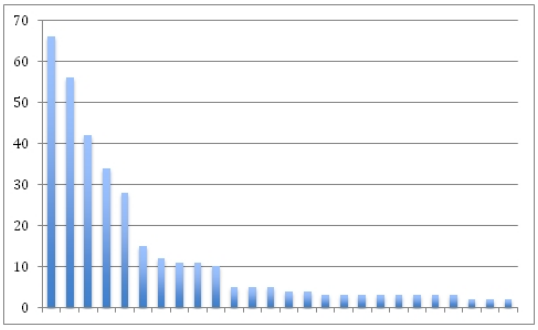

Figure 2: Size of Connected Components from a Graph (Graph A)

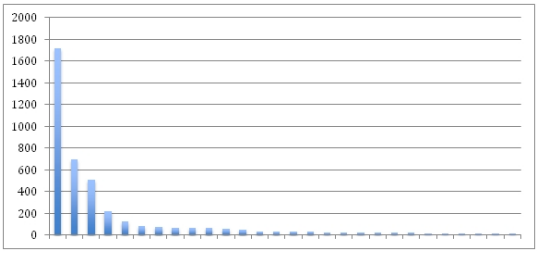

Figure 3: Size of Connected Components from a Graph (Graph B)

2.4.1 *Top-down Clustering based on Connected Components Detection*

Given the NxN distance matrix, we can create a graph where the vertices of the graph correspond to alerts. We can establish an edge between any two vertices (alerts) if the Jaccard distance between the two alerts is less than a threshold (e.g 0.5). After creating this graph, we run a connected components detection algorithm on it to identify parts of the graph that are not connected to other parts.

A connected component [4] is a sub-graph in which there exists at least one path that connects any two vertices, and which is connected to no additional vertices in the complete graph. One advantage of using connected components is that it doesn't assume any cluster shape.

Figure 2 below shows the number of connected components detected from a graph (say Graph A) representing few hundred alerts. There are 5 to 6 significant clusters or connected components detected from this group of alerts.

Similarly, we detected 196 connected components from the another graph (say Graph B) representing few thousand alerts. The size of top 3 or 4 connected components in the case of Graph B tended to be much larger than in the case of Graph A. The sizes of detected connected components from Graph B are shown below in Figure 3.

For instance, the top connected component in Graph B was of size 1717. It was desirable to break this large connected component (cluster) into smaller clusters. Toward this end, we employed normalized cut, which is a hierarchical graph partitioning algorithm. In the following subsection, we describe the details of this graph-cut algorithm.

2.4.2 *Graph Cut*

Given a NxN distance matrix (C) of a connected component, we perform the following steps to derive smaller

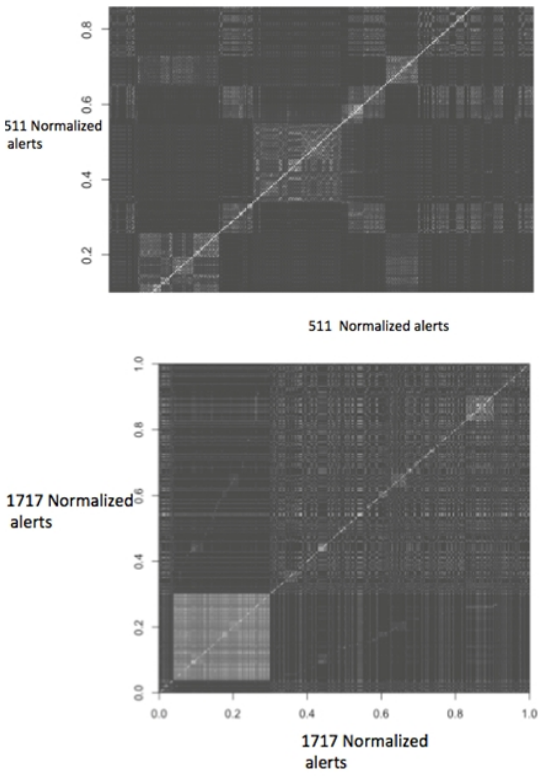

Figure 4: Affinity matrices of large connected components

clusters from this connected component:

- Compute an affinity matrix W of the same size as distance matrix where $W = exp^{(-\frac{C}{\sigma^2})}$. Here σ is a parameter that controls how quickly affinity (similarity) falls off with distance [7]. σ^2 was empirically chosen to be 0.5.
- Compute a diagonal degree matrix D whose diagonal is $d(i,i) = \sum_j W(i,j)$
- Solve the following eigen system: $D^{-1/2}(D-W)D^{-1/2}x = x$
- Here x is the cluster indicator vector that partitions the input graph into two parts A and B such that within-cluster affinity is maximized while between cluster-affinity is minimized.
- Compute cluster radii for the new partitions A and B and their sizes.
- If $clusterradius > radiusthreshold$ and $clustersize > sizethreshold$, then continue partitioning either A or B recursively.

Figure 4 below shows affinity matrices of two such large connected components (clusters). Note that the block structure along the main diagonal of the affinity matrices, indicating the presence smaller clusters within the connected component.

After recursive partitioning of Graph B, it is broken down to 4 significant smaller clusters as shown in Figure 5.The

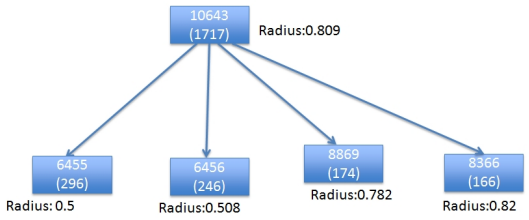

Figure 5: Graph cut based recursive partitioning of Graph B into 4 smaller clusters

connected component of size 1717 is partitioned into 4 clusters of sizes 296,246,174 and 166 respectively and other smaller clusters. Note the cluster radii for the smaller clusters from the graph-cut partitions are smaller than the radius of the large connected component.

If the largest connected component is greater than say 10000 alerts, then computing the full distance matrix and performing Graph cut may not scale. We propose to use the Nystrom method to perform graph cut, in which you randomly sample *say*5% of the distance matrix and compute an approximation to the dense solution of the eigen system using the Nystrom approximation as suggested in [2]

3. HIERARCHICAL CLUSTERING OF UNSTRUCTURED INCIDENT TICKETS

To account for the complexity arising from the unstructured nature of the incident tickets, we extend the framework for clustering semi-structured Netcool alerts by incorporating two new elements: non-negative matrix factorization and KD-tree data structure. The resulting framework consists of the following high-level steps:

1. Token normalization and stopword removal.
2. High-level grouping of tickets based on non-negative matrix factorization and KD-tree.
3. Hierarchical clustering within each group of incidents.
4. A refinement step to merge similar clusters.

3.1 Token Normalization and Stopword Removal

We take the same token normalization approach as applied to Netcool alerts except that timestamps, file paths, URLs, punctuations, and digits are removed from incident summaries, instead of being replaced by common strings. This is based on the observation that unlike semi-structured alerts, ticket descriptions generally do not conform to certain formats, hence semantically non-meaningful strings like timestamps and file paths are unlikely to contribute to the formation of clusters.

In addition, we also remove common English stopwords such as “the” and “is” as well as a custom list of words such as “please” and “following” to further clean the data.

3.2 High-Level Grouping of Tickets

For Netcool data, clustering is performed within each subgroup of alerts. However, sub-group information is not available for incident tickets. This presents a potential computational challenge, due to the need to compute pairwise dis-

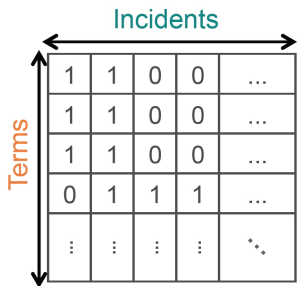

Figure 6: A binary term-incident matrix.

tances between each pair of tickets. In what follows we introduce a novel approach to the creation of high-level grouping of incident tickets. Underlying the approach are two techniques: non-negative matrix factorization and KD-tree.

3.2.1 Non-Negative Matrix Factorization

As the first step towards generating a preliminary grouping of incident tickets, we construct feature vectors to represent tickets. Using bag-of-words representation, we collect the entire set of tickets in a binary matrix. Each row represents a term (word), and each column represents an incident ticket, with the value 1 indicating the presence of a term; and 0 otherwise. As shown in Figure 6, the term-incident matrix is highly sparse. This calls for a dimensionality reduction step to obtain a more compact representation of the data. Towards this end, we factorize the term-incident matrix via non-negative matrix factorization [9, 12].

Non-negative matrix factorization (NMF) is a popular technique for dimensionality reduction and underlies a wide range of machine learning applications such as text mining [13, 15, 18], image analysis [10], recommender systems [19], etc. It refers to the factorization of a non-negative matrix $A_{m\times n}$ into the product of two low-rank matrices $W_{m\times k}$ and $H_{k\times n}$, whose elements are also non-negative. The rank parameter k is typically chosen to be much smaller than $\min(m, n)$. Given the input matrix A and the rank k, NMF can be formulated as the following constrained optimization problem:

$$\min_{W,H} \parallel A - WH \parallel_F^2, \quad \text{s.t.} \quad W, H \geq 0, \tag{2}$$

where $\parallel \cdot \parallel_F$ denotes Frobenius norm, and the constraints: $W, H \geq 0$, enforce W and H to take only non-negative elements. This problem is non-convex in both W and H, but becomes a convex problem when either W or H is fixed.

There are many different approaches to solving (2). Alternating Least Squares (ALS) [12] is among the most popular ones, due to its simplicity and good practical performance; see Alg. 1 for an outline of the algorithm. Starting with an initial estimation of W, ALS first solves the least squares problem $\min_H \parallel A - WH \parallel_F^2$ for H, which is equivalent to solving the matrix equation at Step 5 of Alg. 1. The subsequent Step 6 is a truncation step to ensure non-negativity of the solution. It is clear that the closed-form solution to the matrix equation at Step 5 is $H = (W^\top W)^{-1} W^\top A$. Although matrix inversion is computationally expensive in general, here the size of the positive-semidefinite matrix $(W^\top W)$ is only $k \times k$. Hence, the cost of inverting this small matrix is almost negligible. The rest of the operation to obtain H involves sparse matrix multiplication, which can be

Algorithm 1 Alternating Least Squares for NMF

1: **input** a matrix $A_{m\times n} \geq 0$ and an integer $0 < k \ll \min(m, n)$.
2: **output** two low-rank factors: $W_{m\times k} \geq 0$ and $H_{k\times n} \geq 0$.
3: Initialize W, e.g. using methods introduced in [8].
4: **repeat**
5: fix W, solve $W^\top WH = W^\top A$ for H.
6: set all negative elements in H to 0.
7: fix H, solve $HH^\top W^\top = HA^\top$ for W.
8: set all negative elements in W to 0.
9: **until** a convergence criterion is met.
10: **optional** standardize the solution: normalize each column of W (resp. each row of H) to have norm 1, adjust H (resp. W) accordingly to absorb scaling coefficients.

efficiently implemented in an MPP database [3]. Similarly, given the current H, W is updated via a two-step procedure (Steps 7 and 8). The algorithm proceeds in an alternating fashion until a convergence criterion is met. In our implementation, we also standardize the solution as shown at Step 10 of Alg. 1 by normalizing H.

A known limitation of ALS is that its convergence is difficult to analyze, due to the ad-hoc enforcement of the non-negativity constraints. Nevertheless, ALS and many of its variants (see e.g. [19]) have seen great success in real-world applications. In practical implementations, it is a common strategy to monitor changes in the elements of W and H, and stop the algorithm when the changes fall below a prespecified threshold, or the maximum number of allowed iterations is reached. Another practical aspect of ALS is initialization. Due to the non-convex nature of the NMF problem, ALS (and in fact any NMF algorithm) is bound to be sensitive to the initial estimate of W. We use an initialization strategy suggested in [8], which is to initialize each column of W by averaging a certain number of random columns in A. This simple and intuitive approach has been shown [8] to strike a good balance between performance and computational cost, and worked well in our experiments.

The resulting low-rank factors effectively encode the original matrix A (term-incident matrix) in a k-dimensional latent space. The columns of W are the basis vectors of the latent space. Let $\mathbf{a}_i$ be a column (an incident ticket) in A, and $\mathbf{h}_i$ the corresponding column in H. Then, an $\mathbf{a}_i$ can be approximated by $\mathbf{a}_i \approx W\,\mathbf{h}_i$, meaning that the approximation is an additive linear combination of the basis vectors, weighted by the entries in $\mathbf{h}_i$. Essentially, an $\mathbf{h}_i$ specifies how an incident ticket is expressed in the latent space, where tickets that share similar descriptions still remain "close".

3.2.2 A KD-Tree of Incident Tickets

Given the low-rank factor H, we organize its columns in a KD-tree [11], a commonly used data structure for searching nearest "neighbors" in a multi-dimensional space. KD-tree partitions multi-dimensional data into non-overlapping subsets (also called subtrees) such that nearest neighbors are more likely to be within rather than outside of a subtree. A KD-tree is constructed in a recursive manner (see Fig. 7):

1. Pick a dimension.
2. Find the median along that dimension.
3. Split the data at the median.

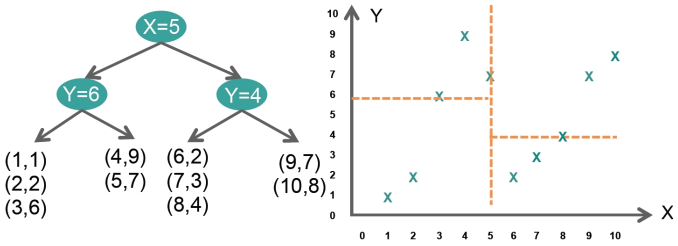

Figure 7: An example KD-tree.

4. Repeat on subtrees until they reach a desired size.

As illustrated in Fig. 7, a KD-tree creates a natural grouping of the data. This motivates our use of KD-tree to produce a high-level grouping of incident tickets

Note that at each iteration KD-tree splits the data along only one dimension. The implication is that the order of dimensions to partition the data can influence the final grouping. Our implementation gives priority to dimensions that contain more non-zero values. The assumption is that denser dimensions are more discriminative. It is also well known that "the curse of dimensionality" is a problem of KD-tree. Data points in a high-dimensional space tend to be far away from each other, hence are unlikely to form large groups. Recall that our KD-tree is constructed in the low-dimensional latent space produced by NMF. This effectively avoids the curse of dimensionality issue. Empirically, we found it sufficient to set the dimensionality (i.e., the rank parameter in NMF) to 15 to obtain a satisfactory grouping result.

3.3 Hierarchical Clustering

Having constructed a KD-tree, agglomerative hierarchical clustering is run within each subtree to obtain a refined grouping of incident tickets. Agglomerative hierarchical clustering builds a clustering tree from bottom up by merging closest clusters. In contrast to the familiar K-means clustering method, hierarchical clustering does not require the number of clusters to be specified *a priori*. Instead, a user decides at what height to cut a clustering tree, which indirectly determines the number of clusters. We find that it is often much easier to empirically find an appropriate height than the number of clusters.

Hierarchical clustering operates on pairwise distances. To measure pairwise distances between ticket descriptions, we design a distance measure as below:

$$dist(A, B) = \max_i |X_i| - |A \cap B|, \tag{3}$$

where X_i, A, and B are sets of distinct words in a ticket description, and $|\cdot|$ denotes the cardinality of a set. The first term: $\max_i |X_i|$ is a constant. It is introduced to ensure positivity of the distance measure. Note that this distance measure is based solely on the intersection of texts but not length. This is motivated by the observation that descriptions of the same issue can vary widely in length; for instance, the two examples shown in Figure 8 both describe a password related issue, but one is brief, while the other contains much more details.

For clustering, we used the complete-linkage method. It is an agglomerative hierarchical clustering method that defines the distance between two clusters by the farthest pair of data

"SQL Password Reset"

"My account password did not work on the following servers which prevented me from resetting the root password. Once my personal account password issue is resolved, I will reset the root account passwords."

Figure 8: Tickets of the same issue can vary widely in length.

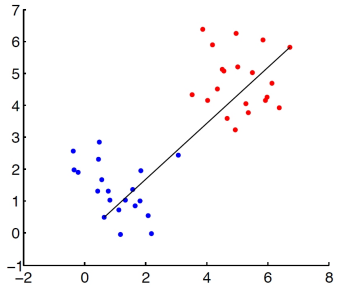

Figure 9: Complete linkage defines the distance between two clusters by the farthest pair of data points.

points (Figure 9). Combined with the intersection-based distance measure (3), complete-linkage clustering ensures that each pair of tickets in a cluster share at least some common words. The resulting clustering tree can be cut at a height to ensure a certain degree of overlap between each pair of tickets within a cluster. Suppose that the maximum length of a ticket description is 30 (i.e., $\max_i |X_i| = 30$), then setting the height to 28 guarantees an overlap of at least 2 words between any two tickets of the same cluster. Moreover, an exemplar is selected for each cluster to show the typical "look and feel" of tickets in a cluster. It is defined as the ticket that has the shortest distance to its farthest "neighbor" (Figure 10):

$$\mathbf{x}^* := \arg\min_{\mathbf{x}\in X} \max_{\mathbf{x}'\in X} d(\mathbf{x}, \mathbf{x}'). \tag{4}$$

3.4 Merging Similar Clusters

Due to the variability of ticket descriptions, the same type

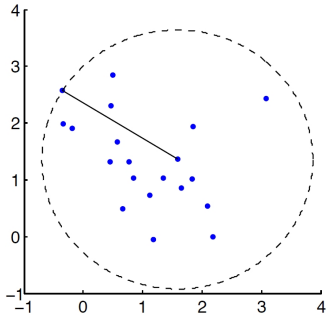

Figure 10: The exemplar (center of the encompassing circle) has the shortest distance to its farthest neighbor.

of tickets can be assigned to different subtrees, or to different clusters even within the same subtree. A refinement step was therefore developed to merge those similar clusters.

First, frequently used words (e.g. top-10 words) are extracted from each cluster to form a feature vector. The intersection-based distance measure (3) is then applied to feature vectors to quantify the dissimilarity between two clusters. Given pairwise distances of clusters, the complete-linkage clustering method is used again to create groups of clusters. For each merged cluster, a representative incident is selected as the exemplar (Figure 10). In our experiments this refinement procedure joins about 66% of clusters produced by the initial run of clustering (as described in Sec. 3.3).

4. EXPERIMENTAL RESULTS

4.1 Dataset and Compute set up

We had collected data from a large enterprise's IT infrastructure that supports several software services that are up all the time over a period of 3 months. In all, we had 5 million rows of alerts over a period of 3 months. Some of these alerts text were identical as they repeated at a certain time interval (say 1 min). Therefore, by focusing our attention only on those unique alerts we were able to reduce the 5 Million rows to about $590,000$. After token normalization and stopword removal, the number of unique alerts came down to 22000 from the original count of 590000. Given these 22000 unique alerts and labels for 4 sub-groups within this set, we wanted to find clusters within 4 sub-groups. The 4 sub-groups are already known to the enterprise and hence we wanted to find clusters within each sub-group.

- Sub-Group 1 with 1408 alerts
- Sub-Group 2 with 4813 alerts
- Sub-Group 3 with 440 alerts
- Sub-Group 4 with 5388 alerts

We loaded all the data onto Pivotal's Greenplum Database (GPDB) running on a Data Computing Appliance (DCA) and implemented the clustering frameworks described above using a combination of psql scripts, MADlib (an opensource big data machine learning toolkit that runs on GPDB), PL/R code and PL/python code. GPDB has the MPP (Massively Parallel Processing) architecture with one master and several segment hosts sharing the computational load and is built for scalable big data analytics.

4.2 Clustering Results

Figure 11 presents a visualization of the clusters from each of 4 Sub-Groups of alerts. The size of the bubbles in the Figure 11 corresponds to the number of alerts belonging to a particular cluster. The intensity of the coloring of the bubbles corresponds to the number of incident tickets created from the alerts of a particular cluster. For instance, the darker the green color in the clusters corresponding to Sub-Group 4, the higher the number of incidents that got created from that cluster of alerts. Furthermore, the sizes of clusters in the 4 alert groups tend to have a long-tail distribution. That is, there are few big to medium sized clusters whereas there are a lot smaller clusters in each of the alert groups.

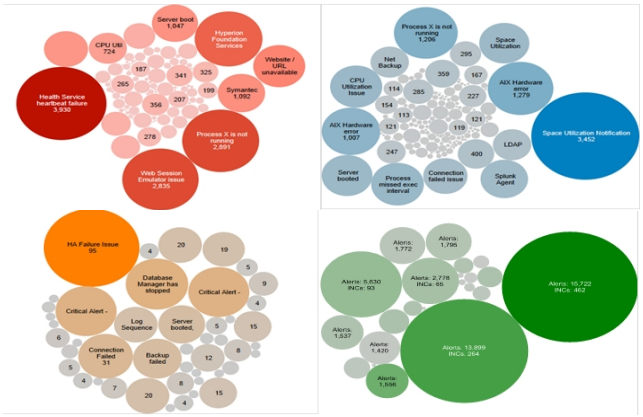

Figure 11: Visualization of clusters with the 4 Sub-Groups of Alerts

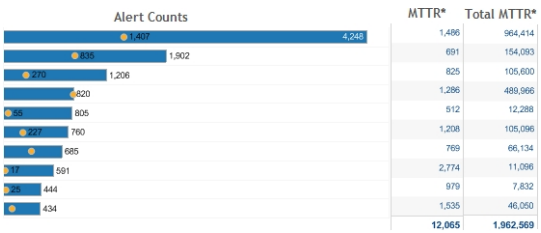

Figure 12: Key Performance Indicator (MTTR) Computed for top k clusters

In order for IT operations to focus on their top critical issues, especially events and incidents that generate major failures, it is sufficient to look at a few big-to-medium sized clusters in each of the alert groups. Furthermore, for each of incidents created we can compute a quantity called time-to-resolve which is the time taken for that particular issue to be resolved. For each of the clusters, one can then look at the mean-time-to-resolve (MTTR) from the incidents that got created from that cluster.

Figure 12 below summarizes this MTTR measure in terms of minutes for each of the top clusters across the 4 Sub-Groups. It also indicates how many of the alerts became incidents in each cluster. For instance, the first cluster has 4248 alerts of which only 1407 became incidents. This implies that there is room for improvement by paying attention to the rest of alerts before they become bigger issues.

4.3 Visualizing Clusters of Alerts

It is challenging to visualize clustered or grouped semi-structured text data such as alerts. Techniques such as word clouds are often used to visualize text data. While word clouds can show the relative frequency of different words across all alerts in a cluster they do not reveal any structure that is common across alerts.

Consider the hypothetical example of alerts shown in Table 1. Word clouds would only pick out the fact that the words 'Server', 'not'and 'responding'occur very often in the set of alerts shown but would not be able to pick out the order in which those words occur. For automated interpretation and readability of semi-structured text clusters or groups, we propose a method to visualize clusters that preserves the structure and human-readability of the text data

as compared to traditional word clouds where the text structure is not preserved. This is especially necessary for large clusters and/or longer sequences of text. The proposed visualization method is also applicable to other types of semi-structured clustered/grouped text or categorical data.

Table 1: Example of semi-structured text data

Alert #	Text
1	Server A not responding
2	Server B is not responding
3	Server C not responding
4	Server D not responding

The steps in the visualization method are as follows:

1. For each alert in a given cluster generate (word, position) tuples, comprising the word and its position in the alert. Depending on whether the need is to visualize word lined up in order from left-to-right or right-to-left the text can be 'left-'or 'right-justified'prior to generating tuples.

2. For each cluster record all (word, position) tuples and the frequency with which they occur. Table 2 shows this for the example data in Table 1 with word positions in left-to-right order.

3. Visualize the frequency of (word, position) tuples from Step 2 above as shown in Figure 13. The vertical axis in the figure represents the position of the word in the alerts and the horizontal axis represents the frequency/percent occurrence of the (word, position) tuple. Each circle in the figure represents a word. Reading the words that occur most frequently in order of their positions we see the following common structure in the example: 'Server — not responding'.

Table 2: Occurrence of (word, position) tuples

Word	Position	Count	Fraction
Server	0	4	1
A	1	1	0.25
B	1	1	0.25
C	1	1	0.25
D	1	1	0.25
not	2	3	0.75
is	2	1	0.25
responding	3	2	0.75
not	3	1	0.25
responding	4	1	0.25

Figure 14 shows an example visualization of three clusters generated from real world semi-structured alerts data discussed in the previous sections. Reading the alerts with high percent occurrence off the figure we see the following elements are common within each respective cluster:

1. 'ora#: exception encountered: core dump — address not mapped to object '

2. 'time gmt positive alert harmless '

3. 'aix hardware error: '

The automatic readability of the most common theme in each cluster that this type of visualization provides is key to quick interpretation of the results.

4.4 Clustering Results of Unstructured Incident Tickets

For unstructured incident tickets, we focus on discovering the "topics" of top clusters. We find that top-25 clusters already accounts for 20% of the total tickets. (There are around 67×10^3 tickets in total.) Figure 15 provides some examples of top clusters. The keywords of each cluster are the words shared by all the ticket descriptions in a cluster, while the exemplar provides a more detailed view into what a typical ticket looks like.

5. CONCLUSIONS

This paper presents a framework to automatically cluster alerts with semi-structured text and incidents with unstructured text, generated from large enterprise IT infrastructure. Such clustering information allows IT operation staff to gain insights on issues they face everyday, based on the top clusters of alerts and incidents, allowing them to prioritize areas for problem-solving. We proposed two distinct frameworks: one framework to cluster alerts with semi-structured text, and another framework to cluster incidents with unstructured text. The text within alert and incident data are each preprocessed using token normalization and stop-word removal. The text within each alert or incident is then represented as a bag of words. To cluster alerts comprised of semi-structured text, we first choose a distance metric, then apply a top-down clustering approach using connected components and graph-cut. To cluster incidents, we proposed an approach based on matrix factorization and KD-tree to create a preliminary grouping of incidents before applying complete linkage clustering per group and later performing cluster merging in a post-processing step.

Based on the clustering output, we computed several key performance indicators per cluster such as the MTTR (Mean-Time-To-Resolve) and visualize them. For alert cluster interpretation, we propose a novel visualization that preserves the structure and human-readability of the text data as compared to traditional word clouds where the text structure is not preserved. One possible future research direction would involve incorporation of temporal information.

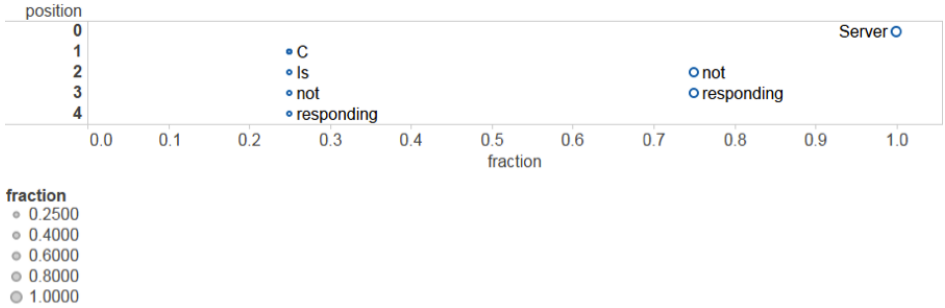

Figure 13: Visualization of the example presented in Tables 1 and 2. The 'Server __ not responding 'structure in the alerts is clearly seen

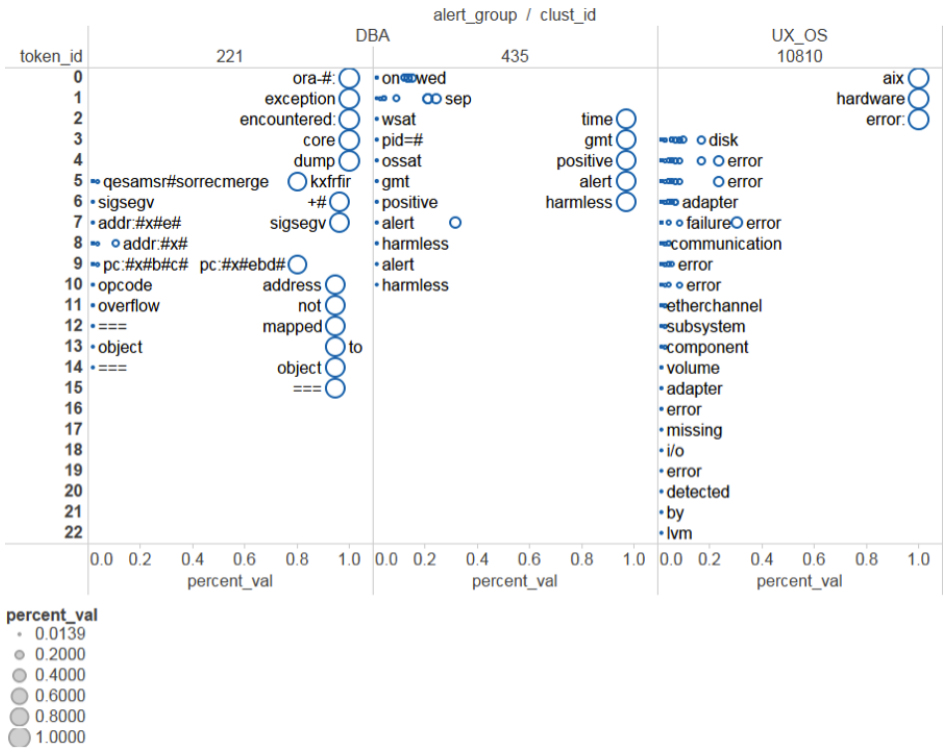

Figure 14: Visualization of clustered semi-structured alerts text

ID	Size	Cumulative Percent	keywords	Exemplar
1	2400	3.59%	ended,notok	ERR1043E DE04 3 Q622A551 134 ENDED NOTOK 20130721 23:06:29 1064 MVS
2	1892	6.42%	alert,splunk	sdesku1-4-linux-server-splunk alert: The fileextract-primary process is not running.
3	925	7.81%	percent,utilization	sdesku2-3-aix-server-/var is at 89.93 percent utilization
4	758	8.94%	booted,ensure,running	sdesku1-4-win-server-SCOM:Server booted at 2013-09-27T21:59:41.0000000 00:00 - Please ensure critical applications are running.
5	567	9.79%	provisioning,vblock	Automated Vblock server provisioning request for linux-server [APP Files Processing System]

Figure 15: Top-5 clusters of incident tickets.

6. REFERENCES

[1] J. Bien and R. Tibshirani. Hierarchical clustering with prototypes via minimax linkage. *Journal of the American Statistical Association*, 106(495):1075–1084, 2011.

[2] S. F. C.Fowlkes and J.Malik. Spectral grouping using the nystrom method. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 26, February 2004.

[3] J. Cohen, B. Dolan, M. Dunlap, J. M. Hellerstein, and C. Welton. Mad skills: new analysis practices for big data. *Proceedings of the VLDB Endowment*, 2(2):1481–1492, 2009.

[4] P. ERDdS and A. R&WI. On random graphs i. *Publ. Math. Debrecen*, 6:290–297, 1959.

[5] IBM Netcool. http://tinyurl.com/m5v8nh2.

[6] S. Jain, I. Singh, A. Chandra, Z.-L. Zhang, and G. Bronevetsky. Extracting the textual and temporal structure of supercomputing logs. In *High Performance Computing (HiPC), 2009 International Conference on*, pages 254–263. IEEE, 2009.

[7] J.Shi and J.Malik. Normalized cuts and image segmentation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22, August 2000.

[8] A. N. Langville, C. D. Meyer, R. Albright, J. Cox, and D. Duling. Initializations for the nonnegative matrix factorization. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 23–26, 2006.

[9] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

[10] S. Z. Li, X. Hou, H. Zhang, and Q. Cheng. Learning spatially localized, parts-based representation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2001.

[11] A. W. Moore. An introductory tutorial on KD-trees, 1991. http://tinyurl.com/cja9o9.

[12] P. Paatero and U. Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5(2):111–126, 1994.

[13] V. P. Pauca, F. Shahnaz, M. W. Berry, and R. J. Plemmons. Text mining using non-negative matrix factorizations. In *Proceedings of SIAM International Conference on Data Mining*, 2004.

[14] F. Salfner and S. Tschirpke. Error log processing for accurate failure prediction. In *WASL*, 2008.

[15] F. Shahnaz, M. W. Berry, V. P. Pauca, and R. J. Plemmons. Document clustering using nonnegative matrix factorization. *Information Processing & Management*, 42(2):373–386, 2006.

[16] L. Tang, T. Li, L. Shwartz, F. Pinel, and G. Y. Grabarnik. An integrated framework for optimizing automatic monitoring systems in large it infrastructures. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1249–1257. ACM, 2013.

[17] W. Webber, A. Moffat, and J. Zobel. A similarity measure for indefinite rankings. *ACM Transactions on Information Systems (TOIS)*, 28(4):20, 2010.

[18] W. Xu, X. Liu, and Y. Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of ACM SIGIR conference on Research and Development in Information Retrieval*, pages 267–273, 2003.

[19] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-scale parallel collaborative filtering for the Netflix prize. In *Algorithmic Aspects in Information and Management*, pages 337–348. Springer, 2008.