This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2587807, IEEE Transactions on Network and Service Management

Resolution Recommendation for Event Tickets in Service Management

Wubai Zhou,Liang Tang, Chunqiu Zeng, Tao Li, Larisa Shwartz, and Genady Ya. Grabarnik

Abstract—In recent years, IT service providers have rapidly achieved an automated service delivery model. Software monitoring systems are designed to actively collect and signal event occurrences and, when necessary, automatically generate incident tickets. Repeating events generate similar tickets, which in turn have a vast number of repeated problem resolutions likely to be found in earlier tickets.

In our work, we develop techniques to recommend appropriate resolution for incoming events by making use of similarities between the events and historical resolutions of similar events. Built on the traditional K Nearest Neighbor algorithm, our proposed algorithms take into account false positives often generated by monitoring systems. An additional penalty is incorporated into the algorithms to control the number of misleading resolutions in the recommendation results. Moreover, as the effectiveness of the K Nearest Neighbor algorithm heavily relies on the underlying similarity measurement, we proposed two other approaches to significantly improve our recommendation with respect to resolution relevance. One approach uses topic-level features to incorporate resolution information into the similarity measurement; and the other uses metric learning to learn a more effective similarity measure. Extensive empirical evaluations on three ticket data sets demonstrate the effectiveness and efficiency of our proposed methods.

Index Terms—IT Service Management, Recommender System, K nearest neighbor, Latent Dirichlet Allocation, Metric Learning

1 Introduction

The competitive business climate and the complexity of service environments dictate the need for efficient and cost-effective service delivery and support. They are largely achieved through service-providing facilities that collaborate with system management tools, combined with automation of routine maintenance procedures, such as problem detection, determination and resolution for the service infrastructure [1], [2], [3], [4], [5]. Automatic problem detection is typically realized by system monitoring software, such as IBM Tivoli Monitoring [6] and HP OpenView [7]. Monitoring continuously captures the events and generates incident tickets when alerts are raised. Deployment of monitoring solutions is a first step towards fully automated delivery of a service. Automated problem resolution, however, is a hard problem.

W. Zhou, C. Zeng and T. Li, the School of Computer Science, Florida International University, Miami, Florida (e-mail: {wzhou005, czeng001, taoli}@cs.fiu.edu)

L. Shwartz, the IBM T.J Watson Research Center, Yorktown Heights, New York (e-mail: {lshwart}@us.ibm.com)

G. Grabarnik, the St John's University, Jamaica, New York (e-mail: {grabarng}@stjohns.edu)

With the development of e-commerce, a substantial amount of research has been devoted to recommendation systems. These recommendation systems determine items or products to be recommended based on prior behavior of the user or similar users and on the item itself. An increasing number of user interactions has provided these applications with a huge volume of historical information that can be converted into practical knowledge. In this paper, we apply a similar approach and develop a method that finds a resolution for an event by making use of similarities between the events and previous resolutions of monitoring tickets. Most service providers keep years' worth of historical tickets with their resolutions. Each event is stored as a database record that consists of several related attributes (Table 1) with values describing the system status at the time this event was generated. For example, a CPU-related event usually contains the CPU utilization and paging utilization information. If an event keeps occuring, it is sent to the ticketing system and becomes one component of tickets, the other component is the problem resolution. The problem resolution of every ticket is stored as a textual description of the steps taken by the system administrator to resolve this problem, with resulting costs in human labor.

TABLE 1: Event Attribute Types

Type	**Example**
Categorical	OSTYPE, NODE, ALERTKEY,...
Numeric	SEVERITY, LASTUPDATE, ...
Textual	SUMMARY,...

TABLE 2: An sample monitoring ticket

SUMMARY	The logical disk C: has a low amount of free space. Percent available: 5 Threshold: 5	
SEVERITY	**FIRSTOCCURRENCE**	**LASTOCCURRENCE**
2	2014-03-01 00:54:20	2014-03-01 00:54:20
RESOLUTION	C drive was successfully cleared. It has now over than ten percent of available space.	
CAUSE	**ACTIONABLE**	**LASTUPDATE**
AUTOMATION	Actionable	2014-04-28 15:00:12

The resolution is usually collected as free-form text and describes steps taken to remediate the issue described in the ticket. We analyzed historical monitoring tickets collected from three different accounts managed by one of the large service providers (an account is an aggregate of services that

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2587807, IEEE Transactions on Network and Service Management

uses a common infrastructure). We noticed that there are many repeating resolutions for monitoring tickets within an account. It is natural to expect that if events are similar, then their respective tickets probably have the same resolution. Therefore, we can recommend a resolution for an incoming ticket based on the event information and historical tickets.

A traditional KNN-based (K Nearest Neighbor) [8] approach using attribute-level features has been first applied to provide resolution recommendations for incoming tickets in service management. However, the traditional KNN-based algorithm has one big drawback when applied to system management.

In automated service management, the resolutions of false tickets reveal unhelpful comments, such as, "this is a false alarm," "everything is fine" and "no problem found." If a false ticket's resolution is recommended for a real ticket, the system administrator might overlook the real system problem. Note that in a large enterprise IT environment, overlooking a real system problem may have serious consequences, such as system crashes. However, the traditional KNN-based algorithm has no mechanism to avoid such a case in the ticket resolution recommendation task. To overcome the aforementioned drawback, we propose two approaches to eliminate misleading resolutions. One takes a two-step strategy; the other incorporates an additional penalty to minimize misleading resolutions. Although the approaches have been successfully used in practice, they still retain two major limitations:

- **Representation of monitoring tickets:** In the KNN-based approach, attribute-based features are used to represent monitoring tickets. However, attribute-level feature representation is not interpretable and often contains a lot of noise. In practice, each monitoring ticket describes an existing problem (e.g., low capacity, high CPU utilization) in service and the associated ticket resolutions should be highly relevant to the problems. Therefore, it is better to use features semantically capturing such problems, instead of attribute-level features, to represent monitoring tickets.
- **Similarity Measurement:** The similarity measure only considers the event component, and ignores the resolution component. In addition, each feature is treated equally when computing the similarity measure. However, the resolutions often reveal their prevalence in historical tickets and contain important information about the events, which can be used to improve the recommendation performance. Moreover, different features should have different weights in computing the similarity measure as they often play different roles in representing the tickets.

Therefore, we propose two additional approaches to address the aforementioned limitations in recommending ticket resolutions for service management. One adopts topic-level features and incorporates resolutions' information; the other utilizes metric learning to gain a more accurate similarity measurement. In summary, we make the following contributions:

- We analyze historical monitoring tickets from three production accounts and observe that their resolutions are recommendable for current monitoring tickets on the basis of event information.
- We first apply traditional KNN algorithm for event ticket resolution recommendation and extend it to two other resolution recommendation algorithms capable of eliminating misleading resolutions.
- We adopt a feature extraction approach capable of representing both the event and resolution information using topic-level features obtained via the LDA model, and we propose two recommendation algorithms to further improve the similarity measurement.
- We conduct extensive experiments for our proposed algorithms on real ticket datasets. Experimental results demonstrate the effectiveness and efficiency of the proposed approaches.

The traditional KNN-based recommendation methodology is first proposed in our preliminary work [9] and its detail and extended algorithms are fully discussed here. The rest of the paper is organized as follows: Section 2 briefly introduces the workflow of the infrastructure management of an automated service and shares our observations on three sets of monitoring tickets. In Section 3, we present resolution recommendation algorithms for monitoring tickets. Section 4 and Section 5 more specifically illustrate the details on proposed algorithms to remove misleading resolutions and improve underlying similarity measurement between events. Section 6 discusses some detailed implementation issues. In Section 7, we present experimental studies on real monitoring tickets. Section 8 describes related work about service infrastructure management and recommendation systems. Finally, Section 9 concludes our paper and discusses our future work.

2 Background

In this section, we first provide an overview of automated service infrastructure monitoring with ticket generation and resolution. Then we present our analysis on real ticket data sets.

2.1 Automated Services Infrastructure Monitoring and Event Tickets

The typical workflow of problem detection, determination and resolution in services infrastructure management is prescribed by the ITIL specification [10]. Problem detection is usually provided by monitoring software, which computes metrics for hardware and software performance at regular intervals. The metrics are then matched against acceptable thresholds. A violation induces an alert. If the violation persists beyond a specified period, the monitor emits an event. Events from the entire service infrastructure are accumulated in an enterprise console that uses rule-, case- or knowledge-based engines to analyze the monitoring events and decide whether to open an incident ticket in the ticketing system. The incident tickets created from the monitoring events are called monitoring tickets. Additional tickets called manual tickets are created upon customer request. The information accumulated in the ticket is used by technical support for problem determination and resolution. However, in this paper, we focus only on those monitoring tickets generated by a service management system (see Figure 1).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2587807, IEEE Transactions on Network and Service Management

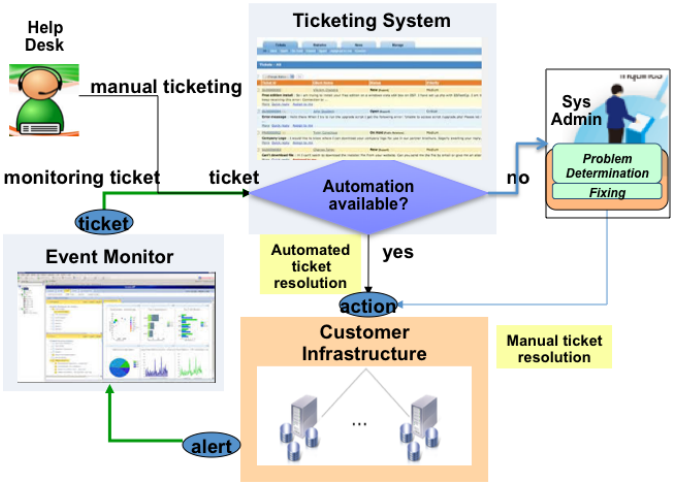


Fig. 1: Service Management System

2.2 Repeated Resolutions of Monitoring Tickets

We analyzed ticket data from three different accounts managed by IBM Global Services. Many ticket resolutions repeatedly appear in the ticket database. For example, for a low disk capacity ticket, usual resolutions mean deletion of temporal files, backup data, or addition of a new disk.

TABLE 3: Data Summary

Data set	Num. of Tickets	Time Frame
account1	31,447	1 month
account2	37,482	4 months
account2	29,057	5 months

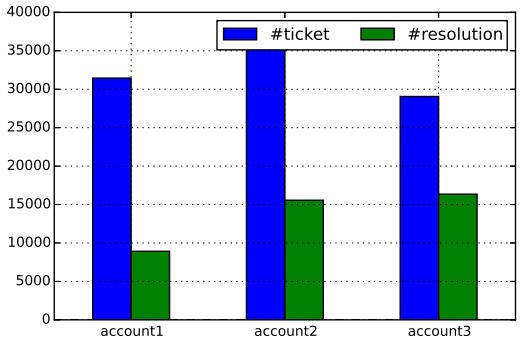


Fig. 2: Numbers of Tickets and Distinct Resolutions

The collected ticket sets from the three accounts are denoted by "account1," "account2" and "account3," respectively. Table 3 summarizes the three data sets. Figure 2 shows the numbers of tickets and distinct resolutions and Figure 3 shows the top repeated resolutions in "account1" denoted by "resolution ID." We observe that a single resolution can resolve multiple monitoring tickets. In other words, multiple tickets share the same resolutions.

3 Resolution Recommendation

In this section, we first give the problem statement and an overview for all algorithms, introduce the basic KNN-based recommendation algorithm, and then present our improved algorithms.

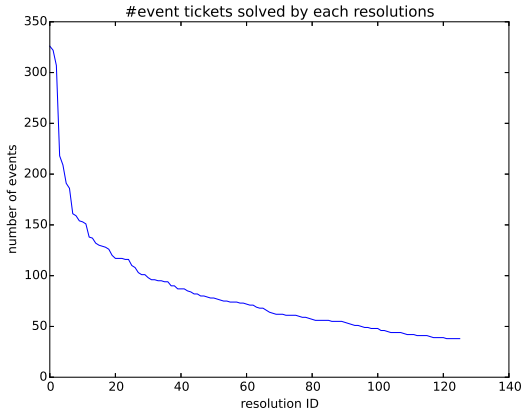


Fig. 3: Number of monitoring tickets resolved by each resolution denoted by "resolution ID" in account1

3.1 Problem Statement

Given an incoming ticket, the objective of the resolution recommendation is to find k resolutions as close as possible to the true one for some user-specified parameter k from the historical tickets.

Each historical ticket has two components: 1) ticket event and 2) ticket resolution. Each component has a few attributes that can be type of categorical, numerical or textual. An incoming ticket is one that has no resolution component. Moreover, tickets are encoded as **feature-level** attributes in which each ticket is considered as composition of attributes and similarity between tickets are calculated as the sum of attributes' similarity according to equation 1, or **topic-level** attributes in which each ticket is considered as the probability distribution of topics extracted using the approach described in Section 5.1.

3.2 The Overview of Proposed Algorithms

Figure 4 summarizes the differences between resolution recommendation algorithms. Six different algorithms are included in the figure and are organized into two groups based on their purposes:

- WKNN [11]: the basic weighted KNN algorithm using **attribute-level** features.
- Proposed approaches aiming to remove misleading/false resolutions:
  - Division: the algorithm adopts a two-step approach for recommendation: first classifying an incoming event into a true or false ticket, then recommending resolutions based on its class.
  - Fusion: the algorithm incorporates an additional penalty to minimize misleading resolutions.
- Proposed approaches aiming to improve underlying similarity measurement used by WKNN:
  - LDABaselineKNN: the algorithm uses topic-level features obtained via Latent Dirichlet Allocation [12] (LDA).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2587807, IEEE Transactions on Network and Service Management

– CombinedLDAKNN: the algorithm incorporates both the event and resolution information with top-level features.
– MLCombinedLDAKNN: the algorithm uses the similarity measure obtained from metric learning (when resolution categories are available).

Figure 4 clearly illustrates the differences among these recommendation methods. The details of the proposed algorithms will be described in detail in Section 4.1, 4.2, 5.1, 5.2, and 5.3, respectively.

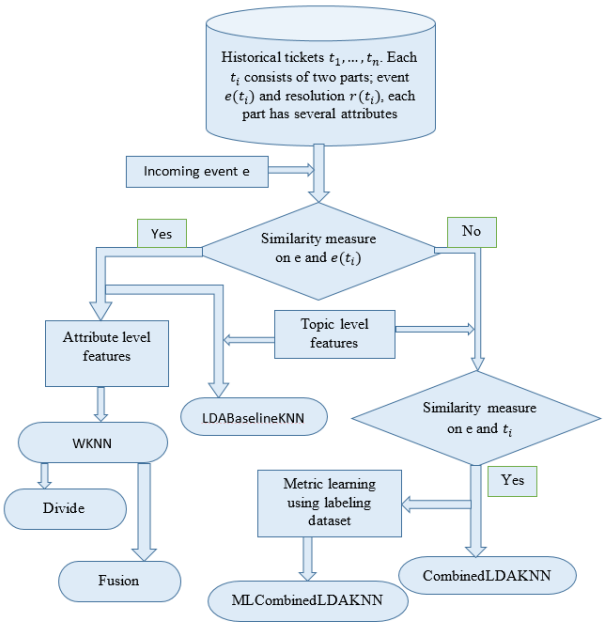


Fig. 4: Differences between proposed algorithms. For example; "LDABaselineKNN" and "CombinedLDAKNN" both use topic level features; however "LDABaselineKNN" searches nearest neighbors based on similarity between e and $e(t_i)$, the other adopts similarity between e and t_i.

3.3 Basic KNN-based Recommendation

The recommendation problem is often related to that of predicting the top k possible resolutions. A straightforward approach is to apply the KNN algorithm, which searches the K nearest neighbors of the given ticket (K is a predefined parameter), and recommends the top $k \leq K$ representative resolutions among them [13], [14]. The nearest neighbors are indicated by similarities of the associated events of the tickets. In this paper, the representativeness is measured by the number of occurrences in the K neighbors.

Table 4 lists the notations used in this paper. Let $D = \{t_1, ..., t_n\}$ be the set of historical monitoring tickets and t_i be the i-th ticket in D, $i = 1, ..., n$. Given a monitoring ticket t, the nearest neighbor of t is the ticket t_i which maximizes $sim(e(t), e(t_i))$, $t_i \in D$, where $sim(\cdot, \cdot)$ is a similarity function for events. Each event consists of event attributes with values. Let $A(e)$ denote the set of attributes of event e. The similarity for events is computed as the summation of the similarities for all attributes. There are three types of event attributes: categorical, numeric and textual (shown by Table 1). Given an attribute a and two events e_1 and e_2, $a \in A(e_1)$ and $a \in A(e_2)$, the values of a in e_1 and e_2 are denoted by $a(e_1)$ and $a(e_2)$. The similarity of e_1 and e_2 with respect to a is

$$sim_a(e_1, e_2) = \begin{cases} I[a(e_1) = a(e_2)], & \text{if } a \text{ is categorical,} \\ \frac{|a(e_1) - a(e_2)|}{max|a(e_i) - a(e_j)|}, & \text{if } a \text{ is numeric,} \\ Jaccard(a(e_1), a(e_2)), & \text{if } a \text{ is textual,} \end{cases}$$

where $I(\cdot)$ is the indicator function returning 1 if the input condition holds, and 0 otherwise. Let $max|a(e_i) - a(e_j)|$ be the size of the value range of a. $Jaccard(\cdot, \cdot)$ is the Jaccard index for the *bag of words model* [15], frequently used to compute the similarity of two texts. Its value is the proportion of common words in the two texts. Note that for any type of attribute, inequality $0 \leq sim_a(e_1, e_2) \leq 1$ holds. Then, the similarity for two events e_1 and e_2 is computed as

$$sim(e_1, e_2) = \frac{\sum_{a \in A(e_1) \cap A(e_2)} sim_a(e_1, e_2)}{|A(e_1) \cup A(e_2)|}. \quad (1)$$

Clearly, $0 \leq sim(e_1, e_2) \leq 1$. To identify the type of attribute a, we only need to scan all appearing values of a. If all values are composed of digits and a dot, a is numeric. If some value of a contains a sentence or phrase, then a is textual. Otherwise, a is categorical.

TABLE 4: Notations

Notation	Description
D	Set of historical tickets
$\|\cdot\|$	Size of a set
t_i	i-th monitoring ticket
$r(t_i)$	Resolution description of t_i
$e(t_i)$	The associated event of ticket t_i
$c(t_i)$	Type of ticket t_i, $c(t_i) = 1$ indicates t_i is a real ticket, $c(t_i) = 0$ indicates t_i is a false ticket.
$A(e)$	Set of attributes of event e
$sim(e_1, e_2)$	Similarity of events e_1 and e_2
$sim_a(e_1, e_2)$	Similarity of a values of event e_1 and e_2
K	Number of nearest neighbors in the KNN algorithm
k	Number of recommended resolutions for a ticket, $k \leq K$

4 Extension to Eliminate Misleading Resolutions

Traditional recommendation algorithms focus on the accuracy of the recommended results. However, in automated service management, false alarms are unavoidable in both the historical and incoming tickets [2]. The resolutions of false tickets are short comments such as, "this is a false alarm," "everything is fine" and "no problem found." If we recommend a false ticket's resolution for a real ticket, it would cause the system administrator to overlook the real system problem. Moreover, none of the information in this resolution is helpful. Note that in a large enterprise IT environment, overlooking a real system problem may have serious consequences such as system crashes. Therefore, we consider incorporating penalties in the recommendation results. There are two cases meriting a penalty: recommendation of a false ticket's resolution for a real ticket, and recommendation of a real ticket's resolution for a false ticket. The penalty in the first case should be larger since the real ticket is more important. The two cases are

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2587807, IEEE Transactions on Network and Service Management

analogous to the *false negative* and *false positive* in prediction problems [14], but note that our recommendation target is the ticket resolution, not its type. A false ticket's event may also have a high similarity with that of a real one. The objective of the recommendation algorithm is now maximized accuracy under minimized penalty.

4.1 The Division Method

A straightforward solution consists of dividing all historical tickets into two sets: the real tickets and the false tickets. Then, it builds a KNN-based recommender for each set respectively. Another ticket type predictor is created, establishing whether an incoming ticket is real or false, with the appropriate recommender used accordingly. In particular, the division method works as follows: it first uses a type predictor to predict whether the incoming ticket is real or false. If it is real, then it recommends the tickets from the real historic tickets; if it is false, it recommends the tickets from the false historic tickets. The historic tickets are already processed by the system administrator, so their types are known and we do not have to predict them.

The division method is simple, but relies heavily on the precision of the ticket type predictor, which cannot be perfect. If the ticket type prediction is correct, there will be no penalty for any recommendation result. If the ticket type prediction is wrong, every recommended resolution will incur a penalty. For example, if the incoming ticket is real, but the predictor says it is a false ticket, then this method only recommends false tickets. As a result, all the recommendations would incur penalties.

4.2 The Probabilistic Fusion Method

To overcome the limitation of the division method, we develop a probabilistic fusion method. The framework of the basic KNN-based recommendation is retained, with the difference that the penalty and probability distribution of the ticket type are incorporated in the similarity function.

Let λ be the penalty for recommending a false ticket's resolution for a real ticket, and $1-\lambda$ for recommending a real ticket's resolution for a false one. λ can be specified by the system administrator based on the actual cost of missing a real alert, $0 \leq \lambda \leq 1$. Larger λ indicates a greater importance of real tickets. The penalty function is

$$\lambda_t(t_i) = \begin{cases} \lambda, & t \text{ is a real ticket, } t_i \text{ is a false ticket} \\ 1-\lambda, & t \text{ is a false ticket, } t_i \text{ is a real ticket} \\ 0, & \text{otherwise,} \end{cases}$$

where t is the incoming ticket and t_i is the historical one whose resolution is recommended for t. Conversely, an award function can be defined as $f_t(t_i) = 1 - \lambda_t(t_i)$. Since $0 \leq \lambda_t(t_i) \leq 1$, $0 \leq f_t(t_i) \leq 1$.

Let $c(\cdot)$ denote the ticket type. $c(t_i) = 1$ indicates t_i is a real ticket; $c(t_i) = 0$ indicates t_i is a false ticket. The idea of this method is to incorporate the expected award in the similarity function. The new similarity function $sim'(\cdot,\cdot)$ is defined as:

$$sim'(e(t), e(t_i)) = E[f_t(t_i)] \cdot sim(e(t), e(t_i)), \quad (2)$$

where $sim(\cdot,\cdot)$ is the original similarity function defined by Eq. (1), and $E[f_t(t_i)]$ is the expected award, $E[f_t(t_i)] = 1 - E[\lambda_t(t_i)]$. If t_i and t have the same ticket type then $E[f_t(t_i)] = 1$ and $sim'(e(t), e(t_i)) = sim(e(t), e(t_i))$, otherwise $sim'(e(t), e(t_i)) < sim(e(t), e(t_i))$. Generally, the expected award is computed as

$$\begin{aligned} E[f_t(t_i)] &= E[1 - \lambda_t(t_i)] = 1 - E[\lambda_t(t_i)] \\ &= 1 - \sum_{c(t), c(t_i) \in 0,1} P[c(t), c(t_i)] \lambda_t(t_i). \end{aligned}$$

We can assume that a new ticket t and historical ticket t_i are independent, i.e., $P[c(t), c(t_i)] = P[c(t)] \cdot P[c(t_i)]$. Then, the expected penalty is

$$E[\lambda_t(t_i)] = \sum_{c(t), c(t_i) \in 0,1} P[c(t)] \cdot P[c(t_i)] \cdot \lambda_t(t_i).$$

Since $c(t_i)$ is already fixed, substituting $\lambda_t(t_i)$, we obtain

$$E[\lambda_t(t_i)] = \begin{cases} P[c(t) = 0] \cdot (1 - \lambda), & t_i \text{ real ticket} \\ P[c(t) = 1] \cdot \lambda, & t_i \text{ false ticket} \end{cases}$$

Note that all factors in the new similarity function are of the same scale, i.e., $[0, 1]$, thus $0 \leq sim'(\cdot,\cdot) \leq 1$.

4.3 Prediction of Ticket Type

Given an incoming ticket t, the probabilistic fusion method needs to estimate the distribution of $P[c(t)]$. The division method also has to predict whether t is a real ticket or a false ticket. Since t is an incoming ticket, the value of $c(t)$ is not known. However, using a ticket type predictor, we can estimate the distribution of $P[c(t)]$. There are many binary classification algorithms for estimating $P[c(t)]$. In our implementation, we utilize another KNN classifier. The features are the event attributes and the classification label is the ticket type. The KNN classifier first finds the K nearest tickets in D, denoted as $D_K = \{t_{j_1}, ..., t_{j_k}\}$. Then, $P[c(t) = 1]$ is the proportion of real tickets in D_K and $P[c(t) = 0]$ is the proportion of false tickets in D_K. Formally,

$$\begin{aligned} P[c(t) = 1] &= |\{t_j | t_j \in D_K, c(t_j) = 1\}|/K \\ P[c(t) = 0] &= 1 - P[c(t) = 1]. \end{aligned}$$

Once we have the estimate of the distribution of $P[c(t)]$, the prediction of ticket type can be performed by comparing $P[c(t) = 1]$ and $P[c(t) = 0]$.

5 Extension to Improve Accuracy

5.1 Representation of Monitoring Tickets

As shown in Section 3.3, attribute level features are used in the traditional KNN algorithm for recommendation. However, attribute-level feature representation is not interpretable and often contains a lot of noise.

Our observation indicates that each monitoring ticket describes the existing problems (e.g., low capacity, high CPU, utilization) in service, and the associated ticket resolution should be highly relevant to the problems. For example, Table 5 presents some sample monitoring tickets for "low free space" and their corresponding resolutions. The problems in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2587807, IEEE Transactions on Network and Service Management

these tickets are described by the "SUMMARY" attribute and they all share the similar semantic meaning "low free space". Therefore, it is better to use features that semantically capture these problems, instead of attribute-level features, to represent monitoring tickets.

In this paper, we apply Latent Dirichlet Allocation [12](LDA) to perform feature extraction, which can first extract hidden topics and then encode monitoring tickets using topic level features. LDA is a generative probabilistic model of a document corpus. Its basic idea is that documents are represented as random mixtures over latent topics, where each topic is characterized by a distribution over words. In particular, LDA represents documents as mixtures of topics that generate words with certain probabilities. With LDA, the documents are written as follows: 1) decide on the number of words in the document; 2) choose a topic mixture for the document; and 3) generate each word by picking a topic and using the topic to generate the word itself. Assuming this generative model for a document collection, LDA then finds the latent topics as well as various distributions. Learning the various distribution (the set of topics, their associated word probabilities, the topic of each word, and the topic probabilities of each document) is a problem of Bayesian Inference [12]. Topic probability distribution of a document is commonly used as its feature vector.

We process tickets as follows to encode them in the same feature space:

- Represent each monitoring ticket as a document by concatenating each textual attribute after stop words removal and tokenization.
- Using historical tickets to train an LDA model.
- Inference feature vectors using the trained LDA model for both incoming events and historical monitoring tickets.

Once monitoring tickets are encoded as feature vectors, then the cosine similarity can be applied to measure their similarities. Experiments in Section 7 demonstrate that the algorithm performance based on topic level features is better than that on attribute level features.

5.2 Incorporating the Resolution Information

In previous KNN-based recommendation approaches, resolutions are ranked according to the similarity measurement using the event information only. However, the resolutions often reveal their prevalence in historical tickets and contain important information about the events, which can be used to improve the recommendation performance. There are two practical motivations for incorporating the resolution information:

1) In a K nearest neighbor search, historical tickets with resolutions that are highly relevant to an incoming event should be ranked higher than those tickets having similar event descriptions, but with fewer related resolutions.
2) In a K nearest neighbor search, those tickets with resolutions that are more prevalent should be ranked higher than those with less prevalent resolutions, even if their event descriptions are similar.

Table 5 presents four tickets having similar event descriptions (shown in the "SUMMARY" attribute) from account1. All four tickets describe a "low free space" problem. In practice, however, the resolution from Ticket 1 should have a higher rank than the one from Ticket 4 since the resolution from Ticket 1 is more informative. Similarly, resolutions from Ticket 1 and Ticket 2 should have higher ranks than the one from Ticket 3 because of their higher prevalence.

TABLE 5: Tickets for explaining motivation of incorporating resolution information

ticketID	SUMMARY	RESOLUTION
1	The logical disk has a low amount of free space. Percent available: 2 Threshold: 5	After deleting old uninstall files, the logical disk now has over 10% of free disk space.
2	The percentage of used space in the logic disk is 90 percent. Threshold: 90 percent	After deleting old uninstall files, the logical disk now has over 15% of free disk space.
3	File system is low. The percentage of available space in the file system is 10 percent. Threshold: 90 percent	After delprof run, the server now has more than 4gb of free space
4	The logical disk has a low amount of free space. Percent available: 3 Threshold: 5	No trouble was found, situation no longer persists.

In Section 3.3, $sim(e, e(t_i))$ is computed to find the K nearest neighbors of an incoming event e, in which $e(t_i)$ is the event information associated with the i-th ticket. To incorporate the resolution information, $sim(e, t_i)$ (i.e., similarity between an incoming event and the i-th ticket), rather than $sim(e, e(t_i))$, is used in the algorithm. $sim(e, t_i)$ can be easily computed since e and t_i can be vectorized with the same dimensions after using topic-level features. Experiments in Section 7 demonstrate the effectiveness of this proposed approach.

5.3 Metric Learning

In previous sections, we improved the recommendation algorithm by using topic-level features and incorporating resolution information into a K nearest neighbor search. However, we still treat each feature equally in computing the similarity measure. According to our observation, topics extracted from the LDA model should have different contributions to the similarity measurement since some topics contain the major descriptive words about events while the others may consist of fewer meaningful words. For example, Table 6 lists two topics for illustration. Apparently Topic 30 contains more descriptive words than Topic 14, and thus we should assign a larger weight to Topic 30 in the similarity measurement. We adopt metric learning [16] to achieve this goal.

TABLE 6: First 6 words are extracted to represent topics trained from LDA

topicID	SUMMARY
14	server wsfpp1 lppza0 lppzi0 nalac application
30	server hung condition responding application apps

The metric learning [17] problem aims at learning a distance function tuned to a particular task, and has been shown to be useful when used in conjunction with nearest-neighbor

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2587807, IEEE Transactions on Network and Service Management

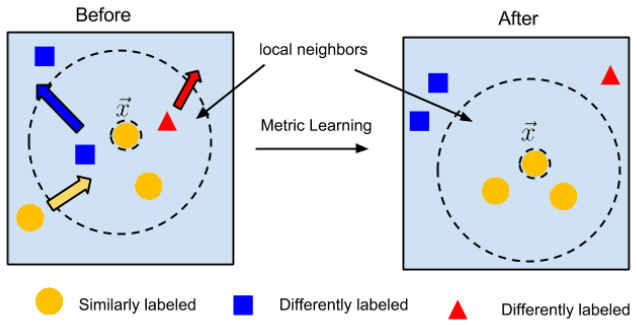


Fig. 5: Before metric learning, its neighbors include instance from different classes; after metric learning, its neighbors are all belong to a same class.

methods and other techniques that rely on distances or similarities [18] as illustrated in Figure 5. Mahalanobis Distance is commonly used for vectorized inputs, which can avoid the scenario in which one feature dominates in the computation of the Euclidean distance. In the metric learning literature, the term "Mahalanobis distance" is often used to denote any distance function of the following form:

$$d_A(x, y) = (x - y)^T A(x - y), \tag{3}$$

where A is some positive semi-definite (PSD) matrix, and x, y are the feature vectors. To facilitate the learning process, in metric learning, a slightly modified form of distance function is commonly used, as described below [16]:

$$d_A(x, y) = x^T Ay. \tag{4}$$

In our work, we have n historical tickets $t_1, t_2, \ldots, t_n$ and n corresponding resolutions $r(t_1), r(t_2), \ldots, r(t_n)$. We consider the resolution categories as supervision for metric learning since intuitively similar resolutions solve similar issues. We pre-calculate matrix $R \in R^{n*n}$ in which $R_{i,j} = sim(r(t_i), r(t_j))$. The resolution categories are usually provided by system administrators. With the available category information, the similarity between two resolutions is computed as follows:

$$sim(r(t_i), r(t_j)) = \begin{cases} 1, & \text{if } r(t_i), r(t_j) \text{ have same category,} \\ 0, & \text{otherwise.} \end{cases}$$

Our goal is to learn a similarity function $S_A(\vec{t_i}, \vec{t_j})$ by solving the following optimization problem:

$$\begin{aligned} minf(A) &= min \sum_{i=1}^{n} \sum_{j=1}^{n} ||R_{i,j} - S_A(\vec{t_i}, \vec{t_j})||^2 \\ &= min||R - SAS^T||^2, \end{aligned} \tag{5}$$

in which we use $S_A(\vec{t_i}, \vec{t_j}) = \vec{t_i}^T * A * \vec{t_j}$ ($\vec{t_i}$ and $\vec{t_j}$ are feature vectors for ticket t_i and t_j) instead of $S_A(e(\vec{t_i}), e(\vec{t_j}))$ as we want to keep the benefits of incorporating the resolution information into K nearest search. Since matrix A is constrained to be a PSD matrix, the projected gradient descent algorithm can be directly applied to solve the optimization problem in Equation 5. In each iteration of gradient descent, the new updated matrix A will be projected into a PSD matrix as the initial value for the next iteration. The singular value thresholding [19] has been applied to project A into a PSD matrix by setting all A's negative eigenvalues to be zero. The following is the gradient for Equation 5:

$$\begin{aligned} \nabla f(A) &= \nabla_A((R - SAS^T)^T(R - SAS^T)) \\ &= 2S^T SAS^T S - 2S^T AS \end{aligned}$$

6 Implementation

In this section, we discuss two issues in implementing the resolution recommendation system.

Finding Nearest Neighbors: Finding the K nearest neighbors in a large collection of historical tickets is time-consuming. There are many standard indexing search methods, such as k-d Tree [20], R-Tree [21], VP-Tree [22], and cover tree [23]. However, in resolution recommendation, the search space of our monitoring tickets is generally not metric and the dimensionality is typically very high. Therefore, locality sensitive hashing [24] is more practical and suitable. Another heuristic solution is the attribute clustering-based method. Different system events have different system attributes, and the clustering algorithm can easily separate all tickets into categories based on their attribute names. If two events share very few common attributes, their similarity cannot be high. Therefore, in most cases, the nearest neighbors search only needs to access those tickets in the same category. In our experiments, we used clustering and locality sensitive hashing to efficiently find nearest neighbors.

Redundancy Removal in Recommendation: KNN-based recommendation algorithms recommend the top k representative resolutions in the K nearest tickets. However, since all of these are similar to the incoming ticket, the resolutions of the K tickets may also be similar to each other, so that there may be some redundancy in the recommended results. To avoid this issue, another validation step is applied. First, the K nearest tickets' resolutions are sorted according to their representativeness in descending order. Then, we go through all K resolutions and check whether or not each of them is redundant to any previously selected resolution. If it is, we skip this resolution and jump to the next one; otherwise, we add it to the selection. Since the resolutions are textual descriptions, the redundancy of two resolutions is measured by the Jaccard index, $Jaccard(\cdot, \cdot)$, introduced in Section 3.3. In practice, if the descriptions of two resolutions $r(t_1)$ and $r(t_2)$ have more than one half common words (i.e. $Jaccard(r(t_1), r(t_2)) > 0.5$), the two resolutions are quite likely to be the same.

7 Evaluation

We implemented six algorithms denoted by WKNN, Division, Fusion, LDABaselineKNN, CombinedLDAKNN and MLCombinedLDAKNN, respectively whose brief interpretations are given in Section 3.2. These algorithms are all based on the weighted KNN algorithm framework. We show our experimental results from two perspectives. We first compare experimental results from WKNN, Divide, and Fusion to show the effectiveness of our proposed methods in eliminating

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2587807, IEEE Transactions on Network and Service Management

misleading resolutions, and then compare experimental results from LDABaselineKNN, CombinedLDAKNN and ML-CombinedLDAKNN to show the efficiency of our proposed methods in improving recommended resolutions' relevance. We use the Weighted KNN algorithm as the underlying algorithm because it is the most widely used Top-N item-based recommendation algorithm.

7.1 Experimental Data

Experimental monitoring tickets are collected from three accounts managed by IBM Global Services, denoted later as "account1," "account2" and "account3." The monitoring events are captured by IBM Tivoli Monitoring [25]. The ticket sets are summarized in Table 3. To evaluate metric learning, 1000 labeled tickets with resolution categories are obtained from "account1". Table 7 shows three sample categories of resolutions [26].

TABLE 7: Three resolution categories with the event description they resolved

Resolution Category	Resolved Event Key Words
Server Unavailable	Server unavailable due to unexpected shutdown, reboot, defect hardware, system hanging
Disk/FS Capacity shortage	Disk or file system capacity problems and disk failure
Performance inefficiency	Performance and capacity problems of CPU or memory

7.2 Evaluation Metric

The following evaluation measures are used in our experiments.

7.2.1 Weighted Accuracy

For each ticket set, the first 90% tickets are used as the historic tickets and the remaining 10% tickets are used for testing. Hit rate is a widely used metric for evaluating the accuracy in item-based recommendation algorithms [27], [28], [29].

$$\text{Accuracy} = \text{Hit-Rate} = |Hit(C)|/|C|,$$

where C is the testing set, and $Hit(C)$ is the set for which one of the recommended resolutions is *hit* by the true resolution. Here we define a recommended resolution as a *hit* if it has a jaccard similarity greater than a *threshold* with the ground truth resolution.

Since real tickets are more important than false ones, we define another accuracy measure, the weighted accuracy, which assigns weights to real and false tickets. The weighted accuracy (WA) is computed as follows:

$$\text{WA} = \frac{\lambda \cdot |Hit(C_{real})| + (1-\lambda) \cdot |Hit(C_{false})|}{\lambda \cdot |C_{real}| + (1-\lambda) \cdot |C_{false}|},$$

where C_{real} is the set of real testing tickets, C_{false} is the set of false testing tickets, $C_{real} \cup C_{false} = C$, λ is the importance weight of the real tickets, $0 \leq \lambda \leq 1$, and also the penalty mentioned in Section 4.2. In this evaluation, $\lambda = 0.9$ since the real tickets are much more important than the false tickets in reality. We also test other large λ values, such as 0.8 and 0.99. The accuracy comparison results have no significant change. As shown by Figure 6, our two proposed algorithms have smaller penalties than the traditional KNN-based recommendation algorithms. The probabilistic fusion method outperforms the division method, which relies heavily on the ticket type predictor. Overall, our probabilistic fusion method only has about one-third of the penalties of the traditional methods.

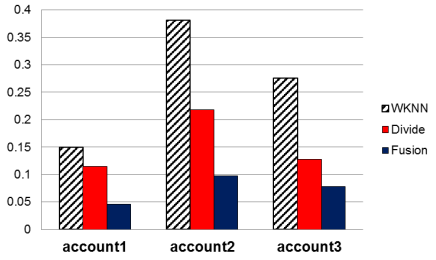


Fig. 6: Average Penalty for $K = 10$, $k = 3$

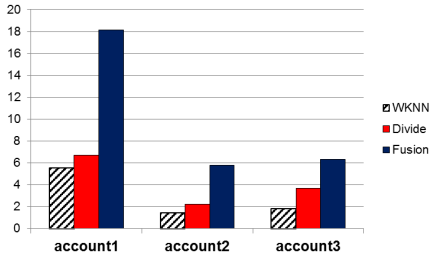


Fig. 7: Overall Score for $K = 10$, $k = 3$

7.2.2 Average Similarity

In general, several resolutions can be recommended for a single testing instance. To consider the relativeness of all recommended resolutions, the *average similarity* (avgSim) is used as one evaluation metric which is given by the following equation:

$$\text{avgSim} = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{n_i} sim(r_{io}, r_j)/n_i,$$

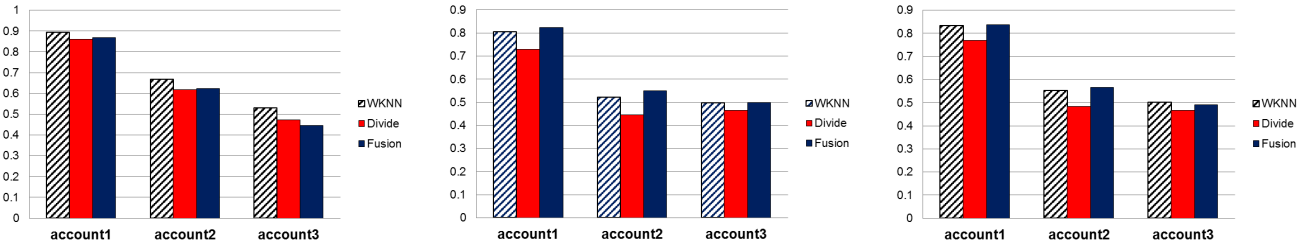
in which N is the number of testing instances, and n_i is the number of recommended resolutions for testing instance i and r_{io} is its original resolution, and r_j is its jth recommended resolution. Jaccard Similarity is used to calculate $sim(r_{io}, r_j)$.

7.2.3 Mean Average Precision

Mean Average Precision (MAP) [30] is widely used for recommendation evaluation. It considers not only the relativeness of all recommended results, but also the ranks of the recommended results.

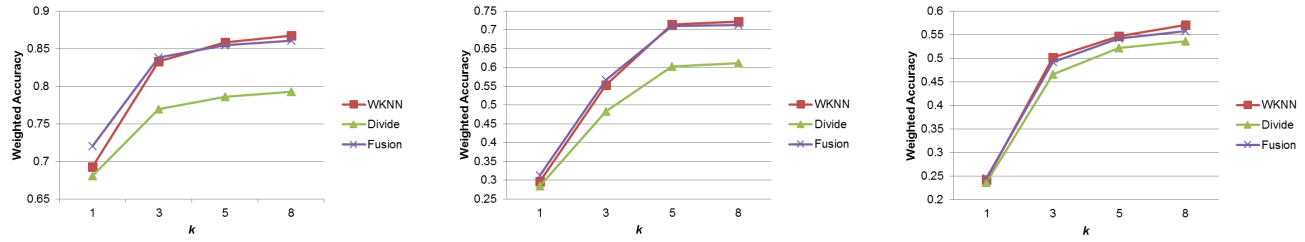
$$\text{MAP@n} = \sum_{i=1}^{N} ap@n_i/N,$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2587807, IEEE Transactions on Network and Service Management



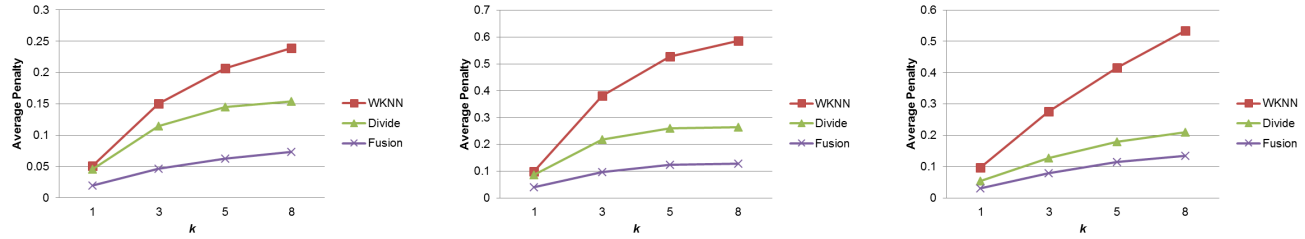
(a) Accuracy (b) Accuracy for Real Tickets (c) Weighted Accuracy

Fig. 8: Test Results for $K = 10$, $k = 3$



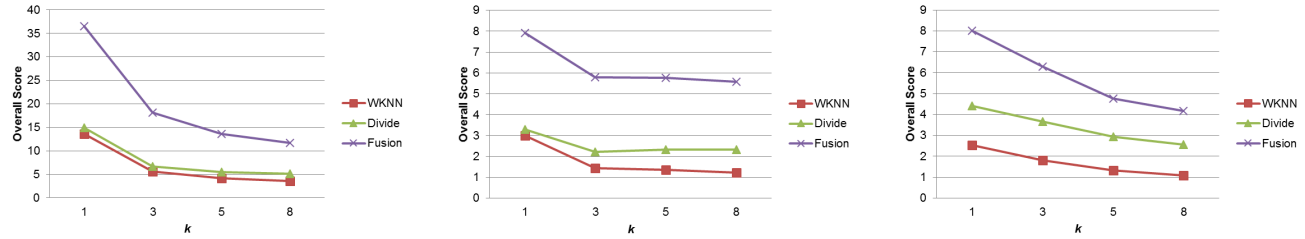
(a) account1 (b) account2 (c) account3

Fig. 9: Weighted Accuracy by varying k, $K = 10$.



(a) account1 (b) account2 (c) account3

Fig. 10: Average Penalty by varying k, $K = 10$.



(a) Overall score for account1 (b) Overall score for account2 (c) Overall score for account3

Fig. 11: Overall Score by varying k, $K = 10$

N is the number of a testing instance, $ap@n$ is given by the following equation:

$$\text{ap@n} = \sum_{k=1}^{n} p(k)\delta r(k),$$

where k is the rank in the sequence of retrieved resolutions, n is the number of retrieved resolutions, $p(k)$ is the precision at cut-off k in the list, and $\delta r(k)$ is the change in recall from items $k-1$ to k.

7.3 Overall Recommendation Performance

7.3.1 *Evaluation on eliminating misleading resolutions*

An overall quantity metric is used for evaluating the recommendation algorithms, covering both the accuracy and the average penalty. It is defined as overall score = weighted accuracy / average penalty. If the weighted accuracy is higher or the average penalty is lower, then the overall score becomes higher and the overall performance is better. Figure 7 shows the overall scores of all algorithms for two parameter settings. Figure 8 gives the accuracy scores

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2587807, IEEE Transactions on Network and Service Management

for the same parameter settings. It is seen that our proposed algorithms are always better than the WKNN algorithm in the perspective of overall score.

Variation of Parameters

To compare the results of each algorithm, we vary the number of each recommendation resolution, k. Figure 9, 10 and 11 show the weighted accuracies, average penalties and overall scores by varying k from 1 to 8, with $K = 10$. As shown by Figure 9, when we increase the value of k, the size of the recommendation results becomes larger. Then the probability of one recommended resolution being *hit* by the true resolution also increases. Therefore, the weighted accuracy becomes higher. Except for the division method, all algorithms have similar weighted accuracies for each k. However, as k is increased and there are more recommended resolutions, there are more potential penalties in the recommended resolutions. Hence, the average penalty also becomes higher (Figure 10). Finally, Figure 11 compares the overall performance by varying k. Clearly, the probabilistic fusion method outperforms other algorithms for every k.

For other values of K ranging from 8 to 20, the comparison results are very similar to $K = 10$. Usually, we set $K = 10$ and $k = 5$ in practice. The choice of k is a tradeoff between accuracy and a reasonable number of recommended resolutions. Large k decreases user experience since system administrators have to choose a proper one out of candidate resolutions, meanwhile small k greatly decreases accuracy.

A Case Study

We select an event ticket in "account1" to illustrate why our proposed algorithms are better than the traditional KNN-based algorithms. Table 8 shows a list of recommended resolutions given by each algorithm. The testing ticket is a real event ticket triggered by a low capacity alert for the file system. Its true resolution of this ticket is: "cleaned up the FS using RMAN retention policies..." RMAN is a data backup and recovery tool in Oracle database. The general idea of this resolution is to use this tool to clean up the old data.

As shown by Table 8, the first resolution recommended by WKNN is a false ticket's resolution: "No actions were taken by GLDO for this Clearing Event..." It might be caused by a temporal file generated by some application, which would clean up the temporal file automatically after its job was done. When the system administrator opened that ticket, the problem was gone, and that ticket is seen as false. However, the testing ticket is real and would not disappear unless the problem was actually fixed. This resolution from the false ticket would have misled the system administrator to overlook this problem. Consequently, a penalty of $\lambda = 0.9$ is given to WKNN.

WKNN, Divide and Fusion all successfully find the true resolution of this testing ticket, but WKNN has one false resolution, so its penalty is 0.9. Our proposed methods, Divide and Fusion, have no penalty for this ticket. Therefore, the two methods are better than WKNN.

7.3.2 Evaluation on improving accuracy

Choosing the Number of Topics

Figure 14 shows the experimental results of choosing the proper number of topics for training the LDA model using data set "account1." The results show that $numTopics = 300$ is a proper setup for the number of topics. Thus, we choose $numTopics = 300$ for all the following experiments.

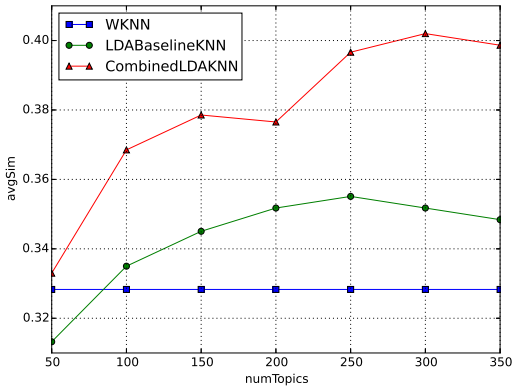


Fig. 14: Variation in accuracy for different $numTopics$ for dataset "account1"

Performance Comparison

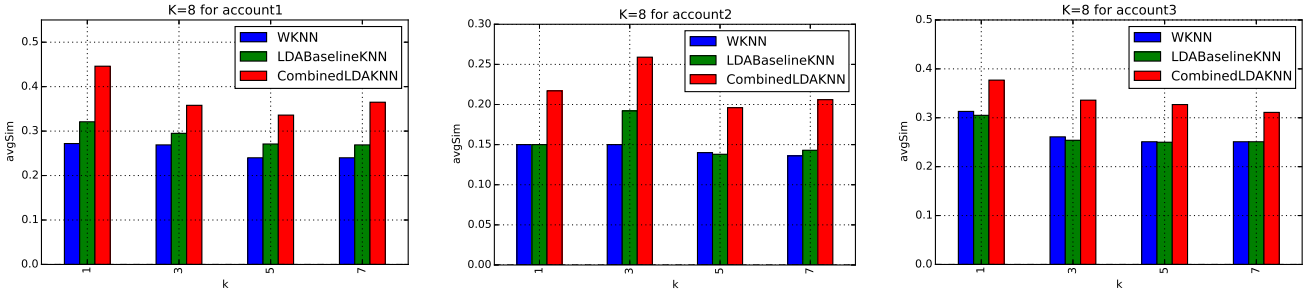
The *average similarity* is used for comparing the performance among WKNN, LDABaselineKNN and CombinedLDAKNN. When resolution categories are available, *MAP@n* is used for comparing the performance between CombinedLDAKNN and MLCombinedLDAKNN since it explicitly considers the relativeness of the recommended results.

To compare the results of each algorithm, we vary the number of recommended resolutions, k. Figure 12 shows the *average similarity* scores by setting $k = 1, 3, 5, 7$ separately, with $K = 8$; and Figure 13 show the *average similarity* scores by setting $k = 1, 3, 5, 7$ separately, with $K = 16$. We have the following observations from the figures: 1) LDABaselineKNN outperforms WKNN on account1 and account 2. On account3, LDABaselineKNN achieves similar performance as WKNN (In fact, the performance of LDABaselineKNN is slightly worse than that of WKNN when k is less than 3). This shows that in general, topic-level features are generally more effective than attribute-level features. Note that the time frame of account3 (5-month) is the largest among the three datasets. The long time frame may lead to the quality decrease of topic-level features as it is difficult for LDA to effectively capture latent topics over a long period of time. 2) CombinedLDAKNN always outperforms LDABaselineKNN and WKNN. This clearly demonstrates the effectiveness of incorporating the resolution information into K nearest neighbor search.

7.3.3 Metric Learning Performance

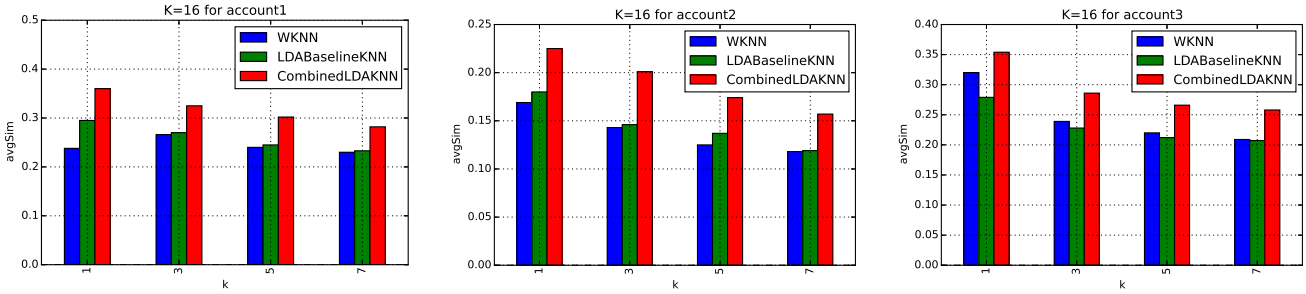
In this section, we conduct experiments to evaluate the effectiveness of using metric learning. Figure 15 and Figure 16 are used to illustrate the usefulness of metric learning. In both figures, X-axis and Y-axis are the event ID's ordered by the resolution categories, and the color value at each entry indicates the similarity score of the corresponding events. As

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2587807, IEEE Transactions on Network and Service Management



(a) account1 (b) account2 (c) account3

Fig. 12: Test Results for three accounts by varying k for $K = 8$.



(a) account1 (b) account2 (c) account3

Fig. 13: Test Results for three accounts by varying k for $K = 16$.

TABLE 8: A Case Study for $K = 10$, $k = 3$

Algorithm	Recommended Resolution	Is Hit	Is Real Ticket's Resolution	Penalty
WKNN	No actions were taken by GLDO for this Clearing Event...	no	false	0.9
	I cleaned up the FS using RMAN retention policies...	yes	true	0
	Duplicated 28106883...	no	true	0
Division	Duplicated 28106883...	no	true	0
	Another device failure has been reported for this node...	no	true	0
	I cleaned up the FS using RMAN retention policies...	yes	true	0
Fusion	Duplicated 28106883...	no	true	0
	Another device failure has been reported for this node...	no	true	0
	I cleaned up the FS using RMAN retention policies...	yes	true	0

shown in Figure 15 and Figure 16, after metric learning, similarity scores between monitoring tickets with resolutions from the same category will be enhanced while similarity scores between monitoring tickets with resolutions from different categories will be reduced. Therefore, for example, for a testing instance whose original resolution belongs to category i, more resolutions from category i will be retrieved first after applying metric learning.

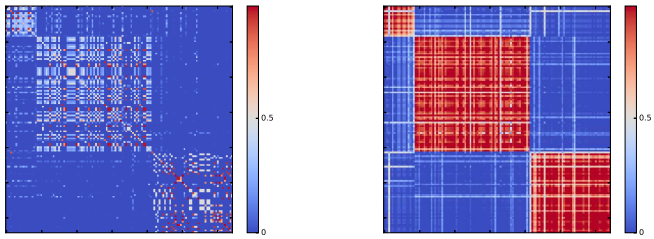


Fig. 15: Similarity measure before and after metric learning for training set

Figure 17 uses *MAP* to evaluate the performance of CombinedLDAKNN and MLCombinedLDAKNN with different values of K. As shown, the overall MAP scores of MLCombinedLDAKNN are higher and more stable than those of CombinedLDAKNN when K increases. This indicates that MLCombinedLDAKNN can retrieve more related resolutions and thus is more robust to noisy resolutions compared to CombinedLDAKNN. In summary, the experimental results from Figure 15, Figure 16, and Figure 17 demonstrates the effectiveness of metric learning in resolution recommendation.

8 Related Work

This section reviews prior research studies related to the automated IT service management and recommendation systems. System monitoring, as part of the automated service management, has become a significant research area of the IT

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2016.2587807, IEEE Transactions on Network and Service Management

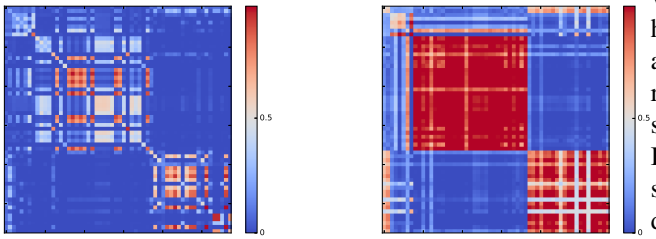


Fig. 16: Similarity measure before and after metric learning for testing set

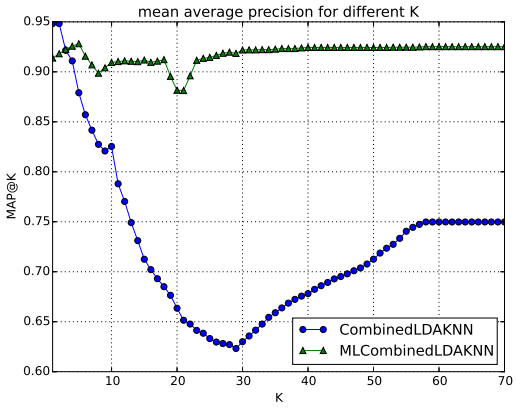


Fig. 17: Mean average precision (MAP) varying parameter K of underlying KNN algorithm

industry in the past few years. Commercial products such as IBM Tivoli [25], HP OpenView [7] and Splunk [31] provide system monitoring. Numerous studies [32] [33] [34] [35] [36] [37] focus on monitoring that is critical for a distributed network. The monitoring targets include the components or subsystems of IT infrastructures, such as the hardware of the system (CPU, hard disk) or the software (a database engine, a web server). Once certain system alarms are captured, the system monitoring software will generate the event tickets into the ticketing system. Automated ticket resolution is much harder than automated system monitoring because it requires vast domain knowledge about the target infrastructure. Some prior studies apply approaches in text mining to explore the related ticket resolutions from the ticketing database [38], [39]. Other works propose methods for refining the work order of resolving tickets [38], [40]. A number of studies focused on the analysis of historical events with the goal of improving the understanding of system behaviors. A significant amount of work was done on analysis of system log files and monitoring events. See example, [41], [42], [43]. Another area of interest is the identification of actionable patterns of events and misses, or false negatives, by the monitoring system.

False negatives are indications of a problem in the monitoring software configuration, wherein a faulty state of the system does not cause monitoring alerts.

One major cost of modern IT service is manpower. In large service providers, service centers are constituted by hundred or thousands of IT experts, who take charge of various incident tickets every day. Therefore, service providers heavily rely on human efficiency for such tasks as root cause analysis and incident ticket resolving. Automatic techniques of recommending relevant historical tickets with resolutions can significantly improve the efficiency of humans in this task. Based on the relevant tickets, the human can correlate related system problems that have happened before and perform a deeper system diagnosis. The solutions described in relevant historical tickets also provide best practices for solving similar issues.

Recommendation techniques have also been widely studied in e-commerce and online advertising areas. With the development of e-commerce and online advertising, a substantial amount of research has been devoted to the recommendation system. The existing recommendation algorithms can be categorized into two types. The first type is learning-based recommendation, in which the algorithm aims to maximize the rate of user response, such as user click or conversation. The recommendation problem is then naturally formulated as a prediction problem. It utilizes a prediction algorithm to compute the probability of the user response on each item. Then, it recommends the one having the greatest probability. Most prediction algorithms can be utilized in the recommendation, such as naive Bayes classification, linear regression, logistic regression and matrix factorization [44], [45].

The second type of recommendation algorithm focuses on the relevance of items or users, rather than the user response. Lots of algorithms proposed for promoting products to online users [46], [47], [48], [49] belong to this type. They can be categorized as item-based [13], [28], [29] and user-based algorithms [50], [48], [46], [47]. Our work in this dissertation is item-based. Every ticket is regarded as an item in our scenario. The difference between our work and traditional item-based algorithms is that, in e-commerce, products are maintained by reliable sellers, or some other procedures to assure the quality of selling products. The recommendation algorithms usually do not need to consider the problem of fake or low-quality products. But in service management, false tickets are unavoidable. The tickets with ticket resolutions are recorded in the database of the ticketing system. In some real-world ticketing systems, false tickets are the majority of all tickets. Moreover, when a ticket arrives, the recommendation algorithm does not know this alert is real or false in advance. The traditional recommendation algorithms do not take into account the types of tickets and as a result would recommend misleading resolutions.

Meanwhile, An accurate distance metric in feature space is highly essential in real-world applications. Good distance metrics are important to many data mining tasks, especially in image classification and content-based image retrieval [51], [18]. Metric learning partially overcomes the difficulties of feature extraction and similarity measurement in those domains. Metric learning requires learning a distance metric for the input space of data from a given collection of a pair of similar/dissimilar points that preserves the distance relation among the training data. In addition, many machine learning algorithms, such as KNN, heavily rely on the underlying

distance metric for the input instances. Previous work [52], [53], [54], [55], [56] has shown that appropriately designed distance metrics can significantly benefit KNN-based algorithms compared to the standard Euclidean distance.

9 Conclusion

This paper studies the problem of resolution recommendation for monitoring tickets in an automated service management. We analyze three sets of monitoring tickets collected from a production service infrastructure and identify a vast number of repeated resolutions for monitoring tickets. Based on our prior work of KNN-based recommendation, we improve the similarity measure by utilizing both the event and resolution information from historical tickets via a topic-level feature extraction using the LDA (Latent Dirichlet Allocation) model. In addition, a more effective similarity measure is learned using metric learning when resolution categories are available.

There are several avenues for future research. First, we plan to investigate and develop intelligent classification techniques to automatically label resolutions [57], [58]. Second, our current recommendation system uses KNN-based algorithms due to their simplicity and efficiency. We will investigate and develop other advanced algorithms to improve the recommendation performance. Finally, we also plan to use an active query strategy to fully automate resolution recommendations.

Acknowledgment

This work is partially supported by the U.S. National Science Foundation under grants IIS-0546280, HRD-0833093, and IIS-1213026.

References

[1] P. Marcu, L. Shwartz, G. Grabarnik, and D. Loewenstern, "Managing faults in the service delivery process of service provider coalitions," in *IEEE SCC*, 2009, pp. 65–72.

[2] L. Tang, T. Li, F. Pinel, L. Shwartz, and G. Grabarnik, "Optimizing system monitoring configurations for non-actionable alerts," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2012.

[3] N. Ayachitula, M. J. Buco, Y. Diao, M. Surendra, R. Pavuluri, L. Shwartz, and C. Ward, "IT service management automation - a hybrid methodology to integrate and orchestrate collaborative human centric and automation centric workflows," in *IEEE SCC*, 2007, pp. 574–581.

[4] B. Wassermann and W. Emmerich, "Monere: Monitoring of service compositions for failure diagnosis," in *ICSOC*, 2011, pp. 344–358.

[5] Y. Yan, P. Poizat, and L. Zhao, "Repair vs. recomposition for broken service compositions," in *ICSOC*, 2010, pp. 152–166.

[6] "IBM Tivoli Monitoring," http://ibm.com/software/tivoli/products/monitor/.

[7] "HP OpenView : Network and Systems Management Products," http://www8.hp.com/us/en/software/enterprise-software.html.

[8] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[9] L. Tang, T. Li, L. Shwartz, and G. Grabarnik, "Recommending resolutions for problems identified by monitoring," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE*. IEEE, 2013, pp. 134–142.

[10] "ITIL," http://www.itil-officialsite.com/home/home.aspx.

[11] S. A. Dudani, "The distance-weighted k-nearest-neighbor rule," *IEEE Transactions on Systems Man and Cybernetics*, vol. SMC-6, no. 4, pp. 325–327, april 1976.

[12] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.

[13] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. T. Riedl, "Application of dimensionality reduction in recommender system – a case study," in *ACM WebKDD Workshop*, 2000.

[14] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Addison Wesley, 2005.

[15] C. D. Manning, P. Raghavan, H. Schütze *et al.*, *Introduction to information retrieval*. Cambridge university press Cambridge, 2008, vol. 1.

[16] B. Kulis, "Metric learning: A survey," *Foundations & Trends in Machine Learning*, vol. 5, no. 4, pp. 287–364, 2012.

[17] B. Aurlien, H. Amaury, and S. Marc, *Metric Learning Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool Publishers, 2015.

[18] A. Frome, Y. Singer, F. Sha, and J. Malik, "Learning globally-consistent local distance functions for shape-based image retrieval and classification," in *Computer Vision, ICCV 2007*. IEEE, 2007, pp. 1–8.

[19] J.-F. Cai, E. J. Candès, and Z. Shen, "A singular value thresholding algorithm for matrix completion," *SIAM Journal on Optimization*, vol. 20, no. 4, pp. 1956–1982, 2010.

[20] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[21] A. Guttman, "R-Trees: A dynamic index structure for spatial searching," in *ACM SIGMOD*, 1984, pp. 47–57.

[22] P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *SODA*, 1993, pp. 311–321.

[23] A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," in *ICML*, 2006, pp. 97–104.

[24] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proceedings of VLDB*, Edinburgh, Scotland, UK, September 1999, pp. 518–529.

[25] "IBM Tivoli : Integrated Service Management," http://ibm.com/software/tivoli/.

[26] J. Bogojeska, I. Giurgiu, D. Lanyi, G. Stark, and D. Wiesmann, "Impact of hw and os type and currency on server availability derived from problem ticket analysis," in *Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–9.

[27] M. Deshpande and G. Karypis, "Item-based top-n recommendation algorithms," *ACM Transactions on Information Systems*, vol. 22, no. 1, pp. 143–177, Jan. 2004.

[28] G. Karypis, "Evaluation of item-based top-n recommendation algorithms," in *CIKM*, 2001, pp. 247–254.

[29] X. Ning and G. Karypis, "SLIM: Sparse linear methods for top-n recommender systems," in *ICDM*, 2011, pp. 497–506.

[30] M. Zhu, "Recall, precision and average precision," *Department of Statistics and Actuarial Science, University of Waterloo, Waterloo*, 2004.

[31] "Splunk: A commerical machine data managment engine," http://www.splunk.com/.

[32] S. R. Kashyap, J. Ramamirtham, R. Rastogi, and P. Shukla, "Efficient constraint monitoring using adaptive thresholds," in *Proceedings of ICDE*, Cancun, Mexico, 2008, pp. 526–535.

[33] S. Agrawal, S. Deb, K. V. M. Naidu, and R. Rastogi, "Efficient detection of distributed constraint violations," in *Proceedings of ICDE*, Istanbul, Turkey, 2007, pp. 1320–1324.

[34] S. Mccanne and V. Jacobson, "The bsd packet filter: A new architecture for user-level packet capture," in *USENIX Technical Conference*, 1993, pp. 259–270.

[35] K. Xu, Z.-L. Zhang, and S. Bhattacharyya, "Profiling internet backbone traffic: behavior models and applications," in *ACM SIGCOMM Conference*, 2005, pp. 169–180.

[36] C. Estan, S. Savage, and G. Varghese, "Automatically inferring patterns of resource consumption in network traffic," in *ACM SIGCOMM Conference*, 2003, pp. 137–148.

[37] M. J. Ranum, K. Landfield, M. T. Stolarchuk, M. Sienkiewicz, A. Lambeth, and E. Wall, "Implementing a generalized tool for network monitoring," in *USENIX SysAdmin Conference*, 1997, pp. 1–8.

[38] Q. Shao, Y. Chen, S. Tao, X. Yan, and N. Anerousis, "EasyTicket: a ticket routing recommendation engine for enterprise problem resolution," *PVLDB*, vol. 1, no. 2, pp. 1436–1439, 2008.

[39] D. Wang, T. Li, S. Zhu, and Y. Gong, "iHelp: An intelligent online helpdesk system," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 41, no. 1, pp. 173–182, 2011.

[40] G. Miao, L. E. Moser, X. Yan, S. Tao, Y. Chen, and N. Anerousis, "Generative models for ticket resolution in expert networks," in *KDD*, 2010, pp. 733–742.

[41] J. L. Hellerstein, S. Ma, and C.-S. Perng, "Discovering actionable patterns in event data," *IBM Systems Journal*, vol. 43, no. 3, pp. 475–493, 2002.

[42] C. Perng, D. Thoenen, G. Grabarnik, S. Ma, and J. Hellerstein, "Data-driven validation, completion and construction of event relationship networks," in *Proceedings of the ninth ACM SIGKDD*. ACM, 2003, pp. 729–734.

[43] G. Grabarnik, A. Salahshour, B. Subramanian, and S. Ma, "Generic adapter logging toolkit," in *Autonomic Computing, 2004. Proceedings. ICAC*. IEEE, 2004, pp. 308–309.

[44] C. D. Manning and H. Schuetze, *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[45] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, 2006.

[46] R. M. Bell and Y. Koren, "Scalable collaborative filtering with jointly derived neighborhood interpolation weights," in *ICDM*, 2007, pp. 43–52.

[47] Y. Ding and X. Li, "Time weight collaborative filtering," in *ACM CIKM*, 2005, pp. 485–492.

[48] Y. Koren, "Collaborative filtering with temporal dynamics," in *KDD*, 2009, pp. 447–456.

[49] L. Liu, N. Mehandjiev, and D.-L. Xu, "Multi-criteria service recommendation based on user criteria preferences," in *Proceedings of the fifth ACM conference on Recommender systems*, 2011, pp. 77–84.

[50] L. Terveen and W. Hill, "Beyond recommender systems: Helping people help each other," in *HCI in the New Millennium*, 2001, pp. 487–509.

[51] A. Frome, Y. Singer, and J. Malik, "Image retrieval and classification using local distance functions," in *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*, vol. 19. MIT Press, 2007, p. 417.

[52] X. He, O. King, W.-Y. Ma, M. Li, and H.-J. Zhang, "Learning a semantic space from user's relevance feedback for image retrieval," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 1, pp. 39–48, 2003.

[53] X. He, W.-Y. Ma, and H.-J. Zhang, "Learning an image manifold for retrieval," in *Proceedings of the 12th annual ACM ICM*. ACM, 2004, pp. 17–23.

[54] R. Yan, A. G. Hauptmann, and R. Jin, "Negative pseudo-relevance feedback in content-based video retrieval," in *Proceedings of the eleventh ACM ICM*. ACM, 2003, pp. 343–346.

[55] C. Domeniconi and D. Gunopulos, "Adaptive nearest neighbor classification using support vector machines," in *Advances in Neural Information Processing Systems*, 2001, pp. 665–672.

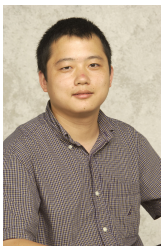
[56] K. Q. Weinberger, J. Blitzer, and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," in *Advances in neural information processing systems*, 2005, pp. 1473–1480.

[57] C. Zeng, T. Li, L. Shwartz, and G. Y. Grabarnik, "Hierarchical multi-label classification over ticket data using contextual loss," in *Proceedings of IEEE/IFIP NOMS*. IEEE, 2014, pp. 1–8.

[58] S. Chang, G.-J. Qi, J. Tang, Q. Tian, Y. Rui, and T. S. Huang, "Multimedia lego: Learning structured model by probabilistic logic ontology tree," in *Data Mining (ICDM)*. IEEE, 2013, pp. 979–984.

Liang Tang is an applied research engineer at LinkedIn. He received a Ph.D. in Computer Science from Florida International University in 2014. His research interests include recommender systems, personalized recommendation algorithms and machine learning applications on system management.

Chunqiu Zeng is a Ph.D. student in the School of Computer Science at Florida International University. He received B.S. and M.S. degrees in Computer Science from Sichuan University in 2006 and 2009, respectively. His research interests include event mining, system management and large scale data mining.



Tao Li is a professor in the School of Computer Science at Florida International University. He received his Ph.D. in Computer Science in 2004 from the University of Rochester. His research interests lie in data mining, computing system management, and information retrieval. He is a recipient of the USA NSF CAREER Award and multiple IBM Faculty Research Awards.

Larisa Shwartz (M'05) received a Ph.D. in mathematics from UNISA, University of South Africa. She is a researcher at IBM T.J. Watson Research Center, Yorktown Heights, NY, USA, with research experience in mathematics and computer science. She is currently managing operational innovation focusing on IT service management technologies for service delivery operations. She has more than 40 publications, 32 patents, and 67 patent applications.

Wubai Zhou is a Ph.D. student in the School of Computer Science at Florida International University. He received a B.S. degree in Computer Science and Technology from Wuhan University in 2012. His research interests include system oriented data mining, system management and machine learning.

Dr. Genady Ya. Grabarnik teaches in the Math and CS Department, St John's University. He is a trained mathematician and has authored over 80 papers. He spent 10 years at IBM T.J.Watson Research Center where his work was celebrated with a number of awards, including Outstanding Technical Achievement Award and Research Achievement Awards. He is a prolific inventor with over 50 US patents. His interests include research in functional analysis, inventions, and research in computer science and artificial intelligence.