A Statistical Machine Learning Approach for Ticket Mining in IT Service Delivery

Ea-Ee Jan, Jian Ni, Niyu Ge, Naga Ayachitula, Xiaolan Zhang
IBM T.J. Watson Research Center, 19 Skyline Dr, Hawthorne, NY
{ejan|nij|niyuge|nagaaka|cxzhang}@us.ibm.com

***Abstract*—Ticketing is a fundamental management process of IT service delivery. Customers typically express their requests in the form of tickets related to problems or configuration changes of existing systems. Tickets contain a wealth of information which, when connected with other sources of information such as asset and configuration information, monitoring information, can yield new insights that would otherwise be impossible to gain from one isolated source. Linking these various sources of information requires a common *key* shared by these data sources. The *key* is the server names. Unfortunately, due to historical as well as practical reasons, the server names are not always present in the tickets as a standalone field. Rather, they are embedded in unstructured text fields such as abstract and descriptions. Thus, automatically identifying server names in tickets is a crucial step in linking various information sources. In this paper, we present a statistical machine learning method called Conditional Random Field (CRF) that can automatically identify server names in tickets with high accuracy and robustness. We then illustrate how such linkages can be leveraged to create new business insights.**

Keywords: Service management, Data model, Analytics, Statistical Models, Machine Learning

I. INTRODUCTION

The success of IT service delivery heavily depends on making strategic decisions based on insights from multiple business components. These decisions require a comprehensive examination of several IT service management processes, such as *incident management*, *configuration management*, and *service request management* [7,9,10,11]. Forming such an encompassing view is a complex process. First, it requires a unification of various tools and data models that are produced by different business units independently. Second, it needs to take into account of the human factors which may result in missing or inconsistent content. Third, as the service delivery business evolves, new problems arise that may demand previously unexplored concepts or features to be derived from the operations content. Thus, existing data models cannot be used and the new knowledge must be inferred from available content.

Recent research has focused on building semi-automated knowledge discovery tools that are capable of locating information across a variety of business units and data models [9,12]. For example, data center operational issues, such as outages and IT system incidents, are recorded and tracked as tickets in IPC (Incident, Problem, Change) management systems. In another department, the data center asset management, a different set of system management tools, keeps track of the available servers in the data center and the hardware and software installed on each server. By linking the tickets and the server configuration details, one can learn about the top server features that lead to the creation of the tickets, and can drive targeted actions for productivity improvement. Also, one can learn how groups of servers with different hardware and software compare with respect to their ticket handling labor costs and equipment costs.

In some IT environments, legacy IPC and asset management systems are not integrated to provide a structured ticket field that captures an accurate reference to related servers. In other instances, although dedicated fields are available, the information might be missing or inaccurate because system administrators may have forgotten to capture it appropriately as they rush to close the ticket and meet productivity targets. Therefore, in a business-knowledge discovery framework that aspires to gain cross-business unit insights, text-mining on the ticket descriptions is essential in linking ticketing records and server configuration records. The tickets and the server configuration details are linked by the common *server names*. Therefore, finding server names in the tickets is an essential first step toward bridging the gap. Although ideally the ticketing application should provide users with a structured interface with a required 'server name' field, there are also many applications that allow users to enter free text. In this paper we focus on such free-text tickets. We present a statistical machine learning method that can identify server names in unstructured tickets text data accurately and robustly.

II. PREVIOUS WORK

Tickets comprise extensive details about the service requests and problem symptoms in unstructured fields such as the abstract, description, and resolution. Previous work [9,12] on finding server names in tickets has been largely heuristic-based. Before processing the tickets, a dictionary of server references is first extracted from the server configuration database. Because the tickets consist mostly of *free* texts, well-formatted texts cannot be assumed. In fact, quote, parenthesis, and other special characters are frequently used around server information. Space delimiters are sometimes missing; prefixes and suffixes are added due to various system operations. In order to build heuristic rules, a set of tickets is set aside for rules training and creation as followed. Each ticket is tokenized, and the tokens are matched against the dictionary using fuzzy matching algorithms. The prefix and suffix matches between each token and dictionary entries are collected. Histograms are created for each prefix and suffix patterns. The patterns with highest frequency counts are used as

978-3-901882-50-0

heuristic rules. The regular expression pattern substitutions are then used for text processing over the entire domain of tickets.

There are two major drawbacks of this approach. First, as with many heuristic-based approaches, the rules are often noisy and incomplete. By taking only the frequent ones, the system may suffer severe recall problems for failing to find those that are not following the rules. On the other hand, if all the rules are taken, the system may suffer precision problems for finding false server names. The second problem is caused by the fact that the rules are largely context independent. When a common word, for example, "*discovery*" is used for the server name, without context, a simple matching algorithm will lead to many false server detections.

In this paper, we present a statistical machine learning approach to discover server names. The model is sophisticated enough not only to identify server names robustly and precisely, but also discover server names that are absent in the dictionary.

III. Problem formulation

A ticket consists of a sequence of *words* (also called *tokens*, and in this paper we will use words and tokens interchangeably). Each word in a ticket is either a server name or not. The problem of finding server names in a ticket can be cast as a *sequential labeling* problem. For each word *w* in the ticket, we want to assign a *label* $l \in \{0,1\}$ to it indicating whether or not the word is a server name: with 0 indicating the current word is not a server name and 1 indicating that it is.

In this paper, we apply a *statistical machine learning* approach to address the ticket mining problem. Let

$$\boldsymbol{W} = (W_1, W_2, \dots, W_n)$$

be a random vector representing the sequence of words in a ticket (with *n* words), and let

$$\boldsymbol{L} = (L_1, L_2, \dots, L_n)$$

be a random vector representing the corresponding labels of the words.

Given a limited set of labeled training data, possibly with noises, we want to learn a *probabilistic model* that can describe the conditional distribution of $P(\boldsymbol{L}|\boldsymbol{W})$. Based on the probabilistic model, for any new ticket $\boldsymbol{w}$, we want to infer the most likely labels $\boldsymbol{l}^*$ for the words in the ticket:

$$\boldsymbol{l}^* = \text{argmax}_{\boldsymbol{l}} P(\boldsymbol{L} = \boldsymbol{l}|\boldsymbol{W} = \boldsymbol{w}) \quad (1)$$

Given the inferred label vector $\boldsymbol{l}^*$, we can predict whether the ticket contains a server name or not.

IV. A Condtional Random field approach

A. The Probablistic Model

Conditional random fields (CRFs) are a class of *probabilistic graphical models* that can be used to label or parse sequential data [8]. For our ticket mining problem, $\boldsymbol{W}$ is a random vector representing the sequence of words in a ticket, and $\boldsymbol{L}$ is a random vector representing the corresponding labels of the words. A CRF models the *conditional probability* of the variables we want to predict ($\boldsymbol{L}$) given the observed variables ($\boldsymbol{W}$). It assumes that conditioned on $\boldsymbol{W}$, the $\boldsymbol{L}$ forms a *Markov random field* with topology described by a graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$. Vertex $i \in \boldsymbol{V}$ in the graph represents the random variable L_i, and the edges connecting the vertices specify the dependence relationship between the random variables.

Let *clique*($\boldsymbol{G}$) be the set of cliques in graph $\boldsymbol{G}$, where a clique of a graph is a fully connected sub-graph of that graph. The distribution of $\boldsymbol{L}$ given $\boldsymbol{W}$ can be written as follows:

$$P(\boldsymbol{L} = \boldsymbol{l}|\boldsymbol{W} = \boldsymbol{w}) = \frac{1}{Z(\boldsymbol{w})} \prod_{C \in clique\ (\boldsymbol{G})} \phi_{\boldsymbol{C}}(\boldsymbol{w}, \boldsymbol{l}_C) \quad (2)$$

In the equation, $\phi_{\boldsymbol{C}}(\boldsymbol{w}, \boldsymbol{l}_C)$ is a *local function* defined on variables $\boldsymbol{W}$ and $\boldsymbol{L}_C$, where $\boldsymbol{L}_C = (L_i : i \in C)$, and $Z(\boldsymbol{w})$ is the *normalization constant* such that $\sum_{\boldsymbol{l}} P(\boldsymbol{L} = \boldsymbol{l}|\boldsymbol{W} = \boldsymbol{w}) = 1$:

$$Z(\boldsymbol{w}) = \sum_{\boldsymbol{l}} \prod_{C \in clique\ (G)} \phi_C(\boldsymbol{w}, \boldsymbol{l}_C) \quad (3)$$

Unlike a traditional classifier which predicts the labels of the individual words in a ticket without considering the dependency of the labels, a CRF classifier can model such dependency and hence improve the accuracy. For example, a *linear chain* CRF models the dependency of the labels of neighboring words.

Often we consider an *exponential family* of distributions where a local function ϕ_C is a weighted summation of features (as we will see in the next subsection, this form can be derived from a *maximum entropy principle* [2]):

$$P_{\boldsymbol{\lambda}}(\boldsymbol{L} = \boldsymbol{l}|\boldsymbol{W} = \boldsymbol{w}) = \frac{1}{Z(\boldsymbol{w})} \exp\left\{\sum_{C \in clique\ (G)} \sum_{1 \le k \le K} \lambda_k f_k(\boldsymbol{w}, \boldsymbol{l}_C)\right\} \quad (4)$$

In equation (4), $\{f_1, f_2, \dots\}$ is a set of *feature functions* defined on the labels and the words, and $\{\lambda_1, \lambda_2 \dots\}$ is a set of *weights* associated with the features. There is a weight λ for each feature f. Feature design and selection plays a very important role for the accuracy of the model, which we will describe in the next section.

B. Training

Under the framework of *supervised learning*, we are given a set of labeled data: $D = \{\boldsymbol{w}^{(t)}, \boldsymbol{l}^{(t)}\}_{t=1}^{T}$, where $\boldsymbol{w}^{(t)}$ is the *t*-th ticket in the training data, and $\boldsymbol{l}^{(t)}$ consists of the labels of the words in that ticket. The (conditional) *log likelihood* of the data with feature weight parameter $\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_K)$ is:

$$l_D(\boldsymbol{\lambda}) = \sum_{t=1}^{T} \log\left(P_{\boldsymbol{\lambda}}(\boldsymbol{l}^{(t)}|\boldsymbol{w}^{(t)})\right) \quad (5)$$

In the *parameter learning* step (also known as the *training* step), we want to find the optimal weight vector $\boldsymbol{\lambda}^*$ that maximizes the log likelihood:

$$\boldsymbol{\lambda}^* = \text{argmax}\, l_D(\boldsymbol{\lambda}) \quad (6)$$

$\boldsymbol{\lambda}^*$ is called the *maximum likelihood estimator (MLE)* of the feature weights. The resulted model $P_{\boldsymbol{\lambda}^*}(\boldsymbol{L}|\boldsymbol{W})$ (5) also has the *maximum entropy* interpretation. The principle of maximum entropy [2] states that the resulting probability distribution maximizes the (conditional) entropy (where $\tilde{P}(\boldsymbol{w})$ is the empirical distribution of the training data):

$$H(P) = -\sum_{\boldsymbol{w},\boldsymbol{l}} \tilde{P}(\boldsymbol{w})P(\boldsymbol{l}|\boldsymbol{w})\log P(\boldsymbol{l}|\boldsymbol{w}) \quad (7)$$

while at the same time meets a set of constraints. The constraints are set by the training data. In other words, the model tries its best to maximize events seen in the training data, while assuming nothing of the unseen events.

In practice, a *regularization term* such as L_2 regularization is often added to the log likelihood to avoid over-fitting. There exist several techniques to search for the optimal parameters, including *iterative scaling* type of algorithms like Generalized Iterative Scaling (GIS) [4], Improved Iterative Scaling (IIS), and *quasi-Newton* methods like L-BFGS.

C. Decoding

After the optimal feature weights $\boldsymbol{\lambda}^*$ are estimated from the training data, in the *inference* step (also known as the *decoding* step), for any new ticket $\boldsymbol{w}$, we want to find the *most likely* labels $\boldsymbol{l}^*$ of the words in $\boldsymbol{w}$:

$$\boldsymbol{l}^* = \text{argmax}_{\boldsymbol{l}}\, P_{\boldsymbol{\lambda}^*}(\boldsymbol{L} = \boldsymbol{l}|\boldsymbol{W} = \boldsymbol{w}) \quad (8)$$

The *max-product* algorithm (a *dynamic programming* algorithm) can be used to compute $\boldsymbol{l}^*$. From the decoded labels, we can predict whether the ticket contains a server name or not.

For linear chain CRFs, the complexity of learning and inference grows linearly with the training and testing data size and the number of features, while quadratically with the number of labels. More generally, for order-*m* CRFs, the complexity grows exponentially in *m*+1. When exact inference is not computationally feasible, we can apply *approximate inference* methods such as *loopy belief propagation* or *Gibbs sampling* [5].

V. FEATURES FOR THE MODEL

As we will see in our experiments, selecting a good set of features is very important for the CRF classifier to produce accurate predictions. Without loss of generality, the features used by the probabilistic model (4) are binary-valued and can be represented as questions on the words. For example, a feature may be specified by

$$f_k(w_i, l_i) = \begin{cases} 1 & \text{if } w_i \text{ contains one or more digits and } l_i = 1 \\ 0 & \text{otherwise} \end{cases}$$

This feature can be thought of asking the question "does word w_i contain one or more digits and is it a server name?" If in the training data, the word under examination (the current token) does contain one or more digits, for example, w_i = "*snoopy12dog*", and it is identified as a server name, then the above feature *fires* (i.e., takes value 1) and it will contribute to the conditional distribution in (4). Note that features with larger weights will contribute more to the distribution.

In our experiments, we use three types of features.

A. Word Features

The first type of feature examines the current word w_i and the neighboring words around it. These features have the form $f_k(\boldsymbol{w}_{I(i)}, l_i)$, where $\boldsymbol{w}_{I(i)}$ contains certain neighboring words around w_i, and l_i is the label associated with the current token w_i. For example, if $I(i) = \{i-2, i-1, i\}$, then $\boldsymbol{w}_{I(i)} = (w_{i-2}, w_{i-1}, w_i)$ contains the two words to the left of the current token w_i as well as w_i. The feature fires if $\boldsymbol{w}_{I(i)}$ matches certain 3-gram words and l_i takes certain value.

B. Pattern Features

This type of feature inspects certain properties of the current word w_i as well as the neighboring words around it, in conjunction with the label associated with the word. For example, this type of feature may check whether w_i (and its neighboring words) contains numerical numbers or not, e.g., if w_i = "*abc001*" and l_i =1, then the feature fires. As another example, it may check whether w_i is of the form of an IP address, e.g., if w_i = "*10.252.2.157*" and l_i=1, then the feature fires. The punctuations, e.g. ">","<", are also presented by this type of feature.

One can think of these features as a regular *pattern* matching on the word. If a certain pattern is present in the word and its label takes certain value, the feature fires.

C. Higher Order Features

The features introduced in the previous two subsections only examine the label of the current token w_i. These features are called *first-order* features. For a CRF classifier, we can also include features that can model the dependency of the labels of neighboring words. These features are called *higher-order* features.

For example, a linear chain CRF examines the dependency of the labels of two consecutive words. To incorporate such dependency, for a feature illustrated in Section V.A, $f_k(\boldsymbol{w}_{I(i)}, l_i)$, we can introduce another feature $f'_k(\boldsymbol{w}_{I(i)}, l_{i-1}, l_i)$, which will fire if and only if $\boldsymbol{w}_{I(i)}$ matches certain word sequence and both the previous label and the current label take certain values. These are called *second-order* features.

VI. EXPERIMENT RESULTS

A. Experiment Setup

CRF is a supervised learning method. It requires a set of training data. In its applications to other fields, e.g. Natural Language Processing (NLP), the annotated linguistic resource is often attainable from various agencies such as Linguistic Data Consortium (LDC). Different conferences or workshops also coordinate common data sets for various algorithms and system integration comparison. In service research area, however, such luxury is rarely afforded. Although there are usually ample data for mining, there are virtually no annotated data for either supervised learning or evaluation.

In order to train the CRF classifier, we need labeled data. Instead of spending huge amount of manual efforts in data annotation, we used an active learning / boot strapping approach adopted from previous research [12]. A dictionary (which could be incomplete and error prone) is first extracted from the server configuration database. It is then followed by a fuzzy match algorithm of the dictionary applied on the

unstructured ticket data. As mentioned in section II, many *free* text tickets are ill-formed and may contain typos, concatenations, short-hands, affixes, and so on. The incomplete and error nature of the dictionary coupled with the ill-formatted tickets complicates the server name detection problem. Otherwise, it would be a simple table lookup to find the server names.

The fuzzy match rules are established by running the partial matches between each list in the dictionary with every token from the tickets. The partial match results are then summarized into the regular expression patterns. Although noisy, the dictionary is still useable, to some extent, to extract reasonable initial training data set. The CRF classifier is then trained by this automatically labeled data. After the initial model is established, active learning can be applied to extract more data for further model improvement.

A total of 29,212 tickets (with a total of 1.5 M tokens from ticket abstract and description) are extracted from the process for our experiments. Table 1 lists a few examples of tickets with the annotations for the experiments. In the annotation, 1 means that the token is a server name and 0 means it is not.

Ticket 1: The server dif.gtq2.com is unavailable or not pingable by tiv. Label: 0 0 1 0 0 0 0 0 0 0 Ticket 2: Please restart ESP service on AWEJSP03. Label: 0 0 0 0 0 1 Ticket 3: Not able to open my lotus notes from onanywhere.xyz.com. Label: 0 0 0 0 0 0 0 0 0 1

Table 1: Sample tickets with their labels

B. Evaluation Metrics and Initial Results

The tickets have been partitioned uniformly into 90% for training set and 10% for testing set. The partitions are repeated 10 times (round-robin) so every ticket has been tested once and the test set and training set are non-overlapping.

After the model is run on the test data, precision and recall metrics are used to evaluate the results. *Precision* measures how accurate the model is in predicting the labels. *Recall* measures how many of the server names the model can identify. Both metrics are useful, but both can be skewed if taken separately. A third metric, *F-measure,* combines precision and recall, and gives a more balanced evaluation. The F-measure is defined as:

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

Table 2 shows the initial results using *n*-gram features on the first partition data set. There are 26,290 tickets for training and 2,922 tickets for testing. In measuring the performance, we are more interested in finding the server names (the '1' label in Table 1). The non-server ('0' label in Table 1) rows have very high performance numbers. It is a easier and less interesting problem. We then focus on server labels performance in the rest of the analysis.

Label	Precision%	Recall%	F-measure%
Server	95.75	90.64	93.12
Non-server	99.50	99.75	99.60

Table 2: Server name identification results

C. Feature Set Comparsions

We then conduct the experiments on the 10 fold partitions data set. Three different features sets are designed to explore the performance improvement. The first feature set, denoted as CRF-F1, includes the *n*-gram word features. The second feature set, CRF-F2, uses additional pattern features as we discussed in Session V.B. For example, is the token a common English word? Does the token contain both alphanumeric alphabets? Is the token an IP4 or IP6 address? Is the token a punctuation, e.g. ">","<", etc? The third feature set, CRF-F3, uses CRF-F2 features with larger window sizes. All three sets of features use second-order features so the classifier is a linear chain CRF classifier. The results in Table 2 are based on CRF-F1, which already achieves very good performance. It might be wondering why new features are needed. Session VI.E will address the details.

CRF w/ different features	Precision Mean% (Std%)	Recall Mean% (Std%)	F-measure Mean% (Std%)
CRF-F1	95.45 (0.29)	91.33 (0.48)	93.34 (0.23)
CRF-F2	95.61 (0.45)	95.08 (0.41)	95.34 (0.24)
CRF-F3	95.92 (0.44)	95.40 (0.43)	95.66 (0.23)

Table 3. 10 folds Round Robin CRF results with different feature sets

Table 3 shows the average performance and the standard deviation of 10 folds partitions. The recall is improved substantially after the pattern features are introduced. More features yield slightly better performance in precision; however, they are somewhat flat across all three feature sets. The small amount of the standard deviations demonstrates the data sets are homogenous and our modeling framework works well when we have enough training data. The 96% of precision and 95% of the recall (from CRF-F3) mean that for the given ticket sets, 95% of the server names in the tickets can be identified, 5% of the server names are missed. Besides, from the proposed server names, 96% are indeed server names, only 4 % are not server names.

D. Error Analysis

In addition to the quantitative evaluation shown in Table 3, preliminary qualitative evaluation is also carried out on a portion of the test data. We inspect a portion of the *false negative* cases where the model predicts the token as non-server name whereas the fuzzy match indicates that it is a server name. We find that some of these cases arise because the sentences are incomplete. The ticket is cut off when it is over maximum length.

We further examine 206 *false positive* cases where the dictionary-based fuzzy match tags a word as *not* a server name, while the CRF model tags it as a server name. These 206 cases are manually examined and 146 of them are actually server names. Table 4 shows some examples:

Word	Dictionary fuzzy match	CRF	Human Evaluation
nraxa004.r.com	No	Yes	Yes
nbdzws222.com	No	Yes	Yes
fsgwds080	No	Yes	Yes
pojbatwapp01.jvt.qa	No	Yes	Yes
var.usf.raleigh.com	No	Yes	Yes

Table 4: Examples of false positive

It is noteworthy that the model learns and discovers new server names that are not in the dictionary with encouraging accuracy (146/206=71%). This can be a valuable tool for IT service delivery in general because it helps to automate the discovery process which is usually labor intensive.

E. Robustness

In addition to the data set in Table 3, we also run on four additional data sets which are evaluated manually. Each set of tickets are from one particular client. The training data is the complete ticket data except the given client; thus the test data is somewhat blind to the training procedure. This setup can be used to evaluate the model robustness against new data. In this experiment, three sets of the data have high 90% of fuzzy matching rate. These data can be used to compare the fuzzy match to our model approach. The forth set has only 0.2% fuzzy match rate. It implies that the dictionary for that set is impaired.

Table 5 shows the results. The 2nd row illustrates the ticket size for a client. The 6th row "Fuzzy Match" shows the percentage of tickets which contain a server name according to the Fuzzy match. For client 4 in our experiment, out of 1338 tickets, fuzzy match finds server names in only 0.2% of the tickets. The 3th to 5th column shows the percentage of tickets that contain a server name extracted by CRF with various feature sets.

	Client1	Client2	Client3	Client4
total ticket	662	284	7637	1338
CRF-F1	35.80%	91.90%	64.14%	76.53%
CRF-F2	81.57%	93.31%	87.04%	84.45%
CRF-F3	86.86%	93.66%	90.73%	83.71%
fuzzy match	92.00%	93.64%	90.00%	0.15%

Table 5. Fuzzy Match vs. CRF

As can be seen from Table 5, the performance of CRF-F1 is not consistent for client 1, 2 and 3. It performs reasonable well on client 2 but falling apart for client 1. The CRF-F2 and CRF-F3 yield more consistent results for all three clients. The CRF-F3 clearly demonstrates the best performance. The evaluation shows that the CRF model without dictionary can achieve very competitive performance comparing to fuzzy match with dictionary.

Upon manual inspection, some of client 1's tickets are much different than the rest of the tickets. The performance degradation is more significant. By comparing table 3 and table 5, the n-gram word features (CRF-F1) might work reasonable well if the test data is similar to training data. However, additional feature sets and larger window size make the CRF model more robust against unseen data. These results justify our efforts to pursue better feature set, e.g. CRF-F2 and CRF-F3.

The dictionary-based fuzzy matching performs rather miserably on client 4 because the majority of the client 4 dictionary is missing. The CRF approach finds much more server names. When the dictionary is impaired and the fuzzy match struggles, our proposed CRF approach can be a useful tool to extract the server name information. In additions, these server names extracted from client 4 can be used to validate the client 4 server databases. This can be a very good diagnosis tool to investigate possible reasons for the impaired data source.

VII. BUSINESS INSIGHTS

A. Ticket-to-server ratio

One of the main purposes of finding server names in tickets is to gain business insights into customers' problems. Accurate identification of server names enables IT delivery service to compute and enhance business statistics. A very important *statistic* in servicing tickets is the ticket-to-server ratio, which shows how tickets are related to a particular server. The higher the ratio is, the more problematic the server is. This statistic enables the IT delivery service to prioritize the workload, giving more urgency and attention to servers with many tickets. The ability to accurately uncover server names in the tickets directly impact the insight gained from the statistic.

Figures 2 and 3 show two distributions of the ticket-to-server statistic. Figure 2 shows results from the naïve approach of dictionary-based fuzzy matching and Figure 3 illustrates results from our CRF approach of finding server names. These results are based on the ticket data from client-4 discussed in Table 5. The X-axis shows the server names and the Y-axis shows the number of tickets for that server. The fuzzy match only finds two tickets with server names in them. It is evident from the figures that more meaningful distribution is found based on the CRF approach. From figure 3, one can try to diagnose top problems servers, ntcsvp06, brksvw18, etc. to find out if there are easy fixes to reduce the tickets.

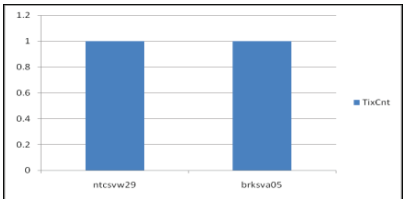

Figure 2. Ticket-to-server ratio by Fuzzy Match

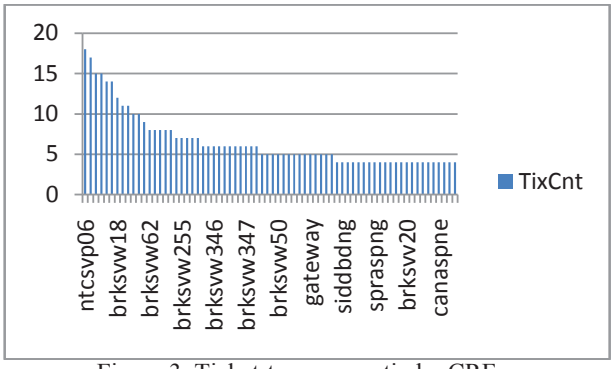

Figure 3. Ticket-to-server ratio by CRF

Another scenario is the server to severity=n ratio analysis. Every ticket has a severity value associated with it. One can calculate *ticket to server ratio* based on severity=1 to help IT engineering to address those high priority troublesome servers.

B. Static business insights

New insights can be obtained when the ticket information is linked with the asset information. In data center IT management, typical configuration information includes OS types, manufacture names, machine types, CPUs, models, software installations, etc. After linking asset db with tickets by server name, the configuration information can be jointed with each ticket, and various configuration parameters to ticket ratio statistics can be calculated for value business insight.

Figure 4 shows an example of OS types to tickets ratio. The X-axis lists all OS variations, and the Y-axis shows the number of tickets for each OS variation. From the figure, we can conclude that Windows OS is driving most of the ticket workload in the given data center. Without linking, such insight would be very difficult, if not impossible, to gather. Additional configuration types, e.g. machine types, vendors, hardware ages, etc, to tickets ratio can be derived for more business insights.

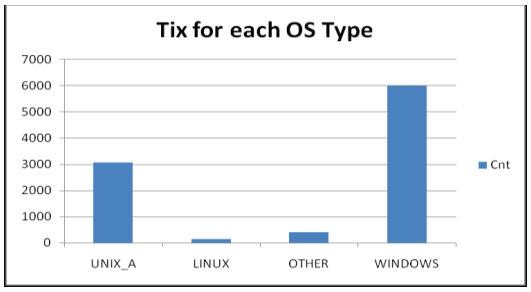

Figure 4. Number of tickets for each OS type

VIII. CONCLUSION AND FUTURE WORK

In this paper we present a statistical machine learning model for detecting server names in free text tickets. The server names are keys to link the tickets with various resources from other business units in IT service delivery. Without knowing the server names, knowledge from different content domains remain separate and disconnected. With server names, the information can be integrated. We show examples of how to use the machine learning approach to facilitate the capture of meaningful business insight. Finding sever names that are absent in the dictionary further supports the effort of data validation. Relevant business units can be alerted when a server is present in the tickets but absent in the dictionary. The scenario serves not only as a warning sign for data integrity issues but also a timely indication that the customer data may be incomplete, in which case the customer should be contacted to resolve the inconsistency.

One next step following the identification of server names is to extend the asset configuration database and link together all the information pertaining to the server. More static and dynamic business insights can be derived from the extended configuration database to discover if a certain combination of configurations is prone to fail.

Obtaining a comprehensive view of the information from various business units is essential to service delivery success. The ability to analytically process unstructured ticket texts opens up a wider range of data integration and knowledge understanding.

REFERENCES

[1] Rakesh Agrawal, Tomasz Imielinski, Arun Swami. Mining Association Rules Between Sets of Items in Large Databases, SIGMOD Conference 1993:207-216

[2] Adam L Berger, Stephen Della Pietra, Vincent J Della Pietra, 1996. A Maximum Entropy Approach to Natural Language Processing. Computational Linguistics 22(1): 39-71

[3] Christian Borgelt. Recursion Pruning for the Apriori Algorithm 2nd Workshop of Frequent Item Set Mining Implementations, FIMI 2004, Brighton, UK.

[4] J. N. Darroch, D. Ratcliff, 1972. Generalized Iterative Scaling for Log-Linear Models. In Annals of Mathematical Statistics 43(5): 1470-1480.

[5] J. Finkel, T. Grenager, C. D. Manning, 2005. Incorporating non-local information into information extraction systems by Gibbs sampling. In Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics.

[6] Ea-Ee Jan, Brian Kingsbury, 2010. Rapid and inexpensive development of speech action classifier for natural lanugage call routing systems. IEEE workshop on spoken language technology. Page 336-341

[7] Sangkyum Kim, Winnie Cheng, Shang Guo, Laura Luan, Daniela Rosu, Abhijit Bose. Polygraph: System for Dynamic Reduction of False Alerts in Large-Scale IT Service Delivery Environments, Usenix ATC 2011

[8] J. Lafferty, A. McCallum, F. Pereira, 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. Proc. 18th International Conf. on Machine Learning. Morgan Kaufmann. pp. 282–289.

[9] J. Lenchner, D. Rosu, N.F. Velasquez, S. Guo, K. Christiance, D. Defelice, P. Deshpande, K. Kummamuru, N. Kraus, L. Luan, D. Majumdar, M McLaughlin, S. Ofek-Koifman, C. Perng. H. Roitman, C. Ward, J. Young. A service delivery platform for server management services. IBM J. of Research and Development. Sep, 2009. pp 1-17

[10] Gengxin Miao, Louise E. Moser, Xifeng Yan, Shu Tao, Yi Chen, Nikos Anerousis. Generative Models for Ticket Resolution in Expert Networks. KDD 2010 Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, 2010, page. 733-742

[11] Office of Government Commerce, 2001. Service Delivery. IT Infrastructure Library.The Stationery Office. ISBN 0113300174.

[12] Daniela Rosu, Winnie Cheng, Ea-Ee Jan, Naga Ayachitula, 2012. IEEE SOLI. Connecting the Dots in IT Service Delivery