Mining software support tickets for assistive routing

Han, Jianglei

2020

Han, J. (2020). Mining software support tickets for assistive routing. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/143908>

<https://doi.org/10.32657/10356/143908>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 15:06:58 SGT

Mining Software Support Tickets for Assistive Routing

Jianglei Han

School of Computer Science & Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

2020

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

09-09-2020

.....

Date

.....

Jianglei Han

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

09-09-2020

.....

Date

.....

Associate Professor Aixun SUN

Authorship Attribution Statement

This thesis contains material from [2] paper(s) published in the following peer-reviewed journal(s) / from papers accepted at conferences in which I am listed as an author.

Chapter 3 is published as [Jianglei Han and Aixin Sun](#). *Mean Average Distance to Resolver: An Evaluation Metric for Ticket Routing in Expert Network*. IEEE International Conference on Software Maintenance and Evolution (ICSME). 2017.

The contributions of the co-authors are as follows:

- I conducted the experiments and prepared the drafts.
- A/Prof Sun provided the initial project direction and edited the drafts.

Chapter 4 is published as [Jianglei Han, Ka Hian Goh, Aixin Sun, and Mohammad Akbari](#). *Towards Effective Extraction and Linking of Software Mentions from User-Generated Support Tickets*. ACM International Conference on Information and Knowledge Management (CIKM). 2018..

The contributions of the co-authors are as follows:

- I prepared data, conducted the experiments and composed the drafts.
- Ka Hian assisted in data preparation and experiments.
- A/Prof Sun guided the research direction and edited the drafts.
- Dr Akbari revised the drafts.

Chapter 5 is currently in communication for journal submission, with the title [Jianglei Han , Jing Li, and Aixin Sun](#). *UFTR: A Unified Framework for Ticket Routing*.

The contributions of the co-authors are as follows:

- I conducted the experiments and prepared the drafts.
- Dr Jing Li provided guidance in the experiments and edited the drafts.
- A/Prof Sun provided the initial project direction and edited the drafts.

Chapter 6 is published as [Jianglei Han, and Aixin Sun](#). *DeepRouting: A Deep Neural Network Approach for Ticket Routing in Expert Network*. IEEE Conference in Service Computing (ICSC). 2020.

The contributions of the co-authors are as follows:

- I conducted the experiments and prepared the drafts.
- A/Prof Sun provided the initial project direction and edited the drafts.

09-09-2020

.....

Date

.....

Jianglei Han

Acknowledgements

The SAP Industrial PhD Program provides an opportunity for me to learn from best from both academia and business. I thank SAP Asia Pte Ltd and Economic Development Board of the Government of Singapore, for providing the scholarship.

This work would not have been possible without the guidance and supports from my supervisor Dr. Aixin Sun, who introduced me to the scholarship and accepted me to his research group. I also wish to thank all the colleagues in the research group who helped and inspired me for completing this thesis, including Dr. Jing Li, Dr. Zongyang Ma, Dr Jialong Han, and Dr. Yequan Wang, for sparing the time for discussions and reviewing my manuscripts. During my years studying in Nanyang Technological University of Singapore, I have the pleasure getting acquainted with many awesome people. Many of them are established researchers working in different fields, I wish them all the success and happiness in life.

At SAP, I would like to thank my managers Steve Lee, Dr. Daniel Dahlmeier, and Dr. Chen Wang, for the careers guidance. A special regard to my first mentor Naveen Nandan, who introduced me to the company, and to Dr. Mohammad Akbari for his advice in research. For this special project on support tickets, I am always grateful to Andreas Heckmann, who envisioned an intelligent support application, and Christian Boos, who recommended me for the assignment. Since the beginning of the project, I worked closely with Project Manager Wilhelm Juetten to form a productive team, his mentorship and friendship are always appreciated. Moreover, I wish to thank David Molinera, Kazuhiro Inaba and Ka Hian Goh for their hands-on support in the experiments.

Most importantly, no word can describe my love and appreciation to my family, who are always there for me. Lastly, to my daughter. Learning is a lifelong journey, always be ready to embrace what is coming next.

Jianglei Han, March 2020

Abstract

The technical support organization of a software service provider consists of groups of experts who are processing customer tickets, which are electronic reports of any technical incident. Operationally, assigning a ticket to the correct expert is an important step towards resolution. An incorrect assignment would result in longer turnaround time and customer dissatisfaction, incurring cost and overhead.

Ticket routing problem is about finding the right expert group that is the most capable of resolving a given ticket. In the ticket processing workflow, routing takes place in both initial assignment and inter-group transfer. Industrial and academic researchers developed and published automated solutions to analyze and process tickets, using different statistical and algorithmic methods. However, a good number of the previous works focus on either initial assignment or inter-group transfer, and propose solutions accordingly. Some take a multi-stage approach, combining the routing results sequentially, without leveraging synergies between the stages. Moreover, each stage only uses a subset of the information available. With more archived data and enhanced technologies, it calls for a fresh look and unified approach to the problem.

In this thesis, we mine tickets from a software support system and investigate ticket routing problem in terms of routing performance evaluation, content analysis, and assistive routing. Firstly, we review and discuss the limitations of existing evaluation metrics and frameworks of routing systems, proposing a novel metric and the assistive routing evaluation framework. Next, we analyze tickets text and extract software product mentions, using manually crafted features. Combining insights from data mining and domain knowledge, we propose two routing frameworks based on learning-to-rank paradigm, incorporating features from ticket, group, ticket-group, and group-group feature groups. Lastly, we present a deep neural network framework, using classic semantic networks to model ticket-group interaction. In assistive routing evaluation, our proposed frameworks are more effective compared to baseline routing systems using multi-stage approach.

Contents

Acknowledgements	ix
Abstract	xi
List of Figures	xvii
List of Tables	xix
Symbols and Acronyms	xxi
1 Introduction	1
1.1 Backgrounds	1
1.2 Ticket	3
1.3 Research Problems	5
1.4 Contributions	7
1.5 Thesis Organization	9
2 Literature Review	11
2.1 Overview	11
2.2 Ticket Information Extraction	12
2.2.1 Ticket Problem Analysis	12
2.2.1.1 Text Processing Pipeline	13
2.2.1.2 Extract information from tickets	15
2.2.1.3 Issues and Root Cause Analysis	16
2.2.2 Ticket Solution Analysis	19
2.2.2.1 Solution recommendation	19
2.2.2.2 Solution quality scoring	20
2.2.3 Summary	21
2.3 Ticket Clustering and Classification	21
2.3.1 Ticket clustering	22
2.3.2 Binary classification	24
2.3.3 Multi-class classification	24
2.3.4 Hierarchical Multi-class classification	26
2.3.5 Summary	27

2.4	Ticket Routing	28
2.4.1	Ticket Routing with Classification	29
2.4.1.1	Ticket Routing with Time Constraint	31
2.4.2	Ticket Routing with Network Analysis	33
2.4.2.1	Expert network analysis	33
2.4.2.2	Using routing path only	34
2.4.2.3	Using routing path and ticket text	36
2.4.3	Summary	39
2.5	Conclusion	40
3	Evaluation of Ticket Routing Performance	41
3.1	Introduction	41
3.2	Related Work	43
3.3	Existing Evaluation Metrics	46
3.3.1	Mean Steps To Resolver	46
3.3.2	Resolution Rate	47
3.3.3	Accuracy	47
3.3.4	Discussion	47
3.4	Distance-based Routing Metric	48
3.4.1	Unification	49
3.4.2	Normalization	50
3.4.3	Quantification	50
3.4.4	Adaptability	50
3.4.5	Analysis	51
3.4.6	Scoring a generated routing path against ground truth	53
3.5	Assistive Routing Evaluation Framework	53
3.6	Experiment and Result	56
3.7	Conclusion	58
4	Software Mention Extraction and Linking	59
4.1	Introduction	59
4.2	Related Work	60
4.3	The Problem and Data Analysis	61
4.3.1	Dataset Annotation	62
4.3.2	Dataset Analysis	64
4.4	Mention Extraction and Linking	66
4.4.1	Product Names Extraction	66
4.4.2	Features	66
4.4.3	Product Name Linking	70
4.5	Experiments	70
4.5.1	Baseline methods	73
4.5.2	Experimental Results	74
4.5.3	Error analysis	78

4.6	Conclusion	79
5	Unified Framework for Ticket Routing	81
5.1	Introduction	81
5.2	Related Work	82
5.3	Preliminaries	83
5.4	Exploratory Analysis	85
5.4.1	Ticket Content Analysis	85
5.4.2	Expert Group and Network Analysis	87
5.5	The UFTR Framework	91
5.5.1	Candidate Resolver Selection	93
5.5.2	Feature Extraction	94
5.5.3	Learning to Ranking	95
5.6	Experiment	97
5.6.1	Experiment Setup	97
5.6.2	Evaluation on Ticket Routing	99
5.6.3	Evaluation on Resolver Ranking	101
5.6.4	Feature Importance	102
5.7	Conclusion	103
6	Deep Neural Network Approach for Ticket Routing	105
6.1	Introduction	105
6.2	Related Work	107
6.2.1	Ticket Routing	107
6.2.2	Multi-view deep neural network model	108
6.3	System Architecture	109
6.3.1	Text View	110
6.3.2	Graph View	111
6.3.3	Multi-view DeepRouting	112
6.4	Experiment	113
6.4.1	Resolver Ranking	114
6.4.2	Assistive Routing	116
6.5	Conclusion	120
7	Conclusions and Future Work	123
	List of Author's Publications	127

List of Figures

1.1	An example ticket with structured (ID, Meta Product) and unstructured (subject, free text) data.	3
1.2	A high-level overview of the ticket processing workflow.	4
1.3	Thesis structure with citation relations.	9
2.1	Using <i>ClassTagger</i> to label and transform sample problem [1, 2] . .	18
3.1	Comparison of different scoring methods.	49
3.2	Illustration on scoring of generated paths. In ground truth (top row), the resolver group is labeled with a green tick. Based on distances to resolver, HLDS scores of groups in ground truth are computed and labeled under the group icons. We use yellow, blue and green to represent intermediate groups in the ground truth path. In subsequent cases, groups with these colors indicate partial matches with ground truth and grey icons are unmatched groups.	54
3.3	A sample expert network, and workflow illustrations for human routing, computed routing, and assistive routing.	55
3.4	Sensitivity of parameter K from 3 to 10 in MADR metric.	57
4.1	An example ticket, with product and version mentions . They are linked to their respective official names.	61
4.2	Performance comparison of the CRF model and the neural network baseline models for the extraction task. Plots on the left column show average <i>Precision</i> , <i>Recall</i> and <i>F1</i> scores over all label entity types, grouped by method and dataset. While the right column shows the MUC metrics, which are lenient measures considering partial matching with ground truth.	71
4.3	Outputs from the end-to-end system on 3 tickets from SV, BC, and BI. The true positive results from product and version extraction are highlighted in green and yellow respectively. False negative entities are underlined. Positive linkings are in solid curves and orange boxes, while the false negative linkings are dashed curves. False positive linking is grey.	78

5.1	AC and PM are the roots for “Accounting” and “Process Management” products. A ticket is being routed from leaf group $AC \cdot BE \cdot DB$ to $PM \cdot BE \cdot DB$, where BE and DB are “back-end” and “database” components of the root product. UI stands for “user interface”.	84
5.2	Ticket length distribution in number of terms, and routing sequence lengths (right) of 500,000 archived tickets.	85
5.3	Distributions of clarity scores of archived tickets, grouped by the number of routing steps.	86
5.4	Log probability distribution of expert group being different roles, and the correlations of probability distribution of the same group playing different roles.	87
5.5	Source to resolver analysis. The left end of the x -axis is fixed for routings from dispatch. Each line characterize one expert group with the distribution of cumulative sources to resolver. The colored lines are the cluster centers (best viewed in color).	89
5.6	System architecture of the UFTR framework. Components at the upper region are artifacts or models produced from offline training processes, where are not shown in the diagram for simplicity. The lower region contains the main steps in online operation. Online and offline components exchange data through an API gateway.	92
5.7	Root level prediction using a unsupervised retrieval-based classifier, expanded with n most connected neighbours in H_{root} . The dashed line indicates the number of cases having the resolver among the candidate, over all tickets (<i>i.e.</i> , hit rate). The blue bar represents the ratio of candidates after expansion, over all expert groups in the network H_{res}	93
5.8	Performance comparisons of UFTR with baselines in evaluated by different metrics.	100
6.1	Multi-view DeepRouting computes a joint relevance score by combining relevance scores from different views.	109
6.2	Overview of text-view processing steps with name labels on the left. Labels on the right indicate the shape of outputs from each step. . .	110
6.3	Number of human routing steps vs. number of Assistive Routing steps in (a) TER-GGT, (b) UFTR-Pairwise, and (c) DeepRouting-DualView. In each subplot, bars with the same colors are represent test tickets that have 1 to 4 steps by human routing. The height of a bar indicates the ratio of tickets that resolved by 1 to 4 steps in Assistive Routing, within its respective color. Note that, we are only interested in assistive routing performance, we exclude cases whereby the resolver is identified after a model failed to recommend after n steps. Therefore, that height of bars with the same color do not necessarily sum up to 1. (Best view in color).	119

List of Tables

1.1	ITIL stages and their functional focus [3]	2
2.1	Example POS Patterns by Potharaju <i>et al.</i> [1], derieved from technical terminology patterns proposed by Justeson <i>et al.</i> [4].	14
2.2	Summary of literature on ticket clustering.	22
2.3	Summary of Methods used in Ticket Classification.	27
2.4	Resolver Search Algorithms (FM: First-order Memoryless search, FMS: First-order Multiple active State search, VMS: Variable-order Multiple active State search [5])	35
2.5	Generative models for ticket routing (RM: Resolution Model, TM: Transfer Model, ONM: Optimized Network Model [6])	36
3.1	Result of different routings; best result in boldface.	57
4.1	Statistics of datasets. Length is in number of tokens.	62
4.2	Number of entity and version mentions in 4 datasets.	63
4.3	Statistics of linkable products. $ M_s $, $ P_{linked} $, and $ P_{meta} $ are the number of unique surface forms, unique official products linked to, and unique meta-products.	63
4.4	Proportions of tickets, linkable and unlinkable product mentions with associated version mentions (w/v)	63
4.5	Mentions of ‘SAP NetWeaver 7.4’ (top table) and a list of different products “NW” has been linked to (bottom table). Exceptional and erroneous mentions are underlined.	65
4.6	Example prototype words. Labels are ‘B’egin and ‘I’nside, for ‘L’inkable, ‘U’nlinkable products and ‘V’ersions.	68
4.7	Features for linking a mention m with version v , to an official product p , and ticket-level meta-product t . $+$ is vector addition operation.	72
4.8	Ablation study of features in CRF, on MUC $F1$	75
4.9	The most effective CRF features with weights.	76
4.10	Linking module results. For each column, the best score is in bold-face and the second best is underlined.	77
5.1	Comparison of UFTR with existing routing systems in terms of feature types and models.	82
5.2	Statistics and characteristics of networks H_{trans} , H_{res} , and H_{root}	90

5.3	Features for routing model. τ is the ticket, E_τ is the entity set contained in τ . g is the candidate group, while S_τ is the current routing sequence. When a new ticket is being processed, $S_\tau = \emptyset$. . .	96
5.4	Hit Rate (HR) of UFTR for the task of resolver ranking on four test sets.	102
5.5	Most important features and their types, ranked by importance scores from the Random Forest Regressor.	103
6.1	Performance for the resolver ranking. The best model is in bold-face, while the second best case is underlined.	115
6.2	Performance comparison in Assistive Routing. The best model is in bold-face, while the second best case is underlined. The models use different information for routing, indicated by $\dagger$ for text and $\ddagger$ for graph.	118

Symbols and Acronyms

Symbols

$\mathcal{R}^n$	the n -dimensional Euclidean space
$\mathcal{H}$	the Euclidean space
$\ \cdot\ $	the 2-norm of a vector or matrix in Euclidean space
$\ \cdot\ _G$	the induced norm of a vector in G-space
$\ \cdot\ _E$	the induced norm of a vector or matrix in probabilistic space
$\odot$	the Hadamard (component-wise) product
$\otimes$	the Kronecker product
$\langle \cdot, \cdot \rangle$	the inner product of two vectors
$\circ$	the composition of functions
∇f	the gradient vector
$\mathcal{C}^k$	the function with continuous partial derivatives up to k orders
$x_{i,k}$	the i -th component of a vector x at time k
$\bar{x}$	the vector with the average of all components of x as each element
$\mathbf{1}$	all-ones column vector with proper dimension
$\mathcal{C}$	the average space, i.e., $\text{span}\{\mathbf{1}\}$
$\mathcal{C}^\perp$	the disagreement space, i.e., $\text{span}^\perp\{\mathbf{1}\}$
$\Pi_\parallel$	the projection matrix to the average space $\mathcal{C}$
$\Pi_\perp$	the projection matrix to the disagreement space $\mathcal{C}^\perp$
$O(\cdot)$	order of magnitude or ergodic convergence rate (running average)
$o(\cdot)$	non-ergodic convergence rate

Acronyms

ATTR	Actual Time to Resolver
CRF	Conditional Random Field
IR	Information Retrieval
IT	Information Technology
ITIL	IT Infrastructure Library
ITSM	IT Service Management
LAA	Latent Association Analysis
LDA	Latent Dirichlet Allocation
MSTR	Mean Steps to Resolver
MTTR	Mean Time to Resolver
ML	Machine Learning
NB	Naïve Bayes
NER	Named Entity Recognition
NLP	Nature Language Processing
POS	Part-of-Speech
RR	Resolution Rate
SLA	Service Level Agreement
SVM	Support Vector Machine
TF	Term Frequency
TF-IDF	Term Frequency Inverse Document Frequency
TRS	Ticket Routing Sequence

Chapter 1

Introduction

1.1 Backgrounds

Every industry has its standards and best practices. In Information Technology (IT), IT Infrastructure Library (ITIL) outlines standards and best practices that are widely adopted by IT service providers [3]. Its objective is to improve the service efficiency and quality to end-users, by standardizing *the selection, planning, delivery and maintenance of IT services within a business*.

ITIL covers five stages in IT service lifecycle. Table 1.1 summarizes the functional focus of each stage. The research problems in this thesis are originated from Service Operation, which focuses on the effective and efficient delivery of IT services. Service Operation is further refined by nine sub-sections, including *Event Management, Incident Management, Request Fulfillment, Access Management, Problem Management, IT Operation Control, Facilities Management, Application Management*, and *Technical Management*. Without undermining the importance and significance of the other topics, we only focus on Incident Management as the scope of our research.

Incident Management details the process and lifecycle management for technical incidents, which are defined as *unplanned interruptions or reductions in quality of IT services* by ITIL. According to the best practices, service providers should record all incidents electronically for status tracking and archiving. A record of an incident is commonly known as a *ticket*. After receiving a *ticket*, the service provider

TABLE 1.1: ITIL stages and their functional focus [3]

Stage	Functional Focus
Service Strategy	Business decisions, such as offering a selection of services to the target customers and/or markets.
Service Design	Identification of service requirements and devises new service offerings as well as changes and improvements to existing ones.
Service Transition	Building and deploying new services, and upgrading existing services.
Service Operation	Fulfilling of user requests, resolving service failures, fixing problems, as well as carrying out routine operational tasks.
Continuous Service Improvement	Continuous improvement of the effectiveness and efficiency of services and processes.

should resolve it, by allocating appropriate resources (e.g. time, personals, budgets, etc). Resolving a ticket promptly and correctly are key business requirements in Service Level Agreements (SLA), which are legally binding contracts agreed between a service provider and their customers. Delay in ticket processing could have detrimental business consequences. According to ITIL, long ticket turnaround time leads to customer dissatisfaction, complains, and/or termination of the contract. For service providers, it costs contractual revenue, as well as the reputation and future business opportunities. If a critical incident is delayed, financial loses could have to borne by the customer and service provider.

Ticket ID	2758
Meta Product	SAP ERP 6.0
Subject	Java is not coming up after the DB restore
Free Text	We are in ERP 6.0 NW 7.0 SP6 running on windows OS 2003 & database version is MSSQL 2008. We did the DB restore for the LE1 system from PE1 system. After finishing ERP shows error message ...

FIGURE 1.1: An example ticket with structured (ID, Meta Product) and unstructured (subject, free text) data.

1.2 Ticket

In ITIL Incident Management, a ticket is a digital record to store information for each incident. As illustrated in an example ticket in Figure 1.1, it contains both structured and unstructured information. The former that associates with a ticket are categorical and meta-information, *e.g.*, predefined type, labels, reporter, and timestamp, etc. The latter is usually user input textual information, describing details regarding the incident, *e.g.*, symptom of the system, steps leading to the failure, any attempt to resolve it, etc. Such information is important for the service provider to understand the situation and the purpose of a resolution. In this research, we focus on the text description of tickets, extracting key product information and model the text for downstream applications. The user-generated nature of tickets poses challenges to analysis and processing. Firstly, unofficial acronyms, synonyms, and homonyms, often appear in technical tickets. The same product can be referred to by different forms of abbreviations and aliases. The variation in version numbers adds to the complexity. Secondly, different expressions are used for the same problem between a user and a support expert. To this end, analyzing linguistic and contextual information could be helpful to figure out the main message from a ticket.

The more products a service provider is offering to its customers, the bigger its support organization tends to be. Typically, technical experts are organized according to their product expertise and geographical locations, forming the basic ticket processing units. In our setting, these groups are known as *expert groups*. An expert group may contain one or more experts with similar skills and expertise. Each group is responsible for a subset of tickets the provider organization may receive. Depending on the business, teams may be split based on different products, hardware and software, database and user interface, etc.

FIGURE 1.2: A high-level overview of the ticket processing workflow.

An expert group can resolve tickets belonging to its expertise domains, but not others. Due to resources limitation, most service providers will only assign one ticket to one expert group at any time. If a ticket is not resolvable by the current group, it should be transferred to another group. In our setting, the decision for such transfer is made locally by the current group. Over some time, ticket movements connect multiple groups from the support organization to form a *Expert Networks*. The groups may be also connected by implicit relations (*e.g.*, organizational hierarchy) or explicit information exchange, such as ticket transfer. The network characteristics are interesting information to be explored in our research. Even though it is common that the personnel or the skill levels within the group may change over time, the overall job scopes at the group level are static over a period of time, for instance, a year. Therefore, in our research, we do not consider the scenarios where expert groups have dynamic skill levels and expertise.

An incident ticket is created to mark the beginning of the ticket process workflow. There are different ways a ticket may be created. An end-user may create a ticket after he/she found out about a multifunctional software. Or the user may contact the helpdesk for help, who will create a ticket to log the conversation and subsequent processes. Following its creation, it is assigned to an expert group for initial processing. The initial assignment is also known as triage in some context, traditionally done by a human expert or an automated system. If the assigned group is unable to resolve the ticket, the ticket will be transferred to another group.

The process repeats until the processing group resolves the ticket by proposing a resolution to the ticket.

Here, we consider the initial assignment and ticket transfer jointly as the *ticket routing problem*. Figure 1.2 illustrates a high-level overview of the ticket processing workflow and the scope of ticket routing. Note that, we do not consider the interaction between the expert group and the end-user, during and after the resolution process. Our objective is to investigate how to improve internal process of ticket routing in the assisting routing evaluation framework.

1.3 Research Problems

Formally, we let a ticket be τ and the service provider has an Expert Network $\mathcal{G}$. The first processing group assigned to τ is A , such that $A: g \in \mathcal{G}$. If A is incapable of solving τ , it must be transferred to another group in the network, *i.e.*, $A \rightarrow B$. If τ is resolved by B , B is the *resolver group*, *i.e.*, $g_\tau^+ = B$. The sequence of groups that have processed the ticket in order, from the first to the current processing group or the resolver (for a resolved ticket), is the *routing path* or *routing sequence*. For a new ticket, the routing path is an empty list, *i.e.*, $S = \emptyset$; for a resolved ticket, the length of routing paths is 1 if the first assigned group is the resolver, *i.e.*, $S_\tau = \{A, B\}$ for ticket τ .

Before any improvement could be done, the first question we need to answer is **how to evaluate a ticket routing system?** Currently, to measure the performance of any routing system, two metrics are commonly used, namely Mean Steps to Resolver (MSTR) and Resolution Rate (RR). For a test ticket set with ground truth routing paths, MSTR measures is simply the average length of routing paths. This can be applied on routing paths by either human or an automatically generated routing system. RR is the ratio system solvable tickets over the entire testset. Even though they are simple to compute, both metrics only consider the true resolver, but ignoring the true routing paths. The ground truth reflects human experts' decision processes. Besides, the MSTR measure is an absolute score, making it hard to compare different systems for a different dataset. RR only considers fully correct routing, without awarding partially correct generated routing paths. We see a need for a new evaluation metric. In Chapter 3, we propose one that considers all

groups in the ground truth routing paths and incorporates distance between each group to the resolver. We also propose an assistive routing framework that takes every human input from ground truth for simulated testings.

Next, we look into the problem **how to identify valuable information from ticket text?**. They are user-generated, domain-specific, and noisy texts that even human experts find them difficult to understand. Specifically, we want to extract the parts that mention specific software products and link them to a predefined product catalog. By annotating and analyzing archived tickets, we observe many homonyms, referring to multiple objects or products. For example, in Figure 1.1, “NW” is an abbreviation of **SAP NetWeaver**. Without mentioning an explicit version, it could mean any version of the **SAP NetWeaver** family. Hence, contextual information is useful in determining the correct product and version from the ticket. Meanwhile, a product may have multiple aliases created by customers. Our analysis of annotated tickets shows that **SAP NetWeaver 7.4** has over 20 different known aliases. Moreover, tickets also contain errors and typos, which are common in user-generated free text. In Chapter 4, we present a solution for extracting and linking software product names, by exploring multiple resources and techniques in designing a rich set of domain-specific features for a linear Named Entity Recognition (NER) paradigm. Experimental results on four datasets saw our proposed model outperform the baseline models. Additionally, we show that despite the increasing popularity of neural network-based NER models, for a highly specific domain like ours, where labels and training instances are relatively scarce, our feature-rich models perform better. The resultant entities could benefit downstream applications, such as ticket routing, by filtering non-technical information from tickets.

Recall that, in our definition of ticket routing, there are two scenarios: initial expert group assignment and inter-group ticket transfer. Previous studies deemed them as two independent problems. The initial assignment, or triage, is commonly solved using text classification models [7–9]; whereas ticket transfer is modeled using probabilistic graphical models [6, 10]. Even for end-to-end solutions, the two problems are solved in stages, using different information and methods. The decoupled models do not leverage the information in their full potential. **Can we combine all the available information in one framework?** In Chapter 5, we propose a joint ticket routing framework to address the two scenarios uniformly. Linear learning-to-rank models are trained using four types of features: ticket,

group, ticket-group, and group-group. Assistive routing performance of models using our unified framework is superior compared to the baseline models. Our experiment also shows the most important features are features modeling relationship between a ticket and a group, *e.g.*, similarity between ticket text and aggregated text of tickets solved by a group. This finding is especially interesting for the next Chapter 6, in which we seek a more complex modeling approach to capture the relationship between a ticket and a group in their text representation.

In recent years, deep neural network models are increasingly popular in many sub-domains of natural language processing. They are especially promising in modeling text similarities, taking advantage of a large amount of training data. Another advantage of deep neural networks is their ability to learn any form of non-linearity using automatic feature generation. Many industrial applications are also motivated to use deep neural network-based solutions. For its close relevancy to our problem, it is sensible to evaluate if we can **leverage deep neural network approach in the context of ticket routing**. In Chapter 6, we adapt deep neural network models and evaluate the performance in ticket routing. Specifically, we proposed DeepRouting, a deep neural network framework for generating feature automatically and computing a relevance score for a ticket and candidate resolver pair. The joint relevance score is a weighted sum of two individual scores, computed in a multi-view architecture. Specifically, in the text-view of both ticket and expert group, we use a classic semantic text matching model to obtain the text relevance. The graph-view outputs expert group representation vector in the same size as the ticket vector for relevance computation. The input is pre-trained embeddings using routing information as network structure, and ticket text as node features. The joint scores are used to rank the candidate resolvers at every routing step. Experimental results show that the proposed approach outperforms baseline models in resolver ranking and assistive routing tasks. Comparative experiments also show that the text has a higher degree of importance than routing information.

1.4 Contributions

By investigating the above-mentioned research problems, we contribute to the industrial and academic research communities in the following ways:

- Firstly, this thesis provides a new perspective to the ticket routing problem, especially in terms of evaluation metric and setup. We consider human routing paths as more informative ground truths, based on which the automatically generated routings are evaluated. The novel assistive routing evaluation framework is a simulated environment to reproduce the points of time when decisions are made. For IT service providers, these could help them to deepen their understanding of the problem and the performance of their organization. For academic researchers, the problem could be an interesting application for state-of-the-art models from fields such as Information Retrieval (IR) and Artificial Intelligence (AI) in a business environment.
- Secondly, we conduct text mining and analysis on the domain-specific tickets, for researchers to understand the effectiveness and challenges in domain-specific information extraction. The state-of-the-art models usually work well on formally written text, such as news articles or Wikipedia. Not only the training resources are abundant, but also the languages used are also cleaner and more standardized. In comparison, user-generated text from online forums or social networks (*e.g.*, Twitter) is similar to support tickets. However, the underlying interaction mechanism and motivation are different. In online forums and social networks, a large majority of user interactions are on a voluntary basis, actions such as replying and following are highly unpredictable. In contrast, in ticket routing, staffs from expert groups are highly motivated and obliged to resolve the issues for their customers.
- Thirdly, we present a solution to ticket routing problem using a unified framework, combining multiple types of features in a real business problem. The framework is inspired by a learning-to-rank framework that was originally proposed for IR applications. The features could incorporate most of, if not all features that were proposed from previous works related to tickets. The frameworks are highly adaptable for other business problems incorporating different information sources. Among all feature types, ticket-group interaction is the most contributive to the task.
- Lastly, we present a deep neural network model for ticket routing problem with automatic feature generation. It computes a relevance score for a ticket and candidate resolver pair. The score comprises two parts, one based on a text view and the other based on graph view. Experimental results show that

FIGURE 1.3: Thesis structure with citation relations.

the proposed approach outperforms baseline models in resolver ranking and assistive routing tasks. We also show that ticket text is the most important information in a multi-view architecture.

1.5 Thesis Organization

The logical linkage among the remaining chapters are illustrated in Figure 1.3. Chapter 2 reviews the related works to ticket processing and routing. Chapter 3 presents a novel evaluation metric and Assistive Routing framework for testing. Chapter 4 details the effort in software product names extraction and linking. Next, a unified routing framework using hand-crafted features is proposed in Chapter 5. Subsequently, a deep neural network routing model is introduced in Chapter 6. Lastly, Chapter 7 concludes the thesis with discussions in future directions.

Chapter 2

Literature Review

2.1 Overview

Research in IT support services are from two main aspects. On one hand, from a business management prospective, there are theoretical works in how to optimize the operational processes, in terms of improving efficiency and reduce cost. An example of such is the publication of ITIL framework. On the other hand, the application perspective of the research concerns methods that can be applied along the process, so the chance to achieve the business goals is increased. Such methods are from the recent advances in various sub-domains of computer science, such as Information Retrieval (IR), Natural Language Processing (NLP), Data Mining (DM), and Machine Learning etc (ML). In this chapter, we focus on the application aspect of the research.

The objective of this review is to understand the research landscape and progress in the effort to automate IT support systems. Specifically, in Section 2.2, we concentrate on investigative, and engineering works in extracting relevant information from support tickets. In Section 2.3, we look at different attempts to use statistical and machine learning methods to classify or cluster support tickets for more efficient processing. Then, in Section 2.4, we discuss the existing ticket routing works. A few works in open-source software bug reporting systems are also covered, due to its proximity to IT related scenario. In such works, bugs reports take similar format and characteristics as IT tickets, formed by a mixture of natural human

language, code snippets, and machine-generated logs. However, the processor assignment in bug management is based on availability and voluntariness, compared to role-based assignment in our problem.

2.2 Ticket Information Extraction

The first step towards building an automated support system is recognition and extraction of important information, or features, from the available data sources. In a typical ticketing system, information extraction can be applied on three areas, namely the *problem* description, the *solutions* provided for closed tickets, and *network* (global) features associated to the system. Text is usually unstructured, written by human users, capturing the behaviors of human processing. Systematic analysis is required before text can be used as features for ML algorithms. Categorical and structured meta-information, such as timestamp, target system, and human/machine assigned labels, are not explicitly discussed here.

2.2.1 Ticket Problem Analysis

Problem description forms the main body of a ticket. For many ticketing systems, they are stored as unstructured text, mainly for human users to read. To enable automatic analytics, the free text needs to be converted to formats that be processed by a computer. Such process is also known as “feature extraction”, in ML terminologies. Past works investigated different text processing pipelines in extracting features from tickets [1, 7, 11–14]. Domain-specific knowledge, such as dictionaries, is incorporated to enhance the quality of extracted features [8, 13]. Topic modeling is another useful tool that is used to generate additional features from text descriptions [15–17]. In addition, some works attempt to infer root cause from the problem text [1, 2, 14, 18, 19]. Through analyzing user input, Gupta *et al.* [20] proposed a system to help the user to provide useful information in terms of reducing overall processing time.

2.2.1.1 Text Processing Pipeline

Unstructured text in IT tickets needs to be converted to an array of numbers before any computer algorithm can process them. Typically, a text processing pipeline consists of the following steps.

Segmentation splits a block of raw text into smaller units. Segmented parts can be words, sentences, and topics. A typical segmentation strategy is to divide by sentence boundary indicated by punctuation. But IT tickets often consist of quite a number of punctuation as part of entity names, which may affect the performance of out-of-shelf sentence segmentation tools. Wei *et al.* [14] implemented a line-based segmentation method, before annotating each with a label to indicate its semantic in the text. Agarwal *et al.* [15] split problem sentences and potential change events from ticket and trained a classification model to predict if a causal relationship exists in a pair.

Tokenization is equivalent to word segmentation, it breaks a string of text to basic word units. White space tokenizer, n-gram tokenizer, and character tokenizer are the most commonly used tokenizers. For example, tri-gram tokens, are used by Acorn *et al.* [21], as features to perform text analysis.

Stop-words removal filters frequent words, like “the”, “this”, and “here”, which are usually ignored in many NLP applications. For ticket processing, some works remove stop-words [8, 15, 22], some against the approach. Zhou *et al.* [23] claimed that some of these terms are useful features for ticket and question quality models. Some works implicitly filter stop-words by keeping only a subset of Part-of-Speech (POS) tags [8, 24]. The effect on performance is mixed and dependent on specific system and applications.

Stemming transforms words in different forms to their root form [25], *e.g.*, “tickets” to “ticket”, “processing” to “process”. Stemming is generally recommended for applications using word features for reducing the number of dimensions, thus mitigating the *curse of dimensionality*. It is also implemented in some ticket processing [22, 24, 26].

POS tagging annotates text with a semantic identifier for each word, *e.g.*, “noun” for “ticket” and “verb” for “submit”. Many researchers apply POS tagging on ticket text, in extraction of key phrases [1, 2, 19, 22, 24, 27]. Predefined POS patterns can be used to extract specific type of words and phrases [1, 2, 8, 27]. Sample patterns are shown in Table 2.1. Some statistically analyze the tags and extract frequent patterns as key phrases [15]. Potharaju *et al.* [1] utilized a set of POS rules defined by Justeson and Katz [4] for technical noun phrases matching. Wang *et al.* [2] improved on the patterns, by considering verbs phrases. Agarwal *et al.* [28] used POS tags together with syntactic features to evaluate ticket quality.

TABLE 2.1: Example POS Patterns by Potharaju *et al.* [1], derieved from technical terminology patterns proposed by Justeson *et al.* [4].

Justeson-Katz Patterns	POS Patterns	Example
Adjective Noun	JJ NN [PS] *	mobile network
Noun Noun	NN[PS]* NN[PS]*	demo phase
Adjective Adjective Noun	JJ JJ NN[PS]*	fast mobile network
Adjective Noun Noun	JJ NN[PS]* NN[PS]*	accessible device logs
Noun Adjective Noun	NN[PS]* JJ NN[PS]*	browser based authentication
Noun Noun Noun	NN[PS]* NN[PS]* NN[PS]*	power supply unit
Noun Preposition Noun	NN[PS]* IN NN[PS]*	device down alert
JJ : Adjective; NN : Singular Noun; NNP : Proper singular noun; NNPS : Proper plural noun; IN : Preposition		

IT tickets are composed of domain-specific vocabularies that usually contain letters, numbers, and symbols. Widely used open domain POS tagger, *i.e.*, Penn Treebank tagger [29], may not work as well as being applied in other text. On the other hand, training a domain-specific tagger requires taxing manual effort to label training data.

After preprocessing, a ticket is converted to *terms* and can be represented in bag-of-words models [30, 31] and n-gram models [21]. Further, term weighting techniques, namely Term Frequency (TF) ([8, 32]) and Term Frequency Inverse Document Frequency (TF-IDF) ([33–36]) are state-of-the-art in many text processing tasks, are applied in ticket processing. TF-IDF is also used for selecting a subset of terms (*e.g.*, 2000) to represent tickets using other weighting schemes [8, 32, 37].

Technical terms used in IT tickets are particularly important for content understanding and analysis. If external dictionaries with domain-specific words are available, they are usually useful [38, 39]. Alternatively, there are statistical means to extract keywords or key phrases [1, 2]. Potharaju *et al.* [1, 2, 15] generated domain-specific key phrases using a combination of statistical analysis on frequently collocated terms and a set of heuristic rules predefined as POS patterns (Table 2.1). The outputs are domain-specific key phrase for downstream applications. In another work, Agarwal *et al.* [15] constructed a domain-specific glossary at run time, using frequent POS tags mining. The difference compared to [15] is that the POS patterns are extracted from ticket corpus, instead of being predefined.

2.2.1.2 Extract information from tickets

Jan *et al.* [13] used a Conditional Random Fields (CRF) based Named Entity Recognition (NER) approach to extract server names from tickets, with lexical and structural features. Using a sample server name dictionary, fuzzy match rules are applied to label training data automatically. The trained classifier can achieve high precision (95.75%) and recall (93.12%) in recognizing server entities in the test set.

Topic modeling is an useful tool in discovering hidden “topics” from a collection of text documents. It analyses the statistics of words with the context it appears in. The output topics group terms that usually appear in the same context together. For example, “printer”, “monitor” and “pc” are likely to appear in a group regarding hardware issues. In a resultant model, a word may appear in multiple topics with different probabilities; and the distribution over words represents a topic. One of the most prominent topic modeling algorithm is Latent Dirichlet Allocation (LDA) [40], used for ticket modeling in [16, 27, 41–43]. It models each ticket as a probability distribution of topics; each topic is a mixture of multiple words. The number of topics needs to be defined for LDA algorithm before training of a model. There is no clear guideline on choosing the number of topics. The results of using LDA-based approaches are data set and application dependent. For example, [41] used topic number 300 for its experiments and 100 topics are used for query expansion to support advanced search applications by Deng *et al.* [42]. A trained LDA model can be used to extract features from tickets. Each ticket is represented by topic distributions and ticket similarity can be computed using Kullback–Leibler

divergence or cosine similarity. However, its effectiveness when applying to short text is not always convincing [44]. Short text corpus, such as Tweets from Twitter, are sparse concerning word co-occurrences. Besides, the number of topics mentioned in short text is rather homogeneous. Topic layer for documents may not be used for short text. We see some similarities between IT tickets and tweets. Tickets are often short, except logs and codes; both are noisy with irregular characters. The actual performance for applying LDA in ticket processing is rather system dependent.

Besides LDA, there are other topic modeling algorithms, such as Latent Semantic Analysis (LSA) [45] and probabilistic Latent Semantic Analysis (pLSA) [46]. Both are commonly used in NLP research and sometimes outperforming the others in some domain data set. But as they are rarely used in previous researches using IT tickets, we do not discuss them in details. The interested readers may refer to their respective literature. A limitation of topic modeling methods in common is that the model is trained on a set of documents that are supposed to be from the same domain. Miao *et al.* [47] argued that the collection of ticket text and solution text may have different terms distribution, thus should be modeled separately. To enable topic modeling for dual data set co-training, the authors proposed a generative Latent Association Analysis (LAA) model to learn the structures of topic and model inter-document correlations, using document pairs. Subsequently, based on LAA-based association score, a ranking scheme is proposed and tested on an IT-Change (24,317 pairs) and an IT-Solution (19,696 pairs) data set for document retrieval task. In the evaluation, the effectiveness semantic topics extractions are demonstrated. During testing, the LAA-based approach yields the lowest average ranking for target documents, compared to baseline methods. However, as the model requires a pair of question and answer for training and inference, the computational cost is high for search for the most appropriate answer from a large candidate space.

2.2.1.3 Issues and Root Cause Analysis

In ticket processing, one of the first steps a human expert would do, is reading a problem description to locate and understand the issues. The parts of the text mentioning the particular problem need to be identified. Indicative keywords such

as “crashed”, “slow,” and “error” are often used in the identification of the problems. Next, with the issues extracted, the cause of the root problem needs to be determined or inferred. Sometimes the reason is obvious and extractable from ticket text, *e.g.*, an action that triggers the error. Other times, it requires logical reasoning to diagnose, based on knowledge and experience, or with the help of external references.

Shimpi *et al.* [48] concluded the following challenges when extracting the issues from tickets systematically and automatically. Firstly, the same problem can be described differently, especially by customers with different levels of expertise in the problem domain. Secondly, the definitions and categorizations of the issues are different in each company. Customizations are necessary to adapt to different service catalogs and IT environments. Thirdly, the technical terms found in tickets are domain and application specific. For instance, the words like “report” have different meanings when they are used for different software. Lastly, the language employed in tickets are user-generated, which is often noisy and error prone. Since conventional NLP tools are usually trained on the open-domain corpora, the performance from applying them on tickets may be unsatisfactory. The authors then proposed two pipelines for issue extraction, using keyword-discovery-based and clustering-based methods, from user-generated and machine-generated tickets, respectively. During an evaluation on 3,854 test tickets, the approach can successfully extract issues in more than 90% of the cases.

A group of works is dedicated to root cause analysis in ticket processing. Liu *et al.* [49] grouped and summarized common features from a group of archived similar tickets, before letting human experts identify root causes for the collection. Liu *et al.* [50] proposed an analysis architecture for root cause analysis using stream processing and MapReduce framework. Giurgiu *et al.* [18] analyzed the tickets processing time, with respect to the root causes and other features. Root causes are stored as categorical features, instead of free text. The results supported the hypothesis that a ticket’s root cause has the highest correlation with the time-lapse from ticket open to its closure.

Wei *et al.* [14] trained a CRF-based sentence tagger to assign semantic labels to ticket text, using formatting features. A total of 21 labels are defined to indicate a sentence’s meaning in the text. Among the labels, some are specifically about the nature and cause of the problem, *i.e.*, “ProblemType”, “ProblemContext”,

FIGURE 2.1: Using *ClassTagger* to label and transform sample problem [1, 2]

“ProblemDescription”, and “RootCause”. Experimental results show more than 80% cross validation accuracy in the test set.

Potharaju *et al.* [1] and Wang *et al.* [2] used domain-specific facts and ontologies extracted from closed tickets to implement a *ClassTagger*. It is then used to label a block of problem text for its symptom, trigger, and attempted remedies. Then, the labeled text is transformed into a structured format for further analysis. In their work, it is used for similarity comparison. Figure 2.1 demonstrates how *ClassTagger* is used for problem analysis.

Ponnalagu *et al.* [19] implemented an online root cause evaluation for software systems, by matching between problem texts and patterns generated from pre-defined software topologies. The quality of these techniques depends on the quality of the underlying knowledge graph and labeled training data. Both require a considerable amount of human efforts to label and curate data.

2.2.2 Ticket Solution Analysis

In modern ticketing systems, closed tickets with solutions are stored for further reference. It can be in a plain text format, with solution appended to the problem description. Some create separate solution documents for archive and to be reused later. Past solutions are the primary educational and referential resources for human experts when processing customers' tickets. Several works explore methods to extract useful information from solutions and use them for downstream applications, *e.g.*, domain-specific search [36, 42] and automatic resolution [1, 15, 51].

2.2.2.1 Solution recommendation

Instructional phrases that contain actions or steps are the interest of extraction, they can be further correlated to form the basis towards automatic solution generation. Heuristic POS patterns and filters are designed to extract phrases from ticket solutions that are clearly defined actions [1, 2, 15, 52]. Aggarwal *et al.* [52] used the phrases extracted to construct summary from a collection of solutions.

Potharaju *et al.* [36] focused on extracting configuration settings in solution documents for automatic fixing of issues about misconfiguration. A set of complex NLP pipelines and handcrafted rules are applied to a large collection of domain-specific for indexing. NER is performed using the same technique developed in [1]. Online evaluation of the system shows high accuracy (80%-97.5%) in recommending solution documents to a misconfiguration issue, taking system snapshots with key parameter values as input to query the knowledge base.

Watanabe *et al.* [53] developed a framework that automatically generated a workflow graph given a set of tickets as input. Firstly, a supervised model is trained to differentiate the relevancy of a sentence in the ticket. Irrelevant ones are ignored from further analysis. Secondly, a novel dynamic programming trick is used to extract and align action descriptions from unstructured text. Lastly, sentences with the same action are aggregated in a normalization step.

Zhou *et al.* [23] proposed an integrated ticket analysis framework. One of the functionalities of the frame work is to score a ticket for its quality, using a regression model with handcrafted features. Next, a deep neural network model is trained to rank a ticket and solution pair. The ranking model considers both text matching

score and the ticket quality score, when recommending a solution for a ticket. Besides, the last hidden layer before the neural network output after training is also effective for ticket clustering and classification.

Wang *et al.* [54] formulated the solution recommendation as a contextual bandit problem in reinforcement learning and proposed a hierarchical multi-armed bandit algorithms. In the problem setting, the solutions for a problem are procedures that can be automatically executed one after another, until the right solution is found. This requires the solutions are organized in a hierarchical structure by domain experts. The algorithm took advantage of the hierarchy and generate a sequence of solutions to the problem. The proposed algorithm is shown to outperform conventional bandit algorithms. In addition, it is also effective for cold-start problem, in which no similar solution was given before for a new type of incident.

2.2.2.2 Solution quality scoring

One of the challenges many automated systems are facing is the data quality. Customers are composing ticketing using non-standard language and expressions; human experts working under time pressure tend to use irregular language with shorthand expressions and errors. Low-quality tickets and solutions impact the performance of downstream applications, *e.g.*, solution retrieval, and recommendation. Majumdar *et al.* [55] designed a ticket quality score, based on the number of technical information exists in the sentences. To help support experts in writing reasonable resolutions text, Agarwal *et al.* [28] proposed an automatic quality assessment tool to score a solution while the human expert is typing. Besides scoring, it also suggests missing elements from the solution. In the proposed system, ticket quality is evaluated by its language, presentation, and business relevance. A multinomial logistic regression model is trained using human annotated tickets. Experiment results show significant improvement over Support Vector Machine(SVM)-based classification model using only weighted word features, from 68.5% to 88.2%. Zhou *et al.* [23] used human labeled quality for a question and solution pair, to train a regression model. The trained model is used to generate more ground truth labels, with reasonable confidence, for a deep neural network based ranking model. Both surface and semantic features are used for the regression model; while the deep neural network model takes a concatenated word embedding and character embedding as input. In addition to unstructured text in tickets, [56] also include

categorical fields selected by users in their ticket quality measurement. Nonetheless, only keywords matching to a small predefined dictionary (29 words) attribute to the final score. All of these models are supervised models relying on labeled data for training. Though the quality of a solution is related to its length, format, and choice of words, the models are not capable of capturing the “usefulness” of a solution to a ticket, which requires more semantic understanding and association on the components in ticket and solution.

2.2.3 Summary

The first step in any IT ticket-related analysis and application is to identify and extract useful information from tickets and their solutions. Linguistic features and lexicon features can be extracted using a combination of text mining and NLP pipelines. However, because IT tickets are technical domain-specific, some tools may not work out-of-the-shelf. Therefore, more domain-specific solutions are often required for individual problems. In the next chapters, we incorporate some of the above-mentioned techniques in the later chapters.

2.3 Ticket Clustering and Classification

Clustering and classification are classic machine learning problems that are well-researched and applied for many domain problems. In IT support ticketing systems, there exist similarities among tickets. The latent patterns can be explored leveraging a myriad of extracted features with different similarity measures. Grouping similar tickets automatically is useful for data management, problem identification [8, 32] and ticket routing [33–35]. In this section, we focus on previous clustering and classification works applied on IT tickets. In particular, we summarize the works by features, similarity measure and clustering/classification algorithms applied.

TABLE 2.2: Summary of literature on ticket clustering.

Authors	Algorithm
Roy <i>et al.</i> [22]	k -means
Li <i>et al.</i> [57]	Hierarchical clustering
Medem <i>et al.</i> [26]	Hierarchical clustering with average-linkage
Lin <i>et al.</i> [58]	Hierarchical clustering with complete-linkage
Botezatu <i>et al.</i> [24]	Hierarchical, fuzzy k -means
Marcu <i>et al.</i> [59]	Rule-based, component mapping, constraint adaptive probing

2.3.1 Ticket clustering

Clustering is an unsupervised process to group instances with common attributes together. Given a collection of tickets and a similarity metric, clustering produces several distinctive groups; each group only contains similar tickets. A summary of the clustering is shown in Table 2.2. By clustering tickets with similar features, it is believed that human experts are able to explore archives more efficiently. In their analysis of IT service tickets, Li *et al.* [57] applied a hierarchical clustering on tickets and identified the prevalent clusters for further investigation. Both [26] and [58] used hierarchical clustering, with average-link and complete-link for cluster similarities, respectively.

Medem *et al.* [26] proposed TroubleMiner, a system used hierarchical clustering technique to discover the structures on error topics from a collection of tickets. Problem texts are represented using bag-of-words features, weighted by document frequency. The vectorized tickets are processed by hierarchical clustering, with a combination of two distances metrics (Euclidean and cosine) and average-link cluster similarity. Post-processing is performed to transform an unlabeled binary tree, which is the output from hierarchical clustering, to a labeled N -ary tree, by merging nodes that have common keywords. The usability of TroubleMiner is demonstrated on two relatively small datasets, with 1,676 and 147 tickets, respectively. However, their work concluded without an evaluation strategy, due to lack of ground truth.

Roy *et al.* [22] employed a medoid-based k -means clustering algorithm to cluster tickets. A hybrid distance score is designed, using Jaccard distance and cosine distance for fixed categorical features and free text features, respectively. After clustering, a set of phrases is extracted to represent each cluster. One drawback of

k -means algorithm is that the number of clusters k needs to be predefined. Besides, the resultant clusters are not labeled, therefore not intuitively interpretable.

Botezatu *et al.* [24] proposed a “multi-view” ticket clustering technique to improve ticket dispatching. The authors considered both the incidents and support experts’ past performance for clustering. In particular, the technique grouped tickets with similar issues and estimated processing time. A ticket similarity matrix and a sparse support expert divergence matrix are constructed and used jointly to derive the similarity of tickets. At clustering step, a combination of three algorithms is employed, incorporating both hard and soft clustering. Following clustering, experts can be profiled by their performance for each cluster of tickets with different difficulty levels, in terms of estimated processing time. The effectiveness of the proposed method is evaluated by the homogeneity of experts performance in a group and processing time reduced for dispatching. Several components in the methods are computationally intensive. As experiments only apply it on small sets of tickets (3696, 2373, and 1916 for three sets).

To generate ticket clusters with quality description, *Lingo* [60] algorithm was used in [15, 27, 27, 61, 62]. Unlike other search result clustering methods, where cluster descriptions are generated after grouping documents text contents, Lingo first generated frequent phrases as labels to describe clusters, using Singular Value Decomposition (SVD) on term-document matrix. After the number of clusters and descriptions are defined, each returned ticket was matched with each cluster label and added to a group if the matching score exceeds a threshold.

To correlate tickets with latent connections together in an unsupervised manner, Marcu *et al.* [59] formulated a correlation model. The model consisted of three filtering parts. The first is a rule-based filter, processing on similarities between service and resources categories. The second part maps the new problem with known error components in the past tickets, enabling “topological comparison”. The last part is temporal correlation using *constraint adaptive probing* to infer required resources for correlation is not known from monitoring the system. Experiments show promising results in correlating tickets with proposed methods.

2.3.2 Binary classification

A straight-forward use-case for binary classification is to differentiate the “complex” tickets from the “simple” ones [33–35, 63]. Such models are relatively easier to train and often have reasonable results. However, the complexity of a ticket is hard to be measured. Frequent used proxies for ticket complexity are processing time [24], number of updates [63], and routing frequencies [33–35].

Tang *et al.* [39] attempted to reduce the number of false positive tickets, by classifying tickets into false and real system alerts. To mitigate the data imbalance and labeled data scarcity, the authors used a domain dictionary to score and label a set of positive training data. The score is the maximum number of words matched between the ticket in question and vocabularies in each predefined situation. Each of the words in the dictionary is also included as a positive sample. In the next step, a SVM-based model is trained using the training set. The two baseline used for comparison are using word-match and random selection for training data generation. The F1 score using selective labeling is higher than baselines until the number of training tickets is large. However, no explanation is given as for what reason a random labeling strategy would achieve such a level of accuracy at all.

2.3.3 Multi-class classification

Diao *et al.* [64] proposed a rule-based system to perform multi-class classification on IT tickets. The system incorporates rules defined by experts and a crowdsourcing platform for a large number of users to validate the rules. Rules are defined, scored, and revised in an iterative process until the rules yield satisfactory coverage and accuracy on test data. Experimental studies show 72% coverage and 88% accuracy for training set and 58% coverage and 81% accuracy for testing set. It is observed that the performance of the supervised method is worse compared to the rule-based methods, even with more training data. The authors attributed such finding to the difficulties for a supervised algorithm to identify the right features over large features space, *e.g.*, terms and phrases.

Dasgupta *et al.* [65] classified unstructured tickets into seventeen error categories. The authors first used four heuristic rules to expand tickets with their corresponding system events. Since the events contain more indicative information, the classification on the enriched tickets outperforms the baseline model from [7]. Both are using a selected discriminative words as features. Giurgiu *et al.* [66] integrated a spherical k -means algorithm to cluster tickets into 20 groups, using pairwise cosine similarity, in their multi-step system. The silhouette measure [67] is then used to evaluate the homogeneity of resultant clusters. In the following step, each cluster is labeled by key phrases extracted from its member tickets.

Kadar *et al.* [68] used a combination of information retrieval and machine learning methods in classifying change request tickets into one of the 200 fine-grained activities. Supervised learning depends on a large number of labeled training samples. To improve the learning efficiency, a pool-based active learning [69] is used. Further, a weakly-supervised learning framework relies on word tags is proposed for evaluation. Experiments are carried out on two sets of 1,317 tickets from two customer accounts, with 54 and 88 classes, respectively. The results show the weakly-supervised method outperforms the IR-based approach in terms of accuracy@ N , but not as good as the supervised model. Consider it only uses 200 labeled words for training, the result is promising for further investigation.

Active learning was also explored by Wang *et al.* [70]. The authors first identified two main challenges in classifying IT tickets. Firstly, the large diversity of tickets from different classes makes general similarity measures difficult to apply. Second, high quality labeled training data is hard to collect. To overcome them, a combined model consisting Discriminative Neighborhood Metric Learning (DNML) and Active Learning with Median Selection (ALMS) were proposed. For 2,193 tickets, by labeling only 20% of data as the training set, the proposed approach can classify the rest into 15 classes with more than 83% accuracy.

Maksai *et al.* [17] implemented a two-stage approach to classify IT tickets into five types, with a few labeled training data. Firstly, a hierarchical clustering is applied to produce clusters that capture the main type of tickets in their corresponding clusters. This step involves a graph clustering layer, and a LDA-based second-layer clustering before ticket types are assigned to the output clusters. The next stage attempts to fine-tune for the classification for each ticket type individually, either by using hierarchical clustering (similar to the first stage), or active learning

techniques. Though the authors claimed the proposed framework works best for “very small classes”, we find the experiment results to be inconclusive. Out of four classes evaluated, the new approach outperforms other baselines in two classes. Even in one of two winning classes, there is no clear trend to show the method is always better. In general, the results are rather mixed.

Machine learning models usually lack interpretability. Explicit rules can hardly be extracted directly from the models. In contrast, the Formal Concept Analysis (FCA) based methods approached by Borchmann *et al.* [71] are able to establish relationships between features and component classes. The authors use these algorithms to classify bug tickets to fault areas, by retrieving a list of past tickets that likely have the same issue. Similarly, Wang [72] applied FCA to derive rules for resolver determination from historical tickets.

2.3.4 Hierarchical Multi-class classification

In many IT management systems, problem categories are organized in hierarchical structures. Categorizing a ticket to multiple classes in the hierarchy automatically and accurately is a challenging problem. Zeng *et al.* [8, 32] addressed this problem by optimizing a problem-specific loss function, the Contextual Hierarchical loss function (CH-loss).

The authors first analyzed the possible error patterns in misclassified parent-child nodes prediction, with respect to a ground truth pair. Four misclassification cases are combined in formulating CH-loss mathematically. It proved to be a generalization of previous loss functions used for hierarchical classification problems, *i.e.*, Hamming-loss [73, 74], H-loss [75], and HMC-loss [76]. None of the previous cost functions considers hierarchical information for error classes.

By applying Bayes decision rules, the optimal solution for CH-loss can be computed for individual class. For each class in the hierarchy, a marginal probability is estimated using separate classifiers trained in parallel. To incorporate domain-specific prior knowledge, a *sub-product* based algorithm is designed to augment the derived marginal probabilities. Lastly, a novel greedy algorithm, *GLabel* is proposed to assign class labels to a ticket greedily. Compared to other algorithms for hierarchical classifications, *GLabel* does not require the number of labels to

TABLE 2.3: Summary of Methods used in Ticket Classification.

Method	Used by
Rule-based	[64]
SVM	[7, 39, 65, 77]
NB	[33–35, 63]
k -means	[66]
Active Learning	[68, 70]
Maximum Entropy	[70]
Random forest	[78]
FCA	[71, 72]
Customized Classifier	[8, 32]

be defined as a parameter. Experiments show its superior performance in both effectiveness and efficiency compared to strong baselines.

However, the proposed approach has two limitations. Firstly, it only works for the hierarchical tree structure. In reality, entity organization could be in multiple trees or directed acyclic graph (DAG). Secondly, prior knowledge integration is a separate step after marginal probabilities' estimation and only took binary input vectors.

2.3.5 Summary

Classification is a typical machine learning task. The performance of a classifier depends on multiple factors. Especially in real-world datasets where the labels are unbalanced, training a model with reasonable performance takes in depth understanding of the business scenarios, as well as trial and error. Besides the technical methods, applications discussed in this Section are different in their labels, or target for prediction. Besides the resolver, other information are used as lables, *e.g.*, priority, complexity, existing solution, and resolution time, etc. The choice of lables were decided by the authors based on their understanding of the problem and the information available to them. In the following chapters, we will refer to some of these methods when designing our routing frameworks. However, it is worth noting that different methods applied in different datasets are not comparable. What worked in other cases are not guaranteed to be working in our problem, even when the prediction targets are similar. In the later chapters, we use some of these models as baseline to compare with our proposed models.

2.4 Ticket Routing

Consider a support ticketing system with multiple *expert groups*, each has one or more support engineers and is responsible for a specific technical domain. A ticket can be transferred from a group to another in the system for processing. Such process is called “ticket routing”. For a resolved ticket in the system, the last expert group processed it is the *resolver*¹. Ticket routing problem investigates solutions that can lead a ticket to its most likely resolver with minimal steps. It is also known as “dispatching”, “resolver recommendation”, or “resolver selection” problem in some applications. In software maintenance, the relationship between bugs and developers as analogous to that of tickets and expert groups in our context. The ticket routing problem corresponds to “bug assignment” or “bug triage” problem in this domain.

In ticket processing work-flow, the potential resolver needs to be selected in two cases. In the first case, when a new ticket arrives, a dispatcher assigns it to the most likely resolver for processing. In the other case, when the processing group is not capable of solving the ticket and needs to route the ticket to the next group. Both cases have the same objective of finding the most likely resolver group; both can be framed as ticket routing problems.

In practice, ticket routing problem has the following challenges:

- The decision-making process depends on local knowledge the processor has on the ticket and global knowledge about other groups.
- Decision made by human experts takes time and is error-prone.
- Routing directly to resolver is not possible since intermediate groups are necessary, in some cases.

Approaches using different information and objectives are proposed in the attempt to automate ticket routing process.

¹Note that “resolver” used here can refer to an individual expert and a group or experts

2.4.1 Ticket Routing with Classification

By only considering the text information in a ticket, the ticket routing can be projected to a text classification problem, or specifically, a multi-class classification problem. Ticket and resolver pairs in historical data are used to train classifiers. Predicted resolver made in test data is evaluated using standard classification metrics, *e.g.*, precision, and recall. For example, Miao *et al.* [6] used a NB classifier for Ranked Resolver routing method. Goby *et al.* [43] applied a random forest classifier to classify tickets into five groups. Other classification algorithms (*e.g.*, support vector machine, decision tree) and heuristic rules were used to improve ticket routing effectiveness, which is measured by standard metrics such as precision, recall, F-score, and accuracy. Instead of reducing MSTR, their objective for classification is to maximize these metrics [7, 11, 12].

Di Lucca *et al.* [11] implemented different classification methods to maintenance tickets in an attempt to automatically assign them to one of eight processing groups, each focusing on a particular type of issues. Both problem description and meta-data from tickets are used. Experimental results showed probabilistic models outperforming other classifiers, *e.g.*, k-nearest neighbor, SVM and CART. It was also shown that classification performance could always be improved with more training data. However, classification performance relying on a few keywords was inconsistent, varying with keyword distributions for the whole ticket set.

Agarwal *et al.* [7] built an automated system to dispatch a ticket to the correct resolver at the beginning of ticket processing named “SmartDispatch”. The central proposition of the authors was that ticket routing is complicated to be fully automated. Some tickets are easy and can be automatically dispatched. For tickets that are not straight forward, human dispatch is required. A multi-step framework was designed. In the first step, the system attempted to classify a ticket to a group using a SVM classifier with text features. Auto dispatch was only executed if the confidence score from the classifier is higher than a threshold. Experiments showed high precision (69% - 81%) at this first step. If the first classifier failed to predict a group with high confidence, the system would send the ticket to an available human dispatcher. Meanwhile, a recommended list of 1 to 5 groups was made to the human dispatcher for reference. If no human dispatcher is available, a “limited broadcast” is activated to send the ticket to multiple expert groups simultaneously.

Compared to the other works [5, 6], data set used in [7] contains fewer number of groups, having 25, 25 and 79 for three datasets respectively. In contrast, [6] used datasets with 847, 638 tickets and 301 groups. Besides, data analysis showed a majority of tickets (55%, 25% and 62% in 3 datasets) can be classified in the first step with 0.9 or more confidence score. Since different data sampling criteria were applied, the results of these works can not be compared directly.

From a practical perspective, the usability of the “limited broadcast” is questionable due to the duplicate efforts spent by multiple groups. As experts in the group is likely to spend time and efforts looking into the problem before deciding if it is solvable. More resources are invested if multiple groups are processing a ticket at the same time. If one of the groups has successfully resolved the issue, the other groups will have to discontinue and move to the next ticket.

Gupta *et al.* [12] used a Configuration Management Database (CMDB) as a reference to correlate a ticket with past tickets. CMDB is a repository storing configuration items (CI) which are domain specific terms that are used for problem identification and diagnosis. Relevant CIs are assigned to a ticket using ML methods as a classification problem. In the next step, resolver classification is performed with Jaccard similarity using CI features. The similar concept is used in Automated Incident Management using Historical Information (AIM-HI) system [79] which used a combination of supervised ML algorithms to predict resolver based on Problem Domain Signatures (PDS) pre-defined in a problem symptom hierarchy. The purpose of assignment is effectively the same as extracting features from a ticket and representing it in a vector space. Resolver groups associated to the same PDS in historical tickets are used to select candidate resolver groups for a new ticket. Then an expert level recommendation is performed based on individual’s problem processing history and the performance is evaluated by classification accuracy.

The main issue with their work is the ambiguity about PDS definition. It is apparent that PDS’s are the key to the effectiveness and performance of such system. However, it is also understandable that authors may deliberately avoid revealing more proprietary information about the system. As we discussed earlier in the chapter, industrial projects may not be as comprehensive regarding description as academic works. Proprietary data and methods may be the competitive advantage to other competitors.

Overall, many classification-based routing methods ignore intermediate groups. Some of them are dispatcher groups which are crucial for problem resolution [5] and should not be ignored.

2.4.1.1 Ticket Routing with Time Constraint

Besides routing correctness and accuracy, resolving a ticket within a time limit is another requirement for a routing system to be deployable in real production environment. In production systems, the time limit for a ticket to be resolved is dictated by Service Level Agreement (SLA) between the service provider and the customer. Failing to fulfill such condition would result in customer dissatisfaction and financial loss for the business. Fulfillment of SLA takes one of two states. If total time spent by a routing sequence is less than the time target, SLA target is *met*; otherwise it is *breached*. An important point to note is that the total time of a routing is not directly dependent on its length.

Unlike shortest path traversal problem, the time cost for a longer path may be less compared to the path with a shorter path. Neither network-based nor classification-based routing consider how long a ticket may be processed by a group, nor by the path as a whole. To incorporate time constraint when predicting routing path to the resolver for a ticket, Moharreri *et al.* [33–35] argued that routing paths for closed tickets that have complied with SLA time should be recommended to a new ticket to ensure time constraint is met.

In their initial analysis [33], 81% of the past tickets used some 5.5% of all routing patterns. Based on these observations, the authors designed a two-level classification framework. The first-level binary classifier is trained on features extracted from ticket contents using NLP techniques, with binary labels to indicate the *routine-ness* of a ticket. The second-level multi-class classifier is trained on pre-existed frequent ticket routing sequences as labels. Both classifiers training are off-line operations. In inference steps, it first differentiates if a ticket should be processed by automated routing or human routing. Then, a pre-existing routing path is recommended to the former group. Experiments show 29% reduction in SLA breach under the proposed framework.

To address the inflexibility due to reusing previous paths as candidates for recommendation, a “flexible” mode is introduced to cover compatible routing paths

which can resolve the same ticket as the ground truth path. However, the definition of such compatible set relies on human expert’s manual effort. In an extended work [35], the performance of the framework is analyzed in the context of an Expert Network. The results show very high recall (99.8%) for the first level classifier to find tickets which can be resolved automatically. Meanwhile, a majority of tickets (87%) that breached SLA are routed to human processors by the first level classifier. Both results are evidence of the effectiveness of such design. However, the false positive rate is high, leaving room for future improvement.

Subsequent work [34] investigated the weakness of the framework, specifically on the relatively low performance of predicting Expect Time-to-Resolver (ETTR) with respect to Actual Time-to-Resolver (ATTR). The authors categorize tickets into six groups, based on comparison results between ETTR predicted and ATTR achieved by recommended routing paths. Special attention is paid to the group with higher ETTR than ATTR. Both have met SLA target. Using probabilistic language models, each ticket in this group is accessed by measuring how much its content differs from the model built by tickets with the same priority and routing paths, measured by normalized cross entropy. Next, the measured result is correlated with the difference in TTR in squared error, showing only weak correlation. Further analysis shows that the performance of the original model is restricted by the static features.

By recommending a sequence of groups as a “super group”, it includes necessary intermediate groups in routing and guarantees that SLA time is likely to be met, assuming the time for a group to solve similar question is fixed. For it to be useful, some groups must always be applied to some problems sequentially, as in a pipeline. This is hardly applicable in IT support, for the following reasons.

- Recommending a path of groups does not consider the intermediate groups that are wrongly assigned.
- It assumes the combined time for a sequence to process a group of similar tickets is fixed. However, the actual processing time is affected by individual’s capability and workload, which is not captured in the model.
- Selection of the next group during routing is more likely to be made by current processing group, with or without computer aid.

- It lacks flexibility as only sequences previous seen are candidates for recommendation. Even though “flexible” models are used to complement candidates, it probably requires more human effort than the automation could ever save.

Besides routing with a time constraint, Sun *et al.* [31] formulated a task to predict TTR for a ticket if assigned by an expert group, using group’s expertise representation. The authors observed two routing decision-making patterns from data analysis. In one scenario, an expert route a ticket entirely based on his/her connection, without considering the ticket in question; in the other, routing is made to take into consideration the task described in the ticket. A generative model is then proposed to simulate routing decisions made by human experts, incorporating these two strategies. The joint model exhibits superiority over several learning baselines during experiments in terms of log-loss, step-loss, and mean absolute error. However, the model does not directly produce routing options. It has to be used in couple with other methods to be useful in real applications.

2.4.2 Ticket Routing with Network Analysis

2.4.2.1 Expert network analysis

The notion of *Expert Network* was first seen in the work of Shao *et al.* [5], then adopted by [6, 80, 81]. The same concept is also referred to as *Collaborative Network* [30, 31, 82], *Enterprise Network*, or *Resolution Network*. We use the term “expert network” to differentiate this type of network involved in this study from other types of networks with identical names but for different purposes, *e.g.*, “Collaborative Network” (*i.e.*, social networks with collective efforts [83] and co-author networks [84]).

The expert network can be defined as a homogeneous network. Each node is an *expert group*. Each group consists of one or more human experts. Typically, each group is responsible for a specific technical domain. A directed edge between two nodes models ticket routing flow from a sender group to a receiver group. Weights may be assigned to the edges to indicate the probability of routing, which is either mined from historical routing records or predefined according to business requirements.

As a variation of a social network, it is differentiable from conventional Facebook or Twitter networks in terms of modeling and motivation for information flows [82]. In conventional social networks, interpersonal interactions are relationship and interest-oriented. Nodes are connected by pre-existing family links, friendships, and preferences. Typical information transferred are influences and messages. In an expert network, communications between groups are task-oriented. Individuals are grouped by functions, and/or by organizational structures. An open ticket submitted to the network is investigated by a group that it is assigned to. If the processing group cannot resolve it, it can contribute to the task completion by passing it to another group who can. Optionally, additional information or attempted solution can be passed along with the ticket. Such information flow depends on an individual's skills, problem-solving capability, and awareness of others' skill sets, less so on the static links existing between entities [85]. This characteristic of expert networks is important to understand the ticket routing problem.

Network analysis concerns overall information flow and patterns as a result of network activities. Transfer sequences across the multiple groups are analyzed and used to characterize the groups, and to make the prediction on their behaviors [5, 6, 31, 33, 86]. It is important to understand the global features to make system-wide performance evaluation and improvement. Statistical methods such as Markov modeling [5, 10, 80], generative modeling [6, 31, 31] and sequence pattern mining techniques are used [5, 10, 33, 80]. Similarly, Bartolini *et al.* [87, 88, 88] present an analytical system which uses a stochastic transition matrix to model ticket routing behaviors among support groups. The proposed system is employed to analyze and improve the performance of IT support systems.

2.4.2.2 Using routing path only

Shao *et al.* [5] analyzed historical routing logs with frequent pattern mining technique [89] and construct a directed graph $\mathcal{G}$ to simulate ticket transfer patterns among groups, *i.e.*, $g \in \mathcal{G}$ are vertices. Edge weights are ticket transfer probability, estimated by the ratio of transfer made after being processed by the sender group, *i.e.*, $P(g_b|g_a) = N(g_b \rightarrow g_a)/N(g_a)$. The target is the resolver group g_r .

TABLE 2.4: Resolver Search Algorithms (FM: First-order Memoryless search, FMS: First-order Multiple active State search, VMS: Variable-order Multiple active State search [5])

FM

$$g_r = \underset{g}{\operatorname{argmin}} P(g|g_{\text{current}}), \forall g \in \mathcal{G} \setminus \mathcal{G}_{\text{visited}} \quad (2.1)$$

FMS

$$g_r = \underset{g}{\operatorname{argmin}} P(g|g_v), \forall g \in \mathcal{G} \setminus \mathcal{G}_{\text{visited}}, g_v \in \mathcal{G}_{\text{visited}} \quad (2.2)$$

VMS

$$g_r = \underset{g}{\operatorname{argmin}} P(g|S_k), \forall g \in \mathcal{G} \setminus \mathcal{G}_{\text{visited}}, S_k \subseteq \mathcal{G}_{\text{visited}} \quad (2.3)$$

Searching for resolver is a greedy graph traversal operation, following the path with the highest conditional probabilities given the current step and list non-resolver visited groups. Table 2.4 lists three search algorithms which were proposed for resolver research in [5]. Both Equation 2.1 and 2.2 are first-order Markov models, using only one group as a condition to compute the conditional probability. Equation 2.1 selects the most probable group given the current group. While 2.2 iterates through the visited list as a condition to select the most probable next step. Equation 2.3 expands the model to higher-orders, by including subsets of the visited list as given conditions for computing the probability. Experiments on 1.4 million tickets from a real production system show the average numbers of routing steps are reduced compared to human routing results, especially for those tickets that took long human routing steps. Among the three algorithms, VMS yields the best results. It is also adopted by later works [80, 85] as a strong baseline method.

The significance of *et al.* [5] is two folded. Firstly, it set a precedent for modern research in ticket routing problem, using group transfer data mined from the network. Secondly, it uses Mean Steps to Resolver (MSTR) to measure the performance of proposed system, which becomes a standard metric adopted by subsequent works [6, 80–82, 85]. However, as we will discuss in the next chapter, the metric is insufficient to evaluate routing systems in many cases.

The authors acknowledge the absence of ticket contents in the model is the main

TABLE 2.5: Generative models for ticket routing (RM: Resolution Model, TM: Transfer Model, ONM: Optimized Network Model [6])

RM	$P(t g_i) \propto \prod_{w_k \in t} P(w_k g_i)^{\#(w_k, t)} \quad (2.4)$ $P(w_k g_i) = \frac{\#(w_k, \mathcal{T}_i)}{\sum_{w_j \in \mathcal{W}} \#(w_j, \mathcal{T}_i)}$
TM	$P(w_k e_{ij}) = \frac{\#(w_k, \mathcal{T}_{ij})}{\sum_{w_\ell \in \mathcal{W}} \#(w_\ell, \mathcal{T}_{ij})} \quad (2.5)$
ONM	$P(g_j t, g_i) = \frac{P(t e_{ij}P(g_j g_i))}{Z(t, g_i)} \quad (2.6)$ $= \frac{\prod_{w_k \in t} P(w_k g_i)^{\#(w_k, t)} P(g_j g_i)}{Z(t, g_i)}$

shortcoming of their work. The second limitation is that locally optimized greedy search methods may not produce globally optimal routing paths as global network information is not considered during graph traversal. Another issue is that the count-based probability may suffer from small training data size. With limited historical ticket transfer records, the constructed network is not fully connected. As a result, the candidate routing destinations are limited.

2.4.2.3 Using routing path and ticket text

Extending from [5], two separate works were proposed, using different methods to integrate text information from tickets into network models.

Sun *et al.* [80] used a two-steps approach to recommend a routing path. For a new ticket, the first step is to filter a subset of training tickets that are similar to it. The similarity between two tickets is measured with text-based cosine similarity widely used in Information Retrieval to generate relevance score between two text blocks. In the second step, the group routing histories from the filtered tickets are used to construct a Markov model, and the resolver is searched using VMS (Equation 2.3) from [5]. Since there are much more tickets that are not similar to a given ticket, a

normalization trick is applied to make the similarity of the new ticket to all tickets a uniform distribution.

Using the same data set used in [5] for experiments, [80] showed improvements in Mean Steps to Resolver (MSTR) compared to its predecessor’s result (routing information only). Notably, the authors claimed some additional 15% drop in MSTR after normalizing the training data to a uniform distribution. But as the filtering step further reduces the search space, the resolver may be unreachable via graph traversal. Therefore, another metric, Resolution Rate (RR) is introduced to measure the ratio of completed test cases.

Instead of processing text in a separated step as [80], in another work, Miao *et al.* [6] proposed generative models incorporating both ticket text and routing statistics, as listed in Table 2.5. A resolution model (Equation 2.4) estimates the probability of a group being able to solve a ticket, *i.e.*, $P(t|g_i)$. Where w_k is a word, or token, after text tokenization and $\#(w_k, t)$ is the number of occurrence the word appears in a ticket t . $\mathcal{T}_i$ denotes the collection of tickets solved by g_i and $\#(w_k, \mathcal{T}_i)$ is the total number of occurrence the word appears in the ticket set $\mathcal{T}_i$. A transfer model (Equation 2.5) models the likelihood of a ticket being transferred to all other groups. e_{ij} is the directed edge between g_i and g_j , a ticket transfer between the two groups. $\mathcal{T}_{ij}$ is a set of all tickets that has been transferred from g_i to g_j . Lastly, an Optimized Network Model (Equation 2.6) makes globally optimized routing decisions, where $Z(t, g_i) = \sum_{g \in \mathcal{G}} P(t|e_{ij})P(g_j|g_i)$ and $P(g_j|g_i) = \frac{\#(\mathcal{T}_{ij})}{\#(\mathcal{T}_i)}$. The global optimized solution for $P(w_k|e_{ij})$ is obtained by solving a non-convex maximum likelihood estimation problem.

Subsequently, three ticket routing strategies, namely the Ranked Resolver, Greedy Transfer and Holistic Routing, using the models to find the resolver, are proposed and evaluated. The results on the same dataset as [5] show the addition of text information indeed improve MSTR over VMS (Equation 2.3), which only considers the routing information. Among the different combinations, ONM+Holistic Routing approach outperforms the rest. Another benefit of using the generative models is its interpretability. It is possible to review what are the indicative words in a ticket for a transfer or resolution to take place for a current group.

As far as we understand, Miao *et al.* [6] used single words as tokens, without considering bigrams or phrases. The authors did not mention any preprocessing or

stopwords removal applied, which is controversial in text analytic tasks, even more so in IT domain. Besides, it is unclear how terms frequency distribution might have impacted the model’s performance. One can speculate that the results will suffer if many words are shared among many edges.

Perhaps because the two works were being done almost around the same period, performance comparison was not made between [80] and [6]. Nonetheless, as we will discuss in the next chapter, the comparison may not be even possible if two subjects do not have the same RR.

Inspired by Shao *et al.* [5], Joeng *et al.* [90] used data from bug repositories in open-source software projects to construct a “bug tossing graph” modeling the bug toss (transfer) patterns between developers in the open-source community. Experiments show Machine Learning (ML) and tossing graph joint models outperform ML-only or graph-only baseline. Bhattacharya *et al.* [91] extended the work by incorporating additional information like product-component labels and developers’ activity level into the graph. In open-source projects, a significant number of developers are volunteers. Compared to expert networks with professional engineers, bug tossing graph is more sparse. Skills required to close a bug or issue may not be exclusive to specific individuals. The actual assignment depends on factors like personal preference and commitment, which are time-variant and not easily extractable from the logs. Also for bug triaging, Xiao *et al.* [92] proposed a topic modeling approach, extending the LDA model.

Motahari-Nezhad *et al.* [93] first proposed a *conversation* management framework as a container for communication histories among support engineers in resolving the ticket. Next, the sequence of communication history can be extracted to model the routing steps between engineers [94]. By matching the current routing information of an open ticket with the historical patterns of similar tickets, the model can recommend the next potential routing destination to the human user. The similarity between tickets is computed using a weighted combination of title, keywords, description, and priority. The recommendation algorithm takes into account both the routing path before a matching step, and paths from it to potential resolvers, prioritizing routing options with a shorter distance to the resolver. A case study based evaluation demonstrates the effectiveness of the proposed recommendation method in 16 out of 20 cases investigated. Instead of predicting a routing path, as the other routing models, the recommendation-based routing produces candidates

for the next step for human users to choose. The limitation is that the human users' feedback is not considered when the recommendation is not helpful.

In summary, after modeling the ticket routing histories as a network, two groups of information are available can be used for ticket routing. Methods using ticket text was shown to perform better than those only using routing history.

2.4.3 Summary

In their comprehensive analysis of ticket arrival process, Bartolini *et al.* [88] doubted the real performance gain by adopting automated routing solutions in general. One of their arguments was that, these systems mostly relied on historical routing paths, which contained incorrect and outdated information. In contrast, Shao *et al.* [5] argued for the inclusion of intermediate steps in routing, because they might be necessary dispatching groups instead of mistakes made by engineers. Without detailed domain knowledge in the business requirement and organizational setup, the quality of training data is noisy and unreliable. Secondly, these systems do not consider workload balancing and availability of support groups. This is especially important for a global support organization consisting of teams working in a different time zone. In the following chapters, we will test the hypothesis in our setting.

The authors further proposed to consider the social interaction between support engineers as another dimension impacting the work performance, *e.g.*, people who were friends of each other got answers for requests sooner. However, collecting relevant data to model such relationships were challenging. Current systems are able to record electronic records created by experts during operations. Personal communications such as face-to-face or telephone conversations are neither collected nor correlated with work processes. Besides, collecting such information may infringe individuals' privacy, which is a growing concern in many applications.

The limitation of using historical routing paths as ground truth will be discussed in details in the next chapter. We will review and identify shortcomings of existing evaluation metrics for ticket routing problem. Ticket routing is a unique problem for support ticketing systems. Ticket processing requires tacit knowledge, experiences, and collaboration from human experts. Previous works model the problem

using Markov probabilistic models, or as classification problems. Different features and objectives are considered in the process. However, each organization has a slightly different business requirement and practice, all studies are application specific. After all, there remain open questions regarding how to effectively model the complex human experts' decision-making processing and accurately evaluate the performance of a system.

2.5 Conclusion

In this chapter, we review the important research works and their findings in IT support and service business. Firstly, we discussed information extraction works around ticket data. Standard NLP and IR related techniques have been applied to different datasets for valuable information extraction and root cause analysis. Secondly, we discussed the published works about ticket clustering and classification, with respect to their problem setting (target) and techniques used. Lastly, we analyzed the different ticket routing works, focusing on the information being considered. Overall, most of the techniques used are inspired by works from other applications, implemented with domain-specific adaptations and modifications. We will re-visit some of these works again in the following chapters, for their direct relatedness to our research problems.

Chapter 3

Evaluation of Ticket Routing Performance²

3.1 Introduction

For ticket routing problem, we assume expert group is specialized in a subset of all domains supported by the organization. Each group is only capable of solving tickets from this subset of domains. After examining the details of a ticket carefully, a processor may find that it is not within its technical domain. As a result, the processing group will route the ticket to another group, who is more likely to provide a solution. In a human routing case, the choice of the receiver group is made locally by the expert in the processing group. This decision-making process depends on the local knowledge that the processor has on the ticket and the global knowledge about other groups. General-purpose search tools may be utilized to search for information and closed tickets for reference. However, routing remains challenging, especially for staffs who are inexperienced and unfamiliar with what other groups are specialized in. Furthermore, in the support organization we studied, staffs are remotely located, and working in different time zones. With the proliferation of data mining and machine learning techniques, automated tools are brought in to help. Different systems need to be tested and compared, to understand their performance in operation. In this chapter, we focus on the evaluation of ticket

² The work in this chapter has been published in [\[95\]](#).

routing approaches. This is deemed as the fundamental problem if we want to improve ticket routing performance.

To evaluate a ticket routing system, MSTR has been widely adopted. MSTR computes the average number of routing steps over a ticket set which is used for the evaluation. To count the steps of a system-generated path, the resolver groups are used as ground truth. If the ground truth group is found in a generated path, the steps before the resolver are regarded as the routing path. However, sometimes the ground truth resolver may not appear in a generated path and then “steps to resolver” is undefined in this case. Another metric, RR, is then used to measure the proportion of tickets in a testing set that an algorithm can find resolvers for. A threshold can be set to avoid very long paths. A ticket is considered unresolved if the resolver appears beyond the threshold.

We identify two limitations of the existing evaluation metrics. Firstly, MSTR and RR are two independent metrics, with different objectives and at different scales. MSTR only considers the paths of resolved tickets, as unresolved cases would yield undefined lengths. It is virtually impossible to make a comparison between a system A with low MSTR and low RR, and another system B with high MSTR and high RR. Secondly, the current metrics take on a binary judgement, only considering the ground truth resolver and ignoring all intermediate steps leading to the resolver. MSTR considers the length of routing paths as cost. All groups other than the resolver in a generated path are classified as incorrect and counted towards the overall cost. In RR, only those tickets where the generated paths include the ground truth resolvers are considered resolved. In reality, due to the nature of the task, not all resolvers can be located easily. Routing the ticket to a group leading to the resolver does contribute to its resolution, thus should be rewarded.

In this chapter, we examine ticket routing problem from a more general information retrieval perspective. We consider the expert network as a large expert “corpus”, where each ticket is a query. The information need of a user is to locate relevant groups that can directly or indirectly resolve the ticket. When evaluating the performance of an IR system, three elements are required: a document collection, a set of queries, and a set of relevance assessment methods. Specifically, we focus on incorporating the distance between groups and resolvers in ground truth paths into evaluation. We propose a novel unified metric, namely Mean Average Distance to Resolver (MADR), which combines elements from existing metrics and

rewards partial matches with ground truth paths instead of only resolvers. We first discuss the design and properties of this new metric, then demonstrate its usability in human routing as well as fully-automated systems. Furthermore, we argue that current evaluation scheme cannot reflect the effectiveness of an assistive routing system with human involvement in real applications. Specifically, current frameworks assume a generated path to be followed by human engineers exactly. Such assumption is common in predictive studies, but falls short in a routing system which involves human experts in the process. We propose a new evaluation framework that considers human input at each routing step. The new framework can be used in couple with the new metric to access the effectiveness of automated support systems. The framework is adaptable to different step weighting schemes various from one organization to another. We experiment the evaluation framework using retrieval-based assistive routing recommendation on real testing tickets collected from a production ticketing system.

In the remaining of the chapter, we will first review the existing works in ticket routing and their evaluation methods in Section 3.2, following by discussions on their limitations are in Section 3.3. In Section 3.4, we propose the new metric, with a demonstration on using it to score a routing path. The Assistive Routing evaluation strategy and empirical studies are reported in Sections 3.5 and 3.6, respectively.

3.2 Related Work

Our work in ticket routing evaluation is directly related to studies on optimization of ticket routing and processing.

Shao *et al.* [5, 10] first proposed EasyTicket, a routing recommendation system aiming to automate the routing process. Based on observations of mining frequent routing patterns from a large number of closed tickets, their system builds a Markov model and subsequently uses it to predict the most likely next step. The main limitation of their work is that text contents in tickets are left unused. Nevertheless, Their work sets a precedent for ticket routing research problem, including employing MSTR as a standard performance measure. Sun *et al.* [80]

extended EasyTicket with textual information from ticket contents. After investigating routing sequences and text patterns separately, a hybrid system is proposed, which takes two steps before recommending a routing path. For a new ticket, the system first analyzes its content and filters historical tickets with text-based cosine similarity. In the next step, a Markov model is developed with only groups from the set of similar tickets obtained in the first step. In a similar work, Miao *et al.* [6] proposed a generative model incorporating both ticket text and routing statistics. A resolution model estimates the probability of a group being able to solve a ticket. A transfer model analyzes routing sequences and computes edge weights for every group transferring pair. Then a joint model makes globally optimized routing decisions. Chen *et al.* [85] proposed Variable-order Multiple active State search (VMS) for next routing step prediction. MSTR is the metric used for performance assessment in the above-mentioned studies. Because some algorithms [6, 79, 80] failed to produce the resolvers for some testing tickets, the RR metric is applied. Sun *et al.*'s model [31] predicted the number of routing steps for a ticket and evaluated the results with error analysis. As far as we know, no previous work explores the limitations of MSTR and RR as we do in this work.

Sun *et al.* [31] investigated the effect of expertise difference when routing a ticket. The authors proposed ticket-text-based logistic models to predict the number of required resolvers and evaluate their models' accuracy. Ma *et al.* [81, 96] took a theoretical approach to understand the performance of decentralized search models based on local-information applied to the expert network, with respect to the network's global information. The studies aim to derive bounds for performance measure which is equivalent to MSTR, from different network global setups. Miao *et al.* [82] studied information flow in an expert network. The results show distinctive characteristics exhibited by such network compared with other complex networks, due to human factors in decision making. Based on those observations, the authors propose a method to model networks similar to the real expert network for simulation and optimization experiments. During graph generation, a distance-based utility function is introduced. However, unlike distance in routing sequence used in our scoring function, their distance considers the similarity between two groups in the high-dimensional expertise space.

Venkatesan and Zayaraz [97] looked into applications of MSTR in the Bug Triaging

problem. The Bug Toss Graph is similar to an expert network, except that individual developers are the nodes. The authors argue that MSTR does not measure path alignments, and thus propose two new metrics, namely Path Similarity Metric and Path Alignment Metric. Their metrics are similar to our ideas considering partial matches with ground truth paths instead of just the resolvers. However, they are inspired by researchers in bioinformatics, where Path Alignment Indicator is used to measure the similarity between two routing paths. The problem with path alignment score is that the best prediction is identical to the human routing log, which may not be the optimal case. Recall that, the objective of ticket routing is to reduce the number of steps to resolver. The prediction would make the same mistake if it matches the same paths as in human routing.

Other investigations view this problem as a classification problem, or specifically, a multi-class classification problem. Ticket and resolver pairs in historical data are used to train classifiers. The common task is to predict the resolver or the best receiver group. Khan *et al.* [79] proposed an Automated Incident Management using Historical Information (AIM-HI) system. The system adaptively learned to predict useful resolver groups from resolved tickets using a combination of multiple supervised machine learning algorithms. Problem domain signatures are extracted from ticket contents as features. In similar works [7, 11, 12], standard classification algorithms (*e.g.*, support vector machine, decision tree) and heuristic rules are used to improve ticket routing effectiveness, which is measured by metrics such as precision, recall, F-score, and accuracy. Instead of reducing MSTR, the objective for classification is to maximize those metrics.

In SmartDispath [7], the authors recognized the fact that ticket routing is very difficult to fully automate. They introduced a “limited broadcast” design into the architecture. Specifically, SmartDispath only routed automatically tickets with high confidence. For ambiguous cases, the system would send them to multiple candidate groups simultaneously. An issue with such approach is that, the duplicate efforts spent by multiple groups. However, experts in the group have to spend time and efforts to look into the problem before deciding if it is solvable. More resources are invested if multiple groups are processing a ticket at the same time.

Lastly, research works [34, 35, 94] measuring the performance of routing in terms of Mean Time to Resolver (MTTR) are out of scope of this research.

3.3 Existing Evaluation Metrics

We aim to evaluate the effectiveness of a routing system operating on an expert network $\mathcal{G}$, where $g \in \mathcal{G}$ denotes an expert group. A collection of n tickets $\mathcal{T} = \{t_1, t_2, \dots, t_n\}$ is used as queries, with human routing paths as ground truth. For each testing ticket $t \in \mathcal{T}$, the last processing group in the human routing path is the resolver group g_r .

3.3.1 Mean Steps To Resolver

Mean Steps to Resolver (MSTR) is adopted in [5, 6, 81, 82, 85]. It measures the average length of routing paths in solving a set of tickets. It is used to benchmark both human routing logs and routing paths generated by automated systems.

$$MSTR(\mathcal{T}) = \frac{\sum_{t \in \mathcal{T}} |P(t)|}{|\mathcal{T}_{resolved}|} \quad (3.1)$$

In Equation 3.1, we use $P(t)$ to denote a routing path generated for ticket t , $P(t) = \langle g_0, g_1, \dots, g_r \rangle$ where g_0 and g_r are the first and the resolver groups, respectively. Originally, $|P(t)|$ should denote the length of the routing path, in number of steps (or number of expert groups) involved in resolving the ticket. However, as the name implies, the actual $|P(t)|$ represents the “steps” in a routing path. Each step is between two groups. For simplicity, we use the shorthand notation $|P(t)|$ to represent the number of steps. In other words, $|P(t)|$ is the number of groups minus one.

Assuming that each routing step incurs the same cost, lower MSTR implies better performance. The upper bound of MSTR is the total number of groups. The worst case happens when all groups are involved in resolving all testing tickets. However, models based on historical statistics may result in a computed routing path, which does not contain the resolver group g_r at all, for a testing ticket. In this case, the length of $P(t)$ is undefined. It is impossible to include an undefined length in computing the average length. We, therefore, modify the original denominator from $|\mathcal{T}|$ (number of tickets) to $|\mathcal{T}_{resolved}|$ (number of resolved tickets).

3.3.2 Resolution Rate

To address the shortcoming of MSTR in handling unresolved cases, Resolution Rate (RR) is used in [6, 79, 80]. RR is defined by the fraction of tickets that are resolved by the system. A ticket is resolved if its resolver group in the ground truth path matches a group in the generated routing path. Optionally, a threshold K may be applied to limit the length of a path. A ticket with its resolver matched beyond K (*e.g.*, 10 in [6]) steps is deemed unresolved.

$$RR(\mathcal{T}) = \frac{\sum_{t \in \mathcal{T}} R(t)@K}{|\mathcal{T}|} \quad (3.2)$$

where,

$$R(t)@K = \begin{cases} 1 & \text{if } g_r \in P(t) \text{ and } g_r \text{ is within } K \text{ steps,} \\ 0 & \text{otherwise} \end{cases}$$

3.3.3 Accuracy

For classification-based approaches, accuracy is often used as a measure of performance. It is defined as the fraction of correctly predicted resolvers. Accuracy is a special case of RR when $K = 1$.

$$Accuracy(\mathcal{T}) = \frac{\sum_{t \in \mathcal{T}} TR(t)}{|\mathcal{T}|} \quad (3.3)$$

where,

$$TR(t) = \begin{cases} 1 & \text{if } g_{predict} == g_r , \\ 0 & \text{otherwise} \end{cases}$$

3.3.4 Discussion

MSTR measures the average number of steps in routing paths to resolve testing tickets. But it does not consider cases where the resolver is not identified successfully, as undefined lengths are unable to be processed by the average operator. Since a smaller number indicates better performance, we can see the metric as a form of penalty on every routing that is not to the resolver. The penalty is bounded only by the number of all groups in the network. Moreover, each step in

the generated path incurs the same penalty, regardless of how close a group may be to the resolver in the ground truth path. As we discuss earlier, non-resolver groups are not completely incorrect routing. Sometimes an intermediate group serves as a dispatcher group. It could also be an important intermediate step in a collaborative effort.

On the other hand, RR measures the proportion of tickets in the entire testing set that a system is able to resolve, taking the range of $[0, 1]$. An ideal system that identifies resolvers for all tickets would have $RR = 1$. Human routing logs always have $RR = 1$. It judges each ticket with a binary score, whether resolvable or unresolvable. A ticket with a long routing path to the resolver and another one with just two steps are treated equally.

After all, the two measures are independent, with different objectives and at different scales. During our investigation, we are baffled upon seeing an algorithm achieving small (good) in $MSTR$ but also very small (poor) RR . *Is it better or worse than another system which has large $MSTR$ but better RR ?* This motivates us to design a metric that could make two systems comparable, even comparing with human routing.

3.4 Distance-based Routing Metric

In this section, we articulate the derivation of our metric progressively. It is designed to overcome the limitations of existing metrics, with the following properties.

- **Unification** The new score should be a unified score that can be used to score a system and compare different systems with each other. Ideally, it should be a combination of $MSTR$ and RR in an elegant way.
- **Normalization** The value preferably follows the range and objective of RR , *i.e.*, $[0, 1]$. Since the ultimate goal is to find the resolver with minimal steps, the best case would be that all tickets are resolved in one routing step. The best case leads the new metric to be equal to 1.
- **Quantification** The partial matches appearing in ground truth paths should be quantified, based on their distance to the resolvers.

(A) Distance-based group scoring methods.

(B) Routing path scores with different methods.

FIGURE 3.1: Comparison of different scoring methods.

- **Adaptability** It should be easily modified to incorporate more factors, *e.g.*, cost and workload, according to business requirements in real operations.

3.4.1 Unification

We observe that the denominator in Equation 3.1, $|\mathcal{T}_{resolved}|$, is, in fact, identical to the *number of resolved tickets within K steps* in Equation 3.2, *i.e.*, $\sum_{t \in \mathcal{T}} R(t)@K$. Our first attempt to combine both metrics is to multiply MSTR with RR.

$$\begin{aligned}
 MRSTR(\mathcal{T}) &= MSTR \times RR \\
 &= \frac{\sum_{t \in \mathcal{T}} |P(t)|}{|\mathcal{T}_{resolved}|} \times \frac{\sum_{t \in \mathcal{T}} R(t)@K}{|\mathcal{T}|} \\
 &= \frac{\sum_{t \in \mathcal{T}} |P(t)|}{|\mathcal{T}|}
 \end{aligned} \tag{3.4}$$

The product is a unified score, averaging the lengths of all routing steps in *resolved* tickets over *all* tickets. For now, we call it Mean Resolving Steps to Resolver (MRSTR). For a set of results from System *A* with $RR = 1$, MRSTR is the same as MSTR (the smaller the better). However, when RR is not 1, the result is hard to interpret. On the other hand, the range of the value is not normalized and the distance to resolver is not considered either.

3.4.2 Normalization

There are many ways to rescale an unbounded positive number to a range between 0 and 1. The step in Section 3.4.1 produces a value that is average over a sum, which is intuitive when designing a score for a set of all testing cases. So we keep the “average of sum” behavior and manipulate the element in numerator Equation 3.4. By taking the reciprocal of individual path length, we normalize the score to a range between 0 and 1. A ticket completed in one step (involving two groups) gets a score of 1. A set of tickets with every ticket scored perfectly will produce 1 in Mean Reciprocal Steps to Resolver (MRSR). Formally,

$$MRSR(\mathcal{T}) = \frac{\sum_{t \in \mathcal{T}} \frac{1}{|P(t)|}}{|\mathcal{T}|} \quad (3.5)$$

Up to this point, we are still performing path-level scoring. Groups in a path contribute equally to the overall scoring.

3.4.3 Quantification

To quantify partial matches with ground truth into the metric, we define a generic distance-based scoring function for individual groups involved in a path. Each path takes the average group scores from its respective members. Then the average path score is the Mean Average Distance to Resolver (MADR).

$$MADR(\mathcal{T}) = \frac{\sum_{t \in \mathcal{T}} AverageGroupScore(t)}{|\mathcal{T}|} \quad (3.6)$$

where,

$$AverageGroupScore(t) = \frac{\sum_{g \in P(t)} DistBasedGrpScore(g)}{|P(t)|} \quad (3.7)$$

3.4.4 Adaptability

The generic distance-based group score in Equation 3.7 can be customized to best suit different business requirements. One option is to take the reciprocal of the distance from each group to the resolver, which results in the Reciprocal Distance Score (RDS).

$$RDS(g) = \frac{1}{d(g, g_r)} \quad (3.8)$$

where $d(g, g_r)$ is the distance between a group g and the resolver group g_r .

Inspired by the half-life utility score of Breese *et al.* [98], we propose the Half-life Distance Score (HLDS) to be used to model a group's relevance to the resolver, which drops exponentially with its distance to the resolver. Formally,

$$HLDS(g) = \frac{1}{2^{d(g, g_r)}} \quad (3.9)$$

Both RDS and HLDS are examples of how the distance-based group scoring function could be defined. An organization may decide what scheme best suits its system and requirement. Additional information can also be added to the score. For example, different costs and workload can be assigned to groups before computing the group score.

3.4.5 Analysis

The difference between Equation 3.8 and Equation 3.9 is illustrated in Figure 3.1a. The horizontal axis, *i.e.*, distance to the resolver, is reversed to be consistent with the notion of a routing sequence $\langle g_0, g_1, \dots, g_r \rangle$ where the resolver group is the rightmost node (0 on the axis). The weights of all distances to the resolver are plotted. Note that a group's weight drops rapidly to below 0.1 as soon as the routing exceeds 3 steps when HLDS is used. Groups that are beyond 6 steps away barely gain any weight. Whereas for RDS, a group with 9 steps away from the resolver still receives a score of 0.1. Observe from the figure, RDS is more sensitive for groups beyond two steps away from the resolver. Compared to HLDS, it shows more distinctive intervals. Meanwhile, HLDS is better in differentiating weights from groups that are relatively close to the resolver.

Summing up, Figure 3.1b compares the scores for routing steps from 1 to 9, using different path scoring functions. A perfect score of 1 is given to path with 1 step. Reciprocal Steps to Resolver (RSR) is a path-level scoring. The score is computed with only the length of the path. The initial decline of the curve is steep and

slows down when the routing path is more than 6 steps. As discussed, RDS is more sensitive to groups further from the resolver in a path. Compared to MADR-HLDS, the curve for MADR-RDS shows more decline of scores as the length of the routing path increases.

3.4.6 Scoring a generated routing path against ground truth

In Figure 3.2, we show how to score partially matched paths generated by an automated system against a ground truth path from human routing. Since our sample path is relatively short, we choose MADR with HLDS for demonstration purpose. It can be easily replaced by RDS or other group scoring functions. we use a shaded right arrow to represent a routing step and consider the initial assignment as a routing step.

The difference from existing scoring methods is twofolded. Firstly, instead of resolver only, we consider the whole human routing path corresponding to the testing ticket as the ground truth. This allows us to reward partial matches in a generated path, even if the final resolver is not reached. Secondly, unlike many predictive studies, where the exact match with the ground truth routing is scored highest, we advocate that the human routing results may not be perfect (and this is why the problem of ticket routing is worth studying). Thus it is possible for a generated path to outperform the ground truth one (*e.g.*, paths *A* and *B* in Figure 3.2). Lastly, since the optimal goal is to find the resolver the soonest, a perfect routing scenario is to reach the resolver in the first routing (assignment), as demonstrated in Path *A*. To summarize, the best routing has a score of 1. Short routing is preferred than long routing, and partial matching with human routing is rewarded.

3.5 Assistive Routing Evaluation Framework

In human routing, an expert group makes a decision on which group it should route the ticket to (Figure 3.3a). For existing models computing a routing path for a new ticket as output, the evaluation phase only considers ground truth resolvers. Even with partial scoring metrics proposed, it assumes that human experts would strictly follow a predicted path. As shown in Figure 3.3b, the computed path is a list of expert groups, and the first group will be assigned to the ticket. If it is not the resolver, the ticket will follow the list to be routed to the next group.

However, we argue that human routing decisions should not be ignored during evaluation. The reason is that, if the objective is to find the resolver, it makes little sense to predict a path in the first place, knowing some of them are “incorrect”.

	Routing path	Path scoring	Remarks
Ground truth (with group scores underneath)		$\frac{0.125 + 0.25 + 0.5 + 1}{4} = 0.469$	For each test ticket, we extract the human routing path in history log as ground truth. Note that the ground truth path score less than 1, simply because the human routing is not perfect. Nonetheless, we will use it as a reference to score other output routing paths for the same ticket.
Path A		$\frac{1}{1} = 1$	This is considered as a perfect routing path in our scoring. Since the objective is to find the resolver for a ticket, a direct routing to the resolver deserve the highest score.
Path B		$\frac{0.25 + 1}{2} = 0.625$	This path has managed to reach the resolver in 2 steps. The intermediate step has a match in the ground truth path (blue group). The score for the matched group is counted for the path score. Though not perfect, it is shorter and partially match the ground truth group. Thus it scores higher than the ground truth.
Path C		$\frac{0 + 0 + 0.5 + 1}{4} = 0.375$	This path contains 4 steps to resolver. However, only 1 group match other than the resolver. The path scoring only considers then matched groups. None matched intermediate groups in the generated path receive no score. But the path score takes the average of all groups in the generated path. Thought it has the same length as the ground truth, it scores slightly lower.
Path D		$\frac{0 + 0 + 0.5 + 0.125}{4} = 0.156$	Assuming the routing stops without reaching the resolver. Partial matches to the ground truth still earns some credit. But it is much lower compared to the ground truth score.
Path E		0	Completely unmatched routing deserves nothing according to our path scoring rules.

FIGURE 3.2: Illustration on scoring of generated paths. In ground truth (top row), the resolver group is labeled with a green tick. Based on distances to resolver, HLDs scores of groups in ground truth are computed and labeled under the group icons. We use yellow, blue and green to represent intermediate groups in the ground truth path. In subsequent cases, groups with these colors indicate partial matches with ground truth and grey icons are unmatched groups.

(c) Assistive routing

FIGURE 3.3: A sample expert network, and workflow illustrations for human routing, computed routing, and assistive routing.

Furthermore, if the next routing is not to the resolver, as indicated in the predicted path, it should not be followed. From an information retrieval perspective, the information need of the user at the point before routing is the resolver group. A system should retrieve candidate groups with information presentation. Human experts will review the candidate list to determine the resolver. If the resolver is not in the candidate list, the group should make its own decision on where the next group is in the ground truth path. We propose a new routing strategy, namely *Assistive Routing*, to test the performance of automated systems against the ground truth path.

As shown in Figure 3.3c, for a routing system that assists experts in choosing the next expert group, an evaluation scheme considers experts' input at every routing step. The system generates a list of candidate resolvers for the current expert group to consider. This is to simulate the scenario when a human expert is shown with a list of potential receivers in a real application context. We assume that the human expert who handles a ticket is knowledgeable enough to *identify the right resolver from the top- K candidates*, if the ground truth resolver g_r is indeed in the candidate list, if g_r is not among the top- K , we assume that the human expert would make an independent decision, which is the next group in ground truth human routing. The process continues until the ticket reaches its resolver g_r .

3.6 Experiment and Result

Using the evaluation framework discussed in Section 3.5, we test a retrieval-based routing system on closed tickets in production ticketing system from a large software company. We collected 236,980 incident tickets with logged routing paths from the software provider's customer support system in 2015. These tickets were processed by 10,902 expert groups. Based on the time when a ticket was closed, the first 211,853 tickets were used for training and the remaining 25,127 tickets were used for testing. Human routing is first evaluated with MSTR, RR, and MADR-HLDS as discussed in Section 3.4.

Retrieval-based routing is a simplified form of decentralized search used by Ma *et al.* [81], without considering individual's expertise. We first index texts of tickets containing resolver groups in the training set as documents. Then, we use the

TABLE 3.1: Result of different routings; best result in boldface.

Routing method	MSTR	RR	MADR@3
Human routing	2.770	1.000	0.881
Retrieval-based	2.675	0.565	0.911

FIGURE 3.4: Sensitivity of parameter K from 3 to 10 in MADR metric.

content of a testing ticket to query the resolver groups of relevant tickets, using *MoreLikeThis* similarity query in Lucene³. Finally, the returned groups will be voted and ranked based on the result counts.

In each routing, top 10 candidate groups are retrieved. If the resolver exists in the list, all groups ranked higher in the list are considered intermediate groups. The number of steps to the resolver is the rank of the resolver group in the list. Otherwise, the ticket is unresolved. For assistive routing, in each routing recommendation, top- K candidate groups are retrieved. We experimented K with values from $[3, 10]$. The results are shown in Figure 3.4. If the resolver is not in the list, the next group will follow the ground truth path. To simulate the additional input from the processing group, we append a short description about the group to the testing ticket content to formulate the new query for subsequent routings.

From the results shown in Table 3.1, retrieval-based routing outperforms human routing in terms of MSTR but getting a much lower RR score. Note that both metrics do not consider human routing decision made during each routing. This observation is exactly what motivates us to propose the new metric. When evaluated by MADR metric, human routing scores 0.881. The retrieval-based routing system shows improvements compared with the human baseline. As also seen from

³<http://lucene.apache.org/>

Figure 3.4, the performance improves when the value of K increases. This is expected since it is more likely to find the resolver with a longer candidate list. We see these results are promising, in demonstrating the usability of our proposed metric MADR, as well as the evaluating strategy considering human routing option when the resolver is not correctly recommended.

3.7 Conclusion

In this chapter, we review and identify shortcomings of existing evaluation metrics for ticket routing problem in expert networks. MSTR and RR are independent and do not reward routing to intermediate groups that lead to resolver. To compare systems that may have inconsistent MSTR and RR comparison results, a unified scoring framework is proposed. The final score is a bounded real number between 0 and 1 for ease of comparison of different systems. The distance between a group to the resolver in a ground truth path in evaluation is incorporated in scoring. Components of the new framework can be modified to adapt to different business requirements, making our proposed framework more flexible and appealing to real business needs in software support. We also propose Assistive Routing, considering of human routing input when evaluating any ticket routing approach. We justify why it is more applicable than a fully-automated evaluation. In Chapter 5 and 6, we will evaluate our proposed routing frameworks using the new evaluation metric in Assistive Routing.

Chapter 4

Software Mention Extraction and Linking⁴

4.1 Introduction

In our study, a software company offers hundreds of products in its portfolio, covering numerous application areas. These softwares are installed and used by tens of thousands customers around the world. To provide around-the-clock technical assistance to the customers, a large team of support staffs are employed. They receive and analyze support tickets from the customers, before helping them with solutions.

When a support expert processes a ticket, one of the first clues he or she will look for is which product and version the problem is affecting. Some issues only affect specific versions of a software, while some only occur after change of version. From a ticket, extracting software mentions and correctly linking them to the product and version would benefit ticket routing as well as other ticket processing application. These include automatic ticket analysis and understanding [15, 57], advanced ticket search [42] and classification [9], root cause analysis [19], resolution recommendation [37, 41, 52], and in-domain knowledge-base construction [2, 51].

⁴The work in this chapter has been published in [99].

In this chapter, we investigate how to effectively extract software mentions from ticket, and link them to formal products in a catalog. Firstly, we analyze a collection of manually annotated tickets, for their language patterns, especially on homonyms and synonyms of software product mentions. Then, we explore multiple resources and techniques in designing a rich set of domain-specific features for software mention extraction and linking. Lastly, we show that linear models using the proposed features are able to achieve better performance in both tasks, compared with state-of-the-art baselines.

In the remaining of this chapter, we first review the related works in Section 4.2. Next, we present the problem with data analysis findings in Section 4.3 and propose our solution in Section 4.4. We present the experiment setup and discuss the results in Section 4.5, before concluding the chapter in Section 5.7.

4.2 Related Work

Software entity extraction has been studied to understand software usages. Pan *et al.* [100] proposed a bootstrapping method to extract software entities from scientific publications using contextual pattern matching techniques. Compared with research articles, the software tickets we study are informal and noisy. Different features and knowledge sources are required to address the name variations and other irregularities in our problem domain. Ye *et al.* [101, 102] investigate software-related terminologies (*e.g.*, programming languages, platforms, and APIs) extraction from Stack Overflow⁵ questions. Here, the product names we extract do not fit into their categorization. Further, we extend the extraction task by linking the extracted software mentions to a catalog of formal products.

In an e-commerce context, Putthividhya and Hu [103] extract product names and properties from online marketplace listings using a supervised learning approach. Their method also links brand mentions to a catalog and discovers new brands. On the CPROD1 contest dataset [104], Wu *et al.* [105] propose a hybrid framework for product mention recognition and linking to a product catalog with a large number of products. Yao and Sun [106] investigate mobile phone names extraction and normalization using a novel semi-supervised labeling scheme to generate training

⁵<https://stackoverflow.com/>

FIGURE 4.1: An example ticket, with product and version mentions . They are linked to their respective official names.

data at scale. The weakly labeled data are used to train a Linear Chain CRF model to recognize mobile phone names from online forums.

Focusing on the linking aspect of the problem, Vieira *et al.* [107] use a binary classifier with features generated by exploring similarities between a mention and the official name and its description. The model is used to determine if a pair of mention and official name is correctly linked. Instead of using versions as a supplementary features, we label them along with the product names, and train a model to recognize both as entities.

Our research is also related to analytical IT management systems [1, 15, 108, 109]. Many of these systems only rely on contextual features and patterns for information extraction, *e.g.*, noun phrases extraction. To the best of our knowledge, this is the first work studying software product name extraction and linking from support tickets, leveraging both contextual and external information sources.

4.3 The Problem and Data Analysis

Given a support ticket, the tasks are to recognize a set of product name mentions $M = \{m_1, \dots, m_N\}$ from the ticket, and map the mentions to their official names in a pre-defined product catalog $\mathcal{P} = \{p_1, \dots, p_{|\mathcal{P}|}\}$, *i.e.*, $m_i \mapsto p_j$. The mention m_i is known as a surface form of p_j . The mapping process is known as entity linking, also refereed to as normalization by Yao and Sun [106]. A product name mention can be **linkable** or **unlinkable**, depending on whether there exists a matching entry in the catalog. In Figure 4.1, “ERP” and “NW” are linkable mentions, while “windows” and “MYSQL” are unlinkable products as they do not match any official name in $\mathcal{P}$, *i.e.*, not supported by the company.

TABLE 4.1: Statistics of datasets. Length is in number of tokens.

	#Ticket	Min. Length	Max. Length	Average Length
SV	980	13	176	62.6
BC	961	12	188	61.6
FI	887	12	200	62.1
BI	541	11	166	52.2

4.3.1 Dataset Annotation

To analyze the language usage and patterns, we randomly select and annotate a subset of support tickets from a support system. Firstly, 3 millions anonymized English support tickets are collected. Subsequently, 3,369 from 4 application areas, namely service (SV), basic (BC), finance (FI), and business intelligence (BI), are randomly sampled from the collection to form 4 datasets for manual annotation. The remaining tickets are the unlabeled ticket set. As listed in Table 4.1, the minimum, maximum, and average length of tickets in each dataset are similar. In terms of size, BI is considerably smaller than the others.

Two domain experts are engaged in annotating the tickets. Both are well briefed on the objectives and given the same product catalog for reference. Instead of having each to label all tickets and resolve differences. The annotators work together remotely to label, verify, and revise the results iteratively to make sure the results are agreed and consistent. Tickets from the 4 datasets are mixed and shuffled before annotation to avoid any bias. We deploy a Brat [110] server, a web-based interactive tool for text data annotation. Tickets are tokenized and labeled with the following:

- mentions of software products that are **linkable** to product catalog (*e.g.*, SAP Netweaver), and their respective official names.
- mentions of software products that are **unlinkable** to catalog, *e.g.*, Windows, MySQL;
- mentions of software versions, and the “version-of” association between a product mention and version mention.

Annotators may use the **meta-product** information from the input ticket to infer the official names for linking. Meta-product is a categorical entry that is selected

TABLE 4.2: Number of entity and version mentions in 4 datasets.

Dataset	Linkable product	Unlinkable product	Version
SV	563	39	215
BC	587	376	394
FI	64	2	32
BI	373	288	772

TABLE 4.3: Statistics of linkable products. $|M_s|$, $|P_{linked}|$, and $|P_{meta}|$ are the number of unique surface forms, unique official products linked to, and unique meta-products.

Dataset	$ M_s $	$ P_{linked} $	$ M_s / P_{linked} $	$ P_{meta} $
SV	18	58	4.5	25
BC	53	117	3.2	40
FI	17	24	1.8	21
BI	22	81	5.1	46

TABLE 4.4: Proportions of tickets, linkable and unlinkable product mentions with associated version mentions (w/v)

Dataset	Tickets w/v	Linkable w/v	Unlinkable w/v
SV	388 (39.6%)	393 (69.8%)	27 (69.2%)
BC	395 (41.1%)	509 (86.7%)	312 (83.0%)
FI	45 (5.1%)	25 (39.1%)	0 (0.0%)
BI	245 (45.3%)	360 (96.5%)	277 (96.2%)

when a ticket is being created. For example, in Figure 4.1, the first occurrence of “ERP” is linked to ‘SAP ERP 6.0’ because version “6.0” presents next to it. Similarly, “NW” is linked to ‘SAP NetWeaver 7.0’ indicated by “7.0”. When “ERP” appears without version “6.0” (*i.e.*, the last line in the ticket), the meta-product becomes useful for linking.

Instead of considering versions as part of product names, we label them as a standalone type of entities. In software tickets, versions may precede the product (*e.g.*, “...using **7.3** of SAP NetWeaver ...”), multiple versions may associate with the same product (*e.g.*, “...NW **7.0 SP6 Patch2** ...”), and versions may not appear in the vicinity of a product (*e.g.*, “...upgrading from **7.0** to **7.1**”). Eventually, each token is assigned with the entity type with a prefix to indicate its “BIO”, namely, the ‘B’egin, ‘I’nside of an entity mention, or ‘O’utside. Total 7 label types are generated from the 3 entity types.

4.3.2 Dataset Analysis

Entities. The entity level summary of the annotated data is shown in Table 4.2. Remarkably, FI has fewer labeled entities compared to SV and BC, despite having the similar sizes. SV and FI has fewer unlinkable products, while BI has disproportionately high version mention counts. These could be explained by the distinctive usage patterns in different dataset.

Product names. Table 4.3 reports the statistics of product mentions and linking in the datasets. BC has the most diverse official products linked to, whereas BI has the most average mention per linked product. In terms of unique meta-products, SV and FI have only half as many compared with BI and BC. Note that meta-product may not be consistent with the mention-level product linking, especially when multiple products are mentioned in a ticket.

Versions. We propose to extract version mentions along with product names and use both for official product linking task. As shown in Table 4.4, all datasets, except FI, have a reasonable ratio of tickets having at least one version mention, with BI having the highest. BI also has the highest version association rate for both linkable and unlinkable products, with 96.5% and 96.2% , respectively.

Observation 1. Product name mentions take various surface forms. Customers extensively use, even create aliases for products when composing a support ticket.

The upper portion of Table 4.5 shows the surface forms for ‘SAP NetWeaver 7.4’ and their counts in the annotated tickets. There exist exceptional and erroneous cases due to legacy reasons, or by mistake. We observe that some surface forms are token subset of the official name (*e.g.*, ‘SAP NetWeaver’, ‘NetWeaver’); some contain all uppercase letters and numbers from the official names (*e.g.*, ‘SAP NW’, ‘NW7.4’). After all, string similarities between the surface forms and their official names may vary in different tickets. To make confident extraction and linking decisions, external knowledge and resources should be leveraged.

TABLE 4.5: Mentions of ‘SAP NetWeaver 7.4’ (top table) and a list of different products “NW” has been linked to (bottom table). Exceptional and erroneous mentions are underlined.

Surface form of ‘SAP NetWeaver 7.4’	Count
NW	22
Netweaver	14
SAP Netweaver	4
netweaver; SAP NW	3
NetWeaver; NW740, <u>portal</u>	2
Sap netweaver; nw; NW7.4; SAP	1
NetWeaver; SAP Net Weaver; SAP	
NW7.4; Ntweaver; NetWeaver AS;	
NW7.40; SAP netweaver <u>SAP PO</u> ; <u>BW</u>	
“NW” links to	Count
SAP NetWeaver 7.4	22
SAP NetWeaver 7.3	16
SAP NetWeaver 7.0	11
SAP NetWeaver Enterprise	1
Search 7.3; SAP enhancement	
package 1 for SAP NetWeaver 7.3;	
<u>SAP Solution Manager 7.1</u>	

Observation 2. A mention may link to multiple products; many of them are the same product with different versions.

Due to the Observation 1, mapping surface form product mentions to official names are not trivial. It is especially challenging to disambiguate the same products with different versions. Customers may use the same acronym for different versions of a product. The lower portion of Table 4.5 shows all official products which ‘NW’ is linked to. The top-3 most frequently linked names are the same product with different versions. During linking, version mentions appear in the same ticket could be useful for dictation.

Observation 3. Without a knowledge base, linking is challenging even for domain experts.

Much research efforts on entity linking task focus on mapping entity mentions to their corresponding concepts in a knowledge base (*e.g.*, Wikipedia) [111]. Without

a structured knowledge base or an alias table, linking task is challenging even for human annotators. In our case, only a ticket and a product catalog are available as input. During annotation, the experts use much of their background knowledge and discretion for linking, which may be unexplicit and speculative. To model this process, we explore external and inter-entity features, especially between product and version mentions, to improve linking performance.

4.4 Mention Extraction and Linking

Our system consists of two main modules for software mentions extraction and linking tasks, respectively. Both take a ticket and the product catalog as input. Multiple domain-specific features are generated from the labeled and unlabeled tickets. Using these features, we train a Linear Chain Conditional Random Fields (CRF) [112] extraction model and a Support Vector Regression (SVR) model for Linking module.

4.4.1 Product Names Extraction

The extraction task is a domain-specific named entity extraction, or named entity recognition (NER) problem, which is an instance of the sequence labeling task, technically. We use a CRF model with the features to be discussed in Section 4.4.2. The model is also used in similar applications with different features [101, 106, 107]. Compared with the increasingly popular neural network-based models, the CRF model has the following advantages. It is flexible in incorporating additional categorical signals as features, and easier to train on dataset with sparse labels. It also provides certain explainability, by assigning explicit weights to each feature for a label [113].

4.4.2 Features

We use both labeled and unlabeled tickets, also incorporating external resources to generate the following groups of features.

Basic Features. Following previous studies [106, 107], we use lexical and grammatical features in a base model. Specifically, for each word, we consider its current form, lowercase, prefixes, suffixes, and word shape as features. Part-of-Speech (POS) tags and prefixes of the tags generated from Stanford CoreNLP tool⁶ are also used as features for each token. In addition, n -grams are considered for a word to include its neighbors in a context window size of 5.

Word Clusters. Word clustering techniques, specifically Brown Clustering [114], have been effective in mitigating the term sparsity problem in general and specific domain NER systems [101, 106, 115]. It assumes that similar words should appear in the same contexts. During training, for each iteration, the algorithm merges semantically similar words into a fixed number of classes based on the log-probability and incurs the least loss in global mutual information. After training with the unlabeled tickets as input, the output vocabularies are organized in a binary tree structure, such that each word can be represented using a bitstring cluster ID. We use the prefixes of 10, 12, 14, 16, 18, 20 bits to represent a word as features.

Word embedding techniques [116–118] are significant recent advancements in text representation. Unlike Brown Clustering, word embedding models output dense vectors from training. Semantically similar words are close to each other in the embedding spaces. We apply three different word embedding models, namely Word2Vec [119], Glove [117] and FastText [118] on the unlabeled tickets to generate word vector models with 100 dimensions each. In post-processing, we apply a k -means clustering algorithm to produce 500, 1000, 1500, 2000, 3000 clusters for each embedding model and use the cluster IDs as word features.

Prototypes. In [120], Guo *et al.* compare different ways of using word vectors as features for the CRF model in NER task and propose a novel approach based on *prototype-driven learning* [121]. It assumes that the semantically similar words should be tagged with the same label. Following their method, we generate prototype features for each entity label by analyzing the collocation of labels and words. Specifically, the normalized pointwise mutual information (NPMI) [122] is computed for a label l and word w using $NPMI(l, w) = \frac{\lambda(l, w)}{-\ln p(l, w)}$, where $\lambda(l, w) = \ln \frac{p(l, w)}{p(l)p(w)}$. In labeled tickets, we calculate NPMI score for each pair of word-entity label, then rank the words in descending order of the score for each

⁶<https://stanfordnlp.github.io/CoreNLP/>

TABLE 4.6: Example prototype words. Labels are ‘B’egin and ‘I’nside, for ‘L’inkable, ‘U’nlinkable products and ‘V’ersions.

Label	Top 5 ranked prototype words
B-L	Solution, SolMan, PI, NW, SOLMAN
I-L	Manager, manager, Objects, Platform, BusinessObjects
B-U	Windows, Oracle, IE, JAVA, Java
I-U	Explorer, J9, SERVER, Hat, linux
B-V	4.1, 7.1, 2008, 7.4, 4.0
I-V	.1, sp04, G, bit, SP03

label. We empirically choose the top 50 words as prototypes for each label. The top 5 prototype words for each label are shown in Table 4.6. Binary features are generated for a word token by checking if it is in the prototype list of each label.

Search Results. Rüd *et al.* [123] propose that a public search engine can be a useful tool to determine the validity of an entity or a phrase to a domain. Intuitively, if a span of words is a valid software name or related to the field, a search engine will return more IT-related links in its results. To this end, we experiment with the Bing Web Search ⁷. Specifically, we use a token and its surrounding words to form a query as input to the search engine and collect the top- k results returned (*e.g.*, $k = 50$). Each result entry contains title, summary, and URL. Subsequently, we compute the ratio of results having query terms in their title, summary, and URL, respectively. Also, we compute the ratio of results having predefined domain indicator words (*e.g.*, the company name) in their results.

However, querying search engines is not always free of cost. To reduce the number of candidate queries, we filter all possible n -grams using POS tag patterns, query likelihood score [124], and heuristics. Query likelihood score is the joint probability of neighboring tokens, obtained using a language model trained on unlabeled tickets. Phrases with query likelihood score below 0.7 are filtered as they are unlikely to occur. After filtering, 75,000 valid queries are used as input to the API. The computed ratios from search results, as well as the query likelihood scores, are rounded to the nearest 0.01 and used as categorical features.

⁷<https://azure.microsoft.com/en-gb/services/cognitive-services/bing-web-search/>

Algorithm 1: Dictionary constuction

Input : $p \in \mathcal{P}$ a formal software product name in the product catalog $\mathcal{P}$;
training ticket sets with entity annotation and linking $\mathcal{T}_{Train}$; word
embedding models E

Output: C , list of names of p

$C \leftarrow \emptyset$;

foreach *ticket* $t \in \mathcal{T}_{Train}$ **do**

foreach $m \mapsto p$ *in* t **do**

$C \leftarrow \text{Tokenize}(p) \cup \text{RuleGenerator}(p)$;

$N \leftarrow \text{NearestNeighbor}(m \cup C, E)$;

$C \leftarrow C \cup \{m\} \cup \{N\} \cup \{R\} \cup \text{LowerCase}(C \cup N)$;

pedia. We take the software names from a Wikipedia page⁸ for a crowd-sourced list with reasonable quality. The names are post-processed and used for token and n -grams lookup.

Dictionary. Previous works [101, 106] manually create dictionaries and observe performance improvement in their NER tasks. Chen *et al.* [125] propose an unsupervised method to build software-specific dictionary of synonyms and acronyms by using distributed word similarities. Inspired by their approach, we design an algorithm using a combination of semantic similarity and heuristics to build a dictionary automatically, with the labeled mentions as seed candidates. The outline of the algorithm is shown in Algorithm 1.

In Section 4.3.2, we observe that a mention m could be a substring, or has common tokens with its official name p . In Line 4, we consider each token of p as candidate, and use heuristic to derive more. The following 3 rules are detailed as example:

- p less the company name, with and without the version, *e.g.*, ‘SAP NetWeaver 7.4’ $\rightarrow$ ‘NetWeaver 7.4’, ‘NetWeaver’.
- All capital letters in p less the company name, with and without the version, *e.g.*, ‘SAP NetWeaver 7.4’ $\rightarrow$ ‘NW 7.4’, ‘NW’.
- Capitalized first letters of all tokens in p less the company name and the version *e.g.*, ‘SAP BusinessObjects Business Intelligence platform 4.2’ $\rightarrow$ ‘BOBJ 4.2’, ‘BOBI’

⁸https://en.wikipedia.org/wiki/List_of_SAP_products

In Line 5, the nearest neighbors are obtained from trained word embedding models (Section 4.4.2). During experiments, we observe that the same word may have different nearest neighbors in different word vector models. Therefore, we use multiple embedding models for dictionary building. In Line 6, all candidates from previous steps are merged, together with their lowercase forms. Once built, for a token in ticket, a binary feature is generated representing if it matches an entry in the dictionary.

4.4.3 Product Name Linking

Outputs from the extraction module are the inputs to the linking module. Given a mention m and its version v in a ticket with a ticket-level meta-product t , the task is to create a mapping from a mention to the correct official product $m \mapsto p \in \mathcal{P}$. We assume each mapping is independent of other mappings in the same ticket. Noted that the version mention v and meta-product t may not be always available for every mention m .

We employ a pairwise linking model for the task. Specifically, given a pair of mention and product $\langle m, p \rangle$, the model is trained to determine the likelihood of having m correctly linked to p with a score between 0 and 1. For each m , the likelihood scores are computed for all possible p 's and the p with the highest pair score is taken as the linking target. Formally, $\Gamma(m) = \arg \max_{p \in \mathcal{P}} \phi(m, p)$ where $\phi(m, p)$ is a regression model.

We deploy a SVR model to estimate the probability ϕ , using the features detailed in Table 4.7. In training data, the labeled $\langle m, p \rangle$ pairs are positive samples. For each positive sample, a negative pair $\langle m, p' \rangle$ is generated with a random $p' \in \mathcal{P} \wedge p' \neq p$. During testing, all possible pairs are generated for each $m \in M_{test}$, i.e., $\langle m, p \in \mathcal{P} \rangle$ as input to the trained model. p in the pair that has the highest score is the linking result.

4.5 Experiments

We randomly split each labeled dataset into 70/10/20 percents as training, validation, and testing sets. Using the features and models discussed, we first evaluate

FIGURE 4.2: Performance comparison of the CRF model and the neural network baseline models for the extraction task. Plots on the left column show average *Precision*, *Recall* and *F1* scores over all label entity types, grouped by method and dataset. While the right column shows the MUC measures considering partial matching with ground truth.

TABLE 4.7: Features for linking a mention m with version v , to an official product p , and ticket-level meta-product t . $+$ is vector addition operation.

Feature	Description
Alphabetical	m has only alphabetic characters.
Numerical	m has only digits, dashes, and dots.
Exact match	m is an exact match of p .
Substring match	m is a substring of p .
Character subset	Characters in m and p in the same order.
Character count	Number of characters in m .
Common char	Count of common characters.
Surface match	m is an exact match of p .
Surface subset	m is a substring of p .
Surface position	Position of the first character of m in p .
Version match	Version of m is a substring of p .
$sim(m, p)$	Embedding similarity of m and p .
$sim(m + v, p)$	Embedding similarity of $m + v$ and p .
$sim(m + v + t, p)$	Embedding similarity of $m + v + t$ and p .
$sim(t, p)$	Embedding similarity of t and p .

the product name extraction and linking modules, then examining outputs from the end-to-end system for error analysis.

Standard *Precision* (Pr), *Recall* (Re), and $F1$ measures are used as evaluation metrics. The extraction module is first evaluated at the entity level with strict matching for entity types. A predicted entity is a “true positive”, if and only if both the word spans and entity type are correct. In addition, we also include the metrics used in Message Understanding Conference (MUC) [126] to provide a lenient measure of Pr , Re , and $F1$, considering partial matching [127].

Comparing system outputs with the golden labels, MUC measures account for 4 numbers: 1) the number of entity boundary correct α , 2) the number of type correct β , 3) total number of possible answers (boundaries and classes in golden labels) γ , and 4) total number of output answers (boundaries and classes in output) δ . MUC metrics are computed using $Pr_{MUC} = \frac{\alpha + \beta}{\gamma}$ and $Re_{MUC} = \frac{\alpha + \beta}{\delta}$. Lastly, $F1_{MUC} = \frac{2 \times Pr_{MUC} \times Re_{MUC}}{Pr_{MUC} + Re_{MUC}}$.

4.5.1 Baseline methods

In recent years, deep neural network models have been increasingly popular in NER applications. It is interesting to see how the CRF model using our proposed features compares to neural network baselines. To this end, we select 4 popular deep neural network models for named entity extraction as baselines [128–131]. These models have similar encoder-decoder frameworks, only different in choices of certain network layers. Except for Chiu *et al.* [128], all other models use a CRF output layer. We represent them by their [character encoder]-[word encoder]-[output] triplets for simplicity, namely, **CNN-LSTM-SOFTMAX** [128], **LSTM-LSTM-CRF** [129], **CNN-LSTM-CRF** [130] and **GRU-GRU-CRF** [131]. Both LSTM and GRU models are bi-directional, incorporating contextual information from both left and right side of the current token. The details of each model are beyond the scope of our research.

The performance of neural models is subjective to parameters tuning. We adopt the same common parameter settings for all models, *i.e.*, using 100 and 25 dimension vectors for word embeddings and character embeddings, respectively. Each model is trained using tickets from the training set, optimized on the validation set and evaluated on testing set.

For the linking module, it is not straight forward to compare with standard entity linking methods which are using a knowledge base as input. We choose **Fuzzy string matching** using different queries, as well as the models from [106] and [107] as baselines. Fuzzy string matching computes Levenshtein Distance [132] between a query string q and each $p \in \mathcal{P}$. Three variations of q are proposed: $q = m$, $q = [m, v]$, and $q = [m, v, t]$, where $[\cdot]$ is string concatenation. The p with the minimal distance is linked to m . Ties are broken by comparing the probability of p occur in all tickets.

GLEN [106] is a rule-based linking method explicitly designed for mobile phone names normalization. We modify the approach using a candidate voting with confidence score computed similarly. For each of m and the top- k most similar words in embedding space, inversely lookup for p using the candidate names generated in Section 4.4.2. p with the most congregated vote is linked to m . We heuristically set k to 200.

ProdLink [107] uses features extracted from m and the descriptions of each product to train a Random Forest classifier for linking. Since $\mathcal{P}$ does not contain detail product descriptions in our setting, we tokenize p to generate the features described in [107].

4.5.2 Experimental Results

Figure 4.2 presents results from the extraction models, with one dataset per row. The left column shows the average performance for all entity types in a strict entity level evaluation, while the right column shows the MUC metrics. It appears that the results from neural network models fluctuate over different datasets, perhaps due to the differences in ratio of labels. In comparison, performance of the CRF model with the proposed features is more consistent. It is better than the baselines in Precision and $F1$ on all datasets, in both columns. Remarkably, when the labels are sparse in dataset FI, the model shows a definite advantage over the neural models. Recall that labels for all entity types are much fewer in FI compared to the other three datasets. Our proposed feature-based model is robust enough to learn from limited labeled data points, performing especially well in precision. However, the CRF model on BC and BI datasets yield Recall either on par with or slightly worse than baselines. One explanation is that, without domain-specific and external signals, the neural models may generalize well (higher recall) but make more false positive predictions (lower precision).

Except for the CNN-LSTM-SOFTMAX model, the other neural network models consist of a linear CRF model as a network layer. A question raised is how the CRF layer would affect the performance of a model in NER task on noisy text data. Sutton and McCallum [113] recommend the use of binned features instead of real-valued features as input to a CRF model for better performance. In neural models, the output from the word encoding/hidden layer is used directly as input to the CRF layer for tagging output [129]. Performance improvements are reported when applied in news dataset. It could be the noise in ticket text that limits the effectiveness of the model in general. On the other hand, the model using CNN for character feature extraction exhibits mixed results compared to models with LSTM encoder. Overall, the CRF model with the proposed features shows superior performance in Precision and $F1$ in most cases, but not as significant in Recall.

TABLE 4.8: Ablation study of features in CRF, on MUC $F1$

Feature/Dataset	SV	BC	FI	BI
All features	0.886	0.747	0.744	0.875
- Word clusters	0.884	0.732	0.714	0.870
- Prototypes	0.880	0.739	0.698	0.870
- Search results	0.881	0.737	0.683	0.868
- Wikipedia	0.880	0.738	0.703	0.871
- Dictionary	0.879	0.705	0.684	0.858
Base model	0.859	0.695	0.667	0.850

However, the comparisons among neural models are inconclusive. Factors that may affect the performance of these models deserve a separate study.

To understand the impact of individual feature groups in our CRF model, we compare the MUC $F1$ scores with word clusters, prototypes, search results, Wikipedia and dictionary features removed (Table 4.8). The results in the last row are from the model only using the basic features. The model without using dictionary yields the most significant decline in MUC $F1$. Removing other feature groups also have some negative, but less influence on the overall performance, showing the overall effectiveness of the features.

In Table 4.9, we show the most significant features from the trained CRF models on each dataset per entity type. The weights correspond to the relative influence of a feature on the result of a model. The dictionary features are dominant for linkable product names for all datasets. Prototype features are more useful for identifying unlinkable product names, while past surface form mentions are the most important features for version extraction. Word substring, shape, and cluster features are useful for all entity types. In comparison, importance of the search results are less permanent, even compared with the query likelihood feature which is used to quantify the quality of a query.

Table 4.10 compares the linking performance of 3 variations of our method (M1-3) with the baselines. In general, M2 and M3 are the top 2 performers in all metrics over all datasets. Remarkably on dataset BI, compared to the lowest fuzzy matching baseline, incorporating version information (M2) increases precision and recall by 74% and 120%, respectively. Improvements are also observed on other datasets too. It shows the usefulness of using version entities for product linking in our task. M2 produces results that are consistently better compared to result

TABLE 4.9: The most effective CRF features with weights.

	SV		BC		F1		B1	
	Feature	Weight	Feature	Weight	Feature	Weight	Feature	Weight
Linkable	dictionary	3.361	dictionary	6.101	dictionary	2.506	dictionary	3.822
	left word	2.871	left dictionary	1.991	left dictionary	1.205	query likelihood	1.295
	wikipedia	1.383	word[:2]	1.926	glove cluster	1.005	word[:1]	0.900
	brown cluster	0.930	left word	1.626	query likelihood	0.899	word shape	0.782
Unlinkable	word[:1]	0.510	query likelihood	1.617	word[:1]	0.845	prototype	0.754
	prototype	1.736	mention	3.395	prototype	0.989	mention	3.317
	mention	1.361	mention lower	3.331	mention	0.980	mention lower	3.278
	mention lower	1.359	query likelihood	1.259	mention lower	0.980	prototype	0.828
Version	glove cluster	0.749	glove cluster	1.109	query likelihood	0.772	query likelihood	0.643
			query likelihood	1.020	postag[:2]	0.275	query likelihood	0.492
	mentions	3.256	mentions	3.547	mentions	1.211	mentions	3.024
	word.endsdigit	1.694	mentions lower	2.883	mentions lower	1.074	mentions lower	3.021
	mentions lower	0.700	ends with digit	1.413	glove cluster	0.670	ends with digit	1.444
	glove cluster	0.638	starts with digit	1.231	brown cluster	0.640	glove cluster	0.949
	w2v cluster	0.599	word[-1:]	1.153	word shape	0.629	right word	0.621

TABLE 4.10: Linking module results. For each column, the best score is in boldface and the second best is underlined.

Dataset	SV			BC		
Method	<i>Pr</i>	<i>Re</i>	<i>F1</i>	<i>Pr</i>	<i>Re</i>	<i>F1</i>
Fuzzy Matching $q = m$	0.874	0.193	0.316	0.341	0.144	0.202
Fuzzy Matching $q = (m, v)$	<u>0.921</u>	0.273	0.421	0.535	0.151	0.236
Fuzzy Matching $q = (m, v, t)$	0.606	0.174	0.270	0.242	0.050	0.083
GLEN [106]	0.872	0.303	0.450	0.581	0.505	0.540
ProdLink [107]	0.119	0.126	0.118	0.503	0.385	0.341
Our method M1: $q = m$	0.856	<u>0.669</u>	0.748	0.723	0.392	0.441
Our method M2: $q = (m, v)$	0.932	0.685	0.783	<u>0.708</u>	<u>0.508</u>	<u>0.557</u>
Our method M3: $q = (m, v, t)$	0.916	0.661	<u>0.757</u>	0.768	0.538	0.607
Dataset	FI			BI		
Method	<i>Pr</i>	<i>Re</i>	<i>F1</i>	<i>Pr</i>	<i>Re</i>	<i>F1</i>
Fuzzy Matching $q = m$	0.047	0.081	0.059	0.174	0.163	0.168
Fuzzy Matching $q = (m, v)$	0.155	0.153	0.154	0.188	0.113	0.141
Fuzzy Matching $q = (m, v, t)$	0.011	0.083	0.019	0.011	0.011	0.011
GLEN [106]	0.515	0.622	0.563	0.670	0.456	0.543
ProdLink [107]	<u>0.588</u>	<u>0.692</u>	<u>0.631</u>	0.284	0.232	0.252
Our method M1: $q = m$	<u>0.588</u>	<u>0.692</u>	<u>0.631</u>	0.433	0.232	0.270
Our method M2: $q = (m, v)$	0.703	0.769	0.734	0.753	<u>0.512</u>	<u>0.588</u>
Our method M3: $q = (m, v, t)$	<u>0.588</u>	<u>0.692</u>	<u>0.631</u>	<u>0.749</u>	0.537	0.624

	SAP Solution Manager 7.1	SAP NetWeaver 7.4	SAP ERP 6.0	SAP Simple Finance
Ticket ID	106	288	196	
Meta Product	SAP SOLUTION MANAGER 7.1	SAP EHP1 FOR SAP NETWEAVER 7.3	SAP ERP 6.0	
Subject	Problem with Downloaded License Data from SAP Support Portal	Error in 'db.xxxxxxxxxxxxxxxx'	sFIN 15xx xxxx xxxx FISCAL YEAR ISN'T CHECKED [S]	
Free Text	We have a problem to download Licence data systems from SAP Support Portal to XXXX in <u>Solution Manager</u> . Systems DATA from our customer XXXX with ...	We are using <u>EP</u> system XXXX based on <u>SAP Netweaver 7.4 SP11</u> Java stack only.	We have migrated to <u>Sfin 15xx SP15xx</u> and we have noticed that ...	

FIGURE 4.3: Outputs from the end-to-end system on 3 tickets from SV, BC, and BI. The true positive results from product and version extraction are highlighted in green and yellow respectively. False negative entities are underlined. Positive linkings are in solid curves and orange boxes, while the false negative linkings are dashed curves. False positive linking is grey.

from M1, which only uses $q = m$. It validates our idea of using ‘version’ as a key feature in the linking task. However, after adding meta-product t into the query (M3), the results are mixed. Specifically, performance on SV and FI are worse than M2. Recall that the datasets only have half as many unique meta-product as the other 2 datasets. It appears that t does not contribute to linking model when many tickets share the same meta-products in a dataset.

4.5.3 Error analysis

To study the end-to-end system output, we assume any version token in a window of flexible size is associated to a product mention m . Specifically, let m be at position 0, the left boundary of the window is $\max(-2, pos_w)$ and the right boundary is $\min(6, pos_w)$, where pos_w is the relative position of another product mention w . In Figure 4.3, we choose 3 test tickets from SV, BC, and FI to illustrate typical outcomes and errors generated from their respective models. Specifically, we focus on the following three types of errors.

Error Type 1: Wrong entity. In Ticket 106, a product mention “Solution Manager” is correctly recognized by the extraction model. Without any version token in its context window, the meta-product is useful for linking the mention to the correct entity, *i.e.*, ‘SAP Solution Manager 7.1’. In the same ticket, a token “DATA” is falsely recognized as a product, largely due to its uppercase shape. The subsequent linking result is incorrect and ignored.

Error Type 2: Correct entity, wrong linking. In Ticket 288, a positive mention “SAP Netweaver” is correctly linked to its entity, using the version mention next to it. However, “EP” is correctly recognized as a product mention but linked to an incorrect product.

Error Type 3: Missed entity Our extraction system fails to extract mentions from Ticket 196. The term “Sfin” is a rare acronym of ‘SAP Simple Finance’. For such case, more investigations are required to generalize the model to recognize rare and new acronyms, thus improving the recall.

4.6 Conclusion

In this chapter, we investigate the software product name extraction and linking problems from support tickets. Through manual annotation and analysis, we design domain features from local, contextual, as well as external resources. We demonstrate that, with these deliberately features, the linear models are able to outperform the increasingly popular neural network models, especially when labels are sparse. We use this method to obtain a domain lexicon, and use it for routing in the next chapters.

Chapter 5

Unified Framework for Ticket Routing⁹

5.1 Introduction

In this chapter, we combine the knowledge and output from the Chapter 3 and Chapter 4, and propose a routing framework. Recall that, an incident is defined as an event that prevents users from performing their task. It could be due to a system fault, an access request, or a lack of user knowledge. Incident ticket is an electronic log containing information about the incident. Figure 1.2 depicts a high-level ticket processing workflow. Firstly, the user creates an incident ticket, either directly or by contacting the helpdesk. Each newly created ticket needs to be matched with the expert group in charge of processing it. An expert group is a unit of supporting staff who are experts in certain areas. If the first assigned group solves the problem, the ticket is resolved (or closed). Otherwise, the routing system needs to transfer the ticket to another group for processing until the ticket is resolved. Initial group assignment and inter-group transfer are individually studied as problems of *routing recommendation* [6, 10, 34], or *expert recommendation* [94, 134]. In this work, we consider both collectively as parts of the larger *ticket routing problem*.

In this chapter, we conduct a study in a large software support organization with high ticket volumes. Based on our preliminary analysis, we propose a Unified Framework for Ticket Routing (UFTR). It extends the Learning-to-Rank paradigm

⁹The work in this chapter has been published in [133].

TABLE 5.1: Comparison of UFTR with existing routing systems in terms of feature types and models.

System	T	G	TG	GG	Assign	Trans
EasyTicket [10]	-	-	-	✓	-	✓
AIM-HI [79]	✓	-	-	-	✓	-
Hybrid model [80]	✓	-	-	✓	✓	-
GenerativeModel [6]	✓	-	✓	✓	-	✓
ITSCM [94]	✓	-	-	✓	✓	-
TER [134]	✓	✓	✓	✓	Two stages	
UFTR (this chapter)	✓	✓	✓	✓	Unified	

using four types of features modeling the individual and interactive information from tickets and expert groups. It unifies initial group assignment and inter-group ticket transfer, incorporating findings from most of the previous works as features.

The remaining of this chapter is organized as follows: Section 5.2 summarizes previous works related to ticket routing. Section 5.4 presents exploratory data analysis for key features. In Sections 5.5 and 5.6, we describe the architecture and compare UFTR with baseline models. We conclude this chapter in Section 5.7.

5.2 Related Work

Earlier efforts at solving the ticket routing problem utilized graphical, generative, classification, and retrieval methods. We first summarize existing routing approaches, before discussing related work in ticket classification and retrieval.

One of the first formulations of the problem was from Shao *et al.* [5, 10]. The authors built a graph-based Markov model for the ticket transitions among expert groups. The main idea was to apply frequent pattern mining techniques to find transfers patterns from archived histories. The extracted patterns are used for more efficient routing. The author also investigated the usefulness of intermediate transfer groups in the routing process. They argued that some groups act as distributors and are more important to final routing to resolver. However, the authors did not consider ticket content and the initial group assignment.

Subsequent work attempted to integrate the contents in the routing model in different ways. Sun *et al.* [80] first filtered a subset of nodes based content similarities,

then applied similar routing algorithms by Shao *et al.* [5] on the reduced graph. Miao *et al.* [6] proposed a generative Optimized Network Model (ONM), leveraging information from both ticket contents and transfer histories. It captured the hidden connections between terms in a ticket to the transition probabilities to next group. Both models required an initial group as input together with the ticket. More investigations [31, 82, 85] into the ticket transfer patterns were made, in slightly different problem setting and data set. It was found that expert groups' expertise and awareness of others' profiles are the main factors contributing to decision making. However, ticket routing was not the primary task in these works.

Recently, Xu and He [134] proposed the Two-stage Expert Routing (TER) model. In the first stage, the initial group was determined by taking the nearest neighbor in a distributed expert group representation model, as per [30]. A ticket was projected to the same vector space as the expert groups for distance-based selection. Subsequent routings follow the transition model which did not consider ticket features. That is, models for the two stage are trained separately using different information.

In Table 5.1, we summarize existing routing systems in terms of the features used in their systems. From the table, one can observe that most of these models leverage partial information from the feature types we identified, except TER. In comparison, UFTR use all types for features to better capture the individual and interaction information between tickets and expert groups. Note that UFTR is different from TER, as the latter uses two-stage models, where UFTR is a unified framework.

UFTR incorporates findings from previous studies in ticket routing, classification, and retrieval. Ticket features are researched in all these tasks, which are generalizable to a few feature types. In addition, we explore network features for expert groups as well, and further Ticket-Group and Group-Group interaction features.

5.3 Preliminaries

In the organization we study, expert groups are basic units for ticket processing. They are hierarchically structured in trees, where each tree represents a functional area. We define that an expert network consists of a set of expert groups $\mathcal{G}$. Here,

FIGURE 5.1: AC and PM are the roots for “Accounting” and “Process Management” products. A ticket is being routed from leaf group $AC \cdot BE \cdot DB$ to $PM \cdot BE \cdot DB$, where BE and DB are “back-end” and “database” components of the root product. UI stands for “user interface”.

we consider two levels of abstractions: the *leaf* level (g^{leaf}) is the group itself, and the *root* level (g^{root}) is the root node of the hierarchy the group belongs to. For example, in Figure 5.1, two group trees have AC and PM as roots, respectively. A routing is taking place at leaf level from group $AC \cdot BE \cdot DB$ to $PM \cdot BE \cdot DB$. At root level, the routing is from AC to PM.

A routing sequence is a list of expert groups that are involved in the routing process of a ticket. The group in each step of the routing is a participant group ($g_{participant}$). A participant group may play different roles when it is involved in processing a ticket. The first group in a routing sequence is the initial group ($g_{initial}$) a ticket is assigned to. The group that closes the ticket is the resolver ($g_{resolver}$). A ticket may take only one routing step to close; that is, the initial and resolver groups are the same group.

Problem Statement. Instead of modeling initial group assignment and ticket routing as separate tasks, we consider *ticket routing as a unified problem*. Give a ticket τ and the current routing sequence S_t , the task is to recommend a ranked list of expert groups for human to decide which one to route to. S_t contains all expert groups who have worked on τ , but fails to close it. When a new ticket is received, $S_t = \emptyset$.

FIGURE 5.2: Ticket length distribution in number of terms, and routing sequence lengths (right) of 500,000 archived tickets.

5.4 Exploratory Analysis

We randomly sample 500,000 archived incident tickets from an incident management system. Figure 5.2 shows ticket length distribution in number of terms and the distribution of routing sequence length. The majority of tickets have fewer than 100 terms. The length of routing sequence follows a power law distribution, with most tickets having length of 1. Specifically, more than 50% of the tickets are resolved by the first assigned group without routing. This observation is non-trivial. Previous works [5, 6, 134] taking current processing group as input do not support this majority of cases in our data. Next, we attempt to answer the following research questions through more detailed analysis.

- **RQ1:** Does clarity of a ticket correlate with its routing steps?
- **RQ2:** Are some groups having higher chance of being resolvers?
- **RQ3:** For resolver groups, how often do they receive tickets from other groups, and from dispatch?

5.4.1 Ticket Content Analysis

A ticket contains rich information for human experts to diagnose the incident. In our data, we observe high usage of technical terms/concepts and variant representations of the same concepts and entities.

Technical Entities. In an earlier study, Han *et al.* [99] investigated product name recognition and normalization in tickets. In this study, we extend the findings to a

FIGURE 5.3: Distributions of clarity scores of archived tickets, grouped by the number of routing steps.

more general notion of *technical entities*, including product names, technical terms and concepts. For example, in a ticket which reads “We are in **ERP6.0 NW7.0 SP6** running on **windows OS 2003** & **database version** is **MSSQL2008**. We did the **DB** restore for the **LE1** system from **PE1** system. After finishing **ERP** shows **error message...**”, the technical entities are in boldface.

After determining the entities, tickets can be represented using a “bag-of-technical entities” model. It has several benefits. Technical entities are more semantically representative and the number of dimensions are reduced by an order of magnitude (*e.g.*, from 400 thousand to 40 thousand in our case). Further, sensitive personal information is automatically filtered out from the representation. This is a crucial requirement by General Data Protection Regulations (GDPR) ¹⁰. In this work, we assume the technical entities are readily extracted. ¹¹

Technical Clarity (for RQ1). With “bag-of-technical entities” representation, we further develop a technical clarity score to evaluate topic specificity of a ticket. Cronen-Townsend *et al.* [135] proposed a clarity score to predict query difficulty using the relative entropy between a query language model and the collection language model. We replace words with the technical entities in the original model to

¹⁰<https://eugdpr.org/>

¹¹The details of the extraction process are out-of-scope.

FIGURE 5.4: Log probability distribution of expert group being different roles, and the correlations of probability distribution of the same group playing different roles.

compute a technical clarity for a ticket τ . Formally,

$$\text{Clarity}(\tau) = \sum_{e \in E} P(e|\tau) \log_2 \frac{P(e|\tau)}{P(e|C)},$$

where E is the set of entities in the collection and C is the collection of tickets.

Figure 5.3 plots the distribution of technical clarity of tickets, grouped by their number of routing steps. We observe that tickets with shorter routing sequences have higher mean clarity than those with longer sequences, although the variance is relatively higher.

5.4.2 Expert Group and Network Analysis

Statistical Analysis (for RQ2). With a large number of expert groups in the network and transfer histories, we first characterize groups from statistical perspective. Specifically, we investigate the probability distribution of different roles in transfer histories and study the pairwise correlations. We use three random variables to model the probability of a group as the initial group ($P_{\text{initial}}(g)$), the resolver ($P_{\text{resolver}}(g)$), or participant ($P_{\text{participant}}(g)$) in routing. Note that the participant role includes both initial and resolver group, and other groups which

are in a routing sequence. The values are estimated¹² by $\frac{\#(g \text{ being initial group})+1}{|\mathcal{T}|+|\mathcal{G}|}$, $\frac{\#(g \text{ being resolver group})+1}{|\mathcal{T}|+|\mathcal{G}|}$ and $\frac{\#(\text{All tickets involves } g)+1}{|\mathcal{T}|+|\mathcal{G}|}$, where $|\mathcal{T}|$ and $|\mathcal{G}|$ are total number of tickets and groups, respectively. Figure 5.4 shows the log probabilities in its first row. The histograms show slight right-skewed normal distributions for all three variables. For each variable, we compute skewness and kurtosis, which are statistical descriptors for measuring the symmetry and tailedness of a distribution with respect to a normal distribution. Among the three variables, $\log(P_{\text{initial}}(g))$ is the least skewed but has highest kurtosis, due to infrequent deviations or outliers in the data distribution.

The subplots in second row show pairwise correlations of probability distribution of the same expert group playing different roles. Both x - and y -axis corresponds to expert groups sorted by their probability of being involved in a routing process. We observe a stronger correlation of *initial-participant* and *resolver-participant* compared to *resolver-initial*. In *initial-participant* and *resolver-participant*, the data points below the regression line corresponds to groups which have lower probabilities of being the initial group or resolver, than being involved in routing. Remarkably, in *initial-participant*, there is a cluster of groups that have much lower probabilities to be the initial group than they would be involved in any role. The same pattern does not prevail in *resolver-participant* subplot. Quantitatively, this difference is reflected by the higher kurtosis of $\log(P_{\text{initial}}(g))$ compared to $\log(P_{\text{resolver}}(g))$. In *resolver-initial*, the data points are scattered over a wider area on both sides of the regression lines. This shows some groups are more likely to be the resolver than to be the initial group, and vice versa. Note that the spike at the left end of the x -axis, there are groups that are rarely the initial group, but more likely to be the resolver.

Source to Resolver Analysis (for RQ3). Since our main objective is to recommend resolvers, we are interested in the statistical characteristics for routing to resolvers. From historical routing sequences, we extract and model each resolver group (g_{resolver}) as a probability distribution of the groups in the previous routing step *i.e.*, source to resolver. Since majority of the tickets are solved in one step, we fix the first source for each resolver to be from “dispatch” and estimate the probability. Similarly, we estimate the probability for each source and rank with descending order *for each resolver*. The resultant distributions are shown in

¹²A smoothing factor is added in the estimation.

(A) Distribution of leaf groups

(B) Distribution of root groups

FIGURE 5.5: Source to resolver analysis. The left end of the x -axis is fixed for routings from dispatch. Each line characterizes one expert group with the distribution of cumulative sources to resolver. The colored lines are the cluster centers (best viewed in color).

Figure 5.5. Observe that with “dispatch” source fixed at the left end of x -axis, the source to resolver distribution exhibits distinctive shapes, which we refer to as *fidelity*. We further cluster the distributions using the k -means algorithm into 3 clusters, and show the cluster centers in colors. A large majority of the resolvers receive tickets from mainly one source, corresponding to a power law cluster center

TABLE 5.2: Statistics and characteristics of networks H_{trans} , H_{res} , and H_{root} .

	H_{trans}	H_{res}	H_{root}
# of nodes	3,390	3,390	57
# of edges	44,719	82,904	1,390
Network Diameter	10	8	3
Average degree	14.161	24.455	24.123
Average weighted degree	0.996	0.735	0.599
Average path length	3.299	2.978	1.592
Average clustering coefficient	0.218	0.268	0.722

distribution with steep slop. We label this cluster as *high fidelity* (green line). The second cluster has *medium fidelity*, showing a smooth distribution of sources over the rank (orange line). Lastly, there is a cluster of resolvers with *low fidelity* (blue line). That is, they receive tickets mainly from a source other than from dispatch.

To understand the distribution patterns at the root level (see Figure 5.1), we repeat the process for the root group ($g_{resolver}^{root}$). As shown in Figure 5.5, only two root groups still show low fidelity. An important finding is that a majority of the resolvers receive tickets from dispatch or *another group within the hierarchy with the same root*. Further investigation reveals close business connection between the two low fidelity groups (finance and accounting). This shows that groups from different roots are also connected.

Network Analysis. To understand the inter-group connections, we define three directed graph $H = (\mathcal{G}, R)$ to model the group interaction, where R is the set of edges. The first graph, H_{Trans} , is modeled following [10]: an edge $g_i \rightarrow g_j$ means ticket transfer from g_i to g_j and the edge weight is the cumulative weight normalized over the out-going degree of g_i . However, H_{Trans} does not differentiate the role of resolver in routing, as edges correspond to all routing pairs. We argue that all routing groups in a routing path are related to resolver to different extent. To this end, we construct another network, H_{Res} . In H_{Res} , for a routing path, we connect all previous routing groups to the resolver. The edge weights decrease as further away the group from the resolver. Meanwhile, we also remove edges among previous groups (as our goal is to recommend resolver). For example, given routing sequence $A \rightarrow B \rightarrow C$, there are two edges in H_{Trans} , $R = \{(A, B, 1), (B, C, 1)\}$ with equal weight. In H_{Res} , the edges and weights become $R = \{(A, C, 0.5), (B, C, 1)\}$.

The last graph we build is H_{root} , which is H_{Res} at root level, *i.e.*, aggregating leaf nodes to their roots.

Table 5.2 summarizes key statistics of 3 networks. The values show that a significant number of routings are originate from a group in the same root. With the same number of nodes, H_{Trans} is more sparse than H_{Res} , having only half of the number edges H_{Res} has and smaller average degree. As H_{Res} is more connected, routing models based on it have more options from a node to potential resolvers. The differences in network diameter and average path length measure the longest, shortest, and the average path. Both measure are reduced together with the number of nodes at the root level network, indicating a decreased level of interaction at the root level. Average clustering coefficient measures how likely the nodes in a graph tend to form a cluster together. When the coefficient is low (H_{Trans} and H_{Res}), the nodes are more likely to form many clusters, based on their connections. The higher score for H_{root} suggests that the nodes are more closely connected. We will use these as features when designing the routing model.

5.5 The UFTR Framework

After analyzing the ticket contents and group networks, we elaborate the UFTR framework and features in details. In UFTR we aim to process a ticket with assignment (*i.e.*, as a dispatcher) and routing (*i.e.*, as recommendations to an expert group who cannot close the ticket). That is, we need to match the ticket (which may or may not have existing routing sequences) to expert groups. We formulate this problem as a ranking problem, *i.e.*, to rank the most suitable expert groups. However, given thousands of expert groups, it is costly and inefficient to consider all expert groups as candidate resolvers. To this end, we propose first to select candidate resolvers, then to extract features, and finally to rank the expert groups. As illustrated in Figure 5.6, our UFTR framework consists of three main components. A classification model for **candidate resolver selection** (Section 5.5.1). **Feature extractor** module (Section 5.5.2) extracts 4 types of features from a $\langle ticket, current routing sequence, candidate group \rangle$. Lastly, the **learning-to-ranking** model (Section 5.5.3) recommends top-ranked candidate groups for human to decide which group to route to.

FIGURE 5.6: System architecture of the UFTR framework. Components at the upper region are artifacts or models produced from offline training processes, where are not shown in the diagram for simplicity. The lower region contains the main steps in online operation. Online and offline components exchange data through an API gateway.

FIGURE 5.7: Root level prediction using a unsupervised retrieval-based classifier, expanded with n most connected neighbours in H_{root} . The dashed line indicates the number of cases having the resolver among the candidate, over all tickets (*i.e.*, hit rate). The blue bar represents the ratio of candidates after expansion, over all expert groups in the network H_{res} .

5.5.1 Candidate Resolver Selection

Given a ticket, a subset of expert groups are first selected as candidate resolvers. As shown in Section 5.4.2, a majority of the transfers happen between groups within a hierarchy with the same root. Intuitively, we attempt to make use of the hierarchy to filter the candidates for the next stage of processing.

The idea is to select a root and use the groups from it as candidate groups. It can be modeled either as a ticket classification or retrieval task. Training a supervised classifier involves data set preparation, input vectorization, algorithm selection, and model fitting. To focus on the ticket routing problem, we implement a ticket retrieval-based filter which does not require any training. We first index every ticket as a document using Apache Lucene¹³. Next, we retrieve similar tickets, by using a ticket as query and fuzzy similar query matching built-in by Lucene. From top-100 returned tickets, we aggregate the corresponding resolver roots by majority voting to select the candidate root.

To increase recall for the ground truth resolver, we expand the candidate set by leveraging the network information in H_{root} . Specifically, we find the neighboring root nodes of a predicted root in H_{root} . Recall that, the edge weights model the

¹³<http://lucene.apache.org/>

probability of ticket relevance from a root to a resolver root. Sorting the neighboring nodes by weight of edges, we take the top- n roots for candidate expansion. Figure 5.7 shows the performance of the combined strategy on our data. The figure shows that hit rate is higher than 0.95, with less than 40% of all groups covered in candidate sets, when 10 neighboring roots are selected.

5.5.2 Feature Extraction

Languages used in tickets are highly specific and technical. We harness an entity-oriented [136] ticket representation to bridge the unstructured text to domain vocabularies. We assume that entity extraction is performed in preprocessing with existing methods [99, 137]. We propose four types of features specific for our domain problem, and all features are numerical and normalized to $[0,1]$.

Ticket features. We use the following ticket statistics as features, *e.g.*, length, number of technical entities, and their ratio. From all archived tickets, we compute inverse document frequencies for entities (IEF) and an overall entity language model. Additional features can be derived by summing up the IEF for all entities in a ticket, and technical clarity discussed in Section 5.4.1.

Group features. From network of groups H_{res} , we obtain a rich set of network features to characterize each group, *e.g.*, the in-, out-, and total degree of a node. These features are analogous to the transition probabilities in [10]. However, we focus on the connection to resolver instead of direct routing. From the network, we can also compute hub and authority scores for the nodes [138]. Other features include eccentricity, closeness centrality, harmonic centrality, betweenness centrality, Eigenvector centrality and PageRank [139], and cluster coefficient. Besides the network features, we also include the prior probabilities, $\log(P_{participant}(g))$, $\log(P_{initial}(g))$, and $\log(P_{resolver}(g))$.

Ticket-Group features. This type of feature considers ticket τ and group g as a potential resolver. One example is conditional probability $P(g_i|\tau)$. Using the entity-oriented representation, it can be evaluated as *i.e.*, $P(g_i|\tau) = p(g) \prod_{e_i \in E_\tau} p(e_i|g)$. Another feature considers the similarity between τ and the tickets previously closed by g , denoted by T_g . For ticket representation, we use technical entities and word

embeddings. The former represents τ and T_g as sparse vectors where each dimension corresponds to a technical entity. The latter is learned by FastText [140] with skip-gram on our domain corpus. Specifically, we use the average word embedding to represent a ticket and T_g . Cosine similarity is computed for both sparse vector and embedding vector representations. Lastly, three IR-based relevance scores are computed by query likelihood model, BM25, and sequential dependency model, considering τ as a query and T_g as documents.

Group-Group features. This type of features are used primarily in transition-based routing models [5, 6]. Given a current processing group g_i , we estimate the probability of a ticket being transferred to another group g_j , from historical data. The *first-order multiple active state search* (FMS) and *variable-order multiple active state search* (VMS) routing models are adopted [5]. In addition, we also consider collocation information for two groups, indicating the association of $g_i \in S_\tau$ to g_j if g_j is the resolver. S_τ is the current routing sequence and S_τ is empty for a new ticket. In this case, the only information for this group is the probability from dispatch in history; it can also be seen as the prior probability for each group to be the resolver, *i.e.*, $P(\emptyset \rightarrow g) = P(g)$.

5.5.3 Learning to Ranking

Learning-to-Rank is the state-of-the-art model in many retrieval tasks [141]. It has also been used for recommending assignee for bug reports [142]. Given a query, a system ranks a set of documents according to their relevance to the query. To learn the ranking, a query-document pair is represented by a multi-dimensional feature vector. It is flexible to incorporate a large number of features from multiple sources, through feature engineering.

Learning-to-Rank for Ticket Routing. Features extracted from each $\langle \text{ticket}, \text{current routing sequence}, \text{candidate group} \rangle$ triplet are represented as a feature vector. The training samples consist of $\langle \text{ticket}, \text{current routing sequence}, \text{candidate group}, \text{relevance} \rangle$ quadruplets, where the candidate group is either from the ground truth routing sequence or random sampling. From a sample we create a training instance $(\mathbf{x}_{\tau, S_t, g_{t+1}}, r_{\tau, S_t, g_{t+1}})$ in which $\mathbf{x}_{\tau, S_t, g_{t+1}}$ is a multi-dimensional feature vector, and $r_{\tau, S_t, g_{t+1}} \in \mathbb{R} : 0 \leq r_{\tau, S_t, g_{t+1}} \leq 1$ is the relevance score. Note that,

TABLE 5.3: Features for routing model. τ is the ticket, E_τ is the entity set contained in τ . g is the candidate group, while S_τ is the current routing sequence. When a new ticket is being processed, $S_\tau = \emptyset$.

Type	Feature	Description
T	$ \tau $	Length of ticket τ
	$\sum_{e_i \in E_\tau} c(e_i, \tau)$	Total occurrences of technical entities in ticket τ
	$\sum_{e_i \in E_\tau} c(e_i, \tau) \times 1/ \tau $	Ratio of technical entities in ticket τ
	$\sum_{e_i \in E_\tau} IEF(e_i)$	Sum IEF of technical entities in ticket τ
G	Clarity(τ)	Technical clarity of ticket τ
	$P_{participant}(g), P_{initial}(g), P_{resolver}(g)$	Prior probability of g in any routing, as initial group, and as resolver.
	$deg^-(g), deg^+(g), deg(g)$	Unweighted indegree, outdegree and total degree of group g
	$deg_{weighted}^-(g), deg_{weighted}^+(g), deg_{weighted}(g)$	Weighted indegree, outdegree and total degree of group g
G	Harmonic, Closeness, Betweenness centralities	Measures of how close is g to all other groups.
	Eigenvector centrality, PageRank, Hub, Authority	Measures of the quality of g voted by other groups.
	Cluster Coefficient	Cluster coefficient of g
TG	$P(g \tau)$	Likelihood of g being the resolver of τ
	$\cos_{ent}(\tau, T_g), \cos_{emb}(\tau, T_g)$	Cosine similarity between ticket τ and group g and tickets solved by g .
	Distance(T_{emb}, g_{emb})	Euclidean distance between τ and g in embedding space.
GG	QLM, BM25, SDM	Scores from query likelihood model, BM25, and sequential dependency model.
	$ S_\tau $ $P_{FMS}(S_\tau \rightarrow g), P_{VMS}(S_\tau \rightarrow g), P_{coll}(S_\tau, g)$	length of current routing sequence, 0 for dispatch Transition probabilities and collocations.

a ticket with multiple routing steps are converted to multiple training instances, the non-resolver groups have discounted relevance score as in the way we construct H_{Res} . Specifically, we assign the distance-based scores and generate positive training samples by pairing a ticket with every group in its routing sequence. Each pair is assigned with a value $(0, 1]$ relative to its distance to the resolver; 1 is assigned to resolver and the score reduces by half as one step further from the resolver. For every positive sample, we randomly select an expert group that is not involved in the routing as a negative sample, *i.e.*, setting the relevant score to 0.

We implemente UFTR with both pointwise and pairwise learning-to-rank for group ranker. Specifically, we train a Random Forest Regressor (RFR) [143] with 200 regression trees and bootstrapping for the former approach. The benefit of such model is its explainability. The model scores each feature by its impact to the final scoring. For pairwise approach, we employ LambdaMART [144], one of the state-of-the-art algorithms for learning-to-rank. It also leverages regression trees, with gradient boosting. As the listwise approach is computationally challenging to implement, we leave it for future works.

5.6 Experiment

To test the effectiveness of UFTR, we evaluate its performance in Assistive Routing evaluation. Specifically, given a test ticket from testset, we want to measure how does the framework rank the true resolver among non-resolvers. In addition, we also study its performance in resolver ranking, and investigate contribution of individual features to the results.

5.6.1 Experiment Setup

Dataset. We select archived tickets with routing sequences for evaluation. From all tickets used for analysis in Section 5.4, we randomly select 58,670 (11.7%) tickets for training. From the selected training tickets, we generate 150,000 training instances with positive and negative ratio of 1:1, as the training set. The remaining tickets are grouped by routing steps and 500 are randomly selected from each group having n ($n \leq 4$) steps. Effectively, we created four test sets, S1, S2, S3, and S4.

Evaluation metrics. For each test ticket, we simulate a routing process assisted by our ranking model. Initially, the routing sequence is empty and the model recommends a list of candidate groups as potential resolvers. Depends on the evaluation metric, if the ticket is considered not resolved, routing continues. The performance is evaluated using the following metrics: Mean Steps to Resolver (MSTR), Resolution Rate (RR), and Mean Average Distance to Resolver (MADR@ k).

MSTR is the average length of test routing sequences. The closer the value is to 1, the more effective a system is. In experiment, for each test ticket, we consider the top-1 ranked group recommended by a system in each routing step. The recommendation is successful only if the ground truth resolver is correctly recommended and all previous recommendations are counted for MSTR computation. For each test set, an effective routing solution should be lower than the average archived routing steps. We assume no ticket would take more than 10 steps in this experiment. RR measures the ratio of resolved test tickets within n recommended steps. Given n , more tickets is resolved, the better a system is. Mean MADR@ k scores each routing step with respect to the ground truth routing sequence. At each routing step, a system generates top- k recommendations. The routing is successful if the resolver is among the top- k . Otherwise, the next group in the ground truth routing sequence is added to the routing sequence. In this way, the process incorporates more human decisions in the archived tickets.

Baseline models. As discussed in Section 5.2, the Two-stage Expert Recommendation (TER) framework [134] only use four types of features that are comparable to UFTR. We re-implement TER as a baseline. The first stage selects the group with minimal distance to the ticket in embedding space. The group representations are learned using the model proposed by [30], using the tickets a group has resolved as positive samples and randomly selected negative samples. The second step follows the transition based model as proposed by [5], using group-group transition probabilities modeled in H_{Trans} . By fixing the first step as proposed and only changing the second stage, we have the following three routing baselines¹⁴.

¹⁴We also experimented with the generative model proposed in [6]. Unfortunately, we are unable to reproduce sensible results as the model is untractable on our data. The number of parameters to be optimized are $\mathcal{O}(VTE^2G^2)$, where V is the vocabulary size, T is size of training set, E is the number of edges in the network, and G is the group count.

TER-FM is a first-order memoryless search model using a greedy approach. The next group g_{t+1} is selected using

$$g_{t+1} = \operatorname{argmax}_g P(g|g_t), \forall g \in \mathcal{G},$$

where g_t is the current group.

TER-FMS is a first-order multiple active state search model, considering groups in the current routing sequence instead of only the current group. The next group g_{t+1} is selected using

$$g_{t+1} = \operatorname{argmax}_g P(g|g_r), \forall g_r \in S_\tau, g \in \mathcal{G} \setminus S_\tau,$$

where S_τ is the set of groups that is in the current routing sequence.

TER-VMS is a variable-order search algorithm, using a higher order Markov model. The next group g_{t+1} is selected using

$$g_{t+1} = \operatorname{argmax}_g P(g|S_k), \forall g \in \mathcal{G} \setminus S_\tau, S_k \subseteq S_\tau,$$

where S_k is a subset of the current routing sequence.

5.6.2 Evaluation on Ticket Routing

Ideally, UFTR can be used to rank all candidate groups with the trained model on every route step. For the purpose of quick experimentation, we adopt the *leave-one-out* test scheme that is widely used in evaluating recommendation systems [145, 146]. Specifically, for each test ticket, other than the resolver group, we select 49 other groups to form 50 pairs for feature generation, and ranking. Some of the groups are from the archived routing sequence, and rest are randomly sampled from the candidate set produced by root prediction and expansion with 10 neighbor roots (see Section 5.5.1).

Figure 5.8a reports the results of UFTR against baselines in three metrics. UFTR is the lowest across all four test sets in terms of MSTR. For S1, the scores are dominated by the results of initial group assignments. Our classification with expansion based approach performs better than the TER-based systems which use

(A) Mean Step To Resolver

(B) Resolution Rate

(C) Mean Average Distance to Resolver

FIGURE 5.8: Performance comparisons of UFTR with baselines in evaluated by different metrics.

a distance-based initial group assignment approach. In S2, UFTR reduces MSTR from 2 to 1.4. TER-VMS also cuts on the archived routing length by 0.4, being the second performing model. MSTRs yield from the other two baselines are even longer than those of the archived routing, *i.e.*, human routing. In S3 and S4, all four systems exhibit some degree of MSTR reduction, with UFTR performed the best. Among the TER-based systems, the TER-VMS model performs slightly better than TER-FM and TER-FMS.

Figure 5.8b shows the changes in resolution rate as routing steps increase from 1 to 10. A monotonic increase of RR is observed from all systems, whereas UFTR achieves the best. The growth of RR for UFTR is also more quickly, compared to the baselines.

Figure 5.8c reports MADR. A human routing baseline is computed by averaging the scores of ground truth routing sequences. The x -axis is the value of k , which is the top positions to consider a hit if the ground truth resolver is in. As k increases, the criteria of finding a resolver is relaxed. Our system has the steepest incline as k increases from 1 to 10, approaching the theoretical best score of 1. The baseline systems show slower improvement over human routing, among which TER-VMS is better than the others. To conclude, UFTR is shown to be more effective than the baselines, evaluated by all three metrics.

5.6.3 Evaluation on Resolver Ranking

In this experiment, we examine how well our model ranks the resolver among other candidate groups, and the importance of features. For each test set, Table 5.4 presents the hit rates of resolver within the top- k ranking position, $k \in 1, 3, 5$. The first model uses all features, and the rest of four models are trained by removing one type of features each time. The hit rates increase significantly as k increases, reaching over 0.97 when all features are used when $k = 5$. It shows that our model is capable of ranking the resolver to the top positions among non-resolvers. The hit rates on S2 is slightly better than those on S3, but came short comparing to the results on S1, especially at top-1 position. As S2 and S3 contain tickets that are resolved by 2 and 3 steps respectively, the non-resolver groups may be similar to the resolver group in many aspects. In this case, the model is unable to identify the resolver accurately. However, on S4, the top-1 hit rate is higher than both

TABLE 5.4: Hit Rate (HR) of UFTR for the task of resolver ranking on four test sets.

Features	Test set S1			Test set S2		
	HR@1	HR@3	HR@5	HR@1	HR@3	HR@5
All Features	0.792	0.950	0.972	0.674	0.942	0.978
- less T	0.720	0.938	0.968	0.602	0.928	0.976
- less G	0.740	0.946	0.948	0.592	0.928	0.974
- less TG	0.290	0.494	0.588	0.088	0.226	0.362
- less GG	0.746	0.948	0.972	0.620	0.932	0.974

Features	Test set S3			Test set S4		
	HR@1	HR@3	HR@5	HR@1	HR@3	HR@5
All Features	0.672	0.938	0.978	0.700	0.936	0.970
- less T	0.566	0.884	0.960	0.664	0.906	0.954
- less G	0.580	0.898	0.960	0.692	0.918	0.968
- less TG	0.106	0.234	0.372	0.096	0.218	0.368
- less GG	0.650	0.920	0.960	0.708	0.916	0.970

S2 and S3, only after S1. This suggests that the longer routing sequences in the archived tickets may be less necessary.

For models trained using one less feature type, the hit rate decreases for all k values. The most decline is observed in HR@1. Noticeably, when Ticket-Group (TG) features are excluded, the hit rates drop drastically. The results of this ablation study show that Ticket-Group features are the most important among all feature types. Previous study [134] concluded that ticket content features are more important than routing sequence in routing task. Though the discrepancy could be due to differences in data distribution and problem setup, it is worth to investigate more features to model the interaction of a ticket and an group in future work.

5.6.4 Feature Importance

Knowing that Ticket-Group features are the most important type of features, we want to further understand individual feature’s contribution to the final results. The RFR model is able to output a weight indicating feature importance. Table 5.5 lists the top-10 features that are ranked by this weight, among all features across the four types. The three most weighted features are all from the Ticket-Group type. The score by query likelihood model, the conditional probability of the group given

TABLE 5.5: Most important features and their types, ranked by importance scores from the Random Forest Regressor.

Rank	Feature	Importance	Type
1	QLM	0.351	Ticket-Group
2	$P(g \tau)$	0.273	Ticket-Group
3	BM25	0.013	Ticket-Group
4	Clarity	0.011	Ticket
5	$\sum_{e_i \in E_\tau} c(e_i, \tau) \times 1/ \tau $	0.009	Ticket
6	cos_{ent}	0.008	Ticket-Group
7	$cos_{emb}, \sum_{e_i \in E_\tau} c(e_i, \tau)$	0.007	Ticket-Group, Ticket
8	$P_{resolver}(g)$	0.006	Group
9	ClusCoef, $P_{VMS}(S_\tau \rightarrow g)$	0.005	Group, Group-Group
10	$ \tau , deg_{weighted}^+$	0.004	Ticket, Group

a ticket, and score by BM25 are with great importance. Among features from other types, ticket clarity appears to be the most useful. On the other hand, the Group features carry less weights in general. In Group-Group features, the probability of being a resolver, and the transition probability under the VMS strategy, are relatively more important.

5.7 Conclusion

In this chapter, we propose a unified framework for ticket routing. We first analyze the contents and group characteristics of half a million archived tickets. Then we propose four types of features to represent different aspects of the routing problem. The features are used as the input to a regression based ranking model. Unlike previous solutions that use different models for initial group assignment and transfer, UFTR uses a single model for both subproblems. We show it is more effective in terms of all three routing evaluation metrics, compared to baseline models. Through an ablation study, we find that the Ticket-Group features, which model the interaction between a ticket with a group, are the most contributive features. The last finding motivates us to apply deep neural network models to model the same interaction information, which we will discuss in the next chapter.

Chapter 6

Deep Neural Network Approach for Ticket Routing¹⁵

6.1 Introduction

In an enterprise software support services, incidents and change requests are communicated and recorded using *tickets*. A ticket is created by an end-user or a helpdesk personnel, and is then processed by experts. For the support organization, resolving a ticket promptly and correctly is a KPI to fulfill SLA with their customers. To achieve this, the right expert needs to be identified for a ticket. This is challenging when the problem description is unclear, also when there is a large number of service offerings and human experts.

Ticket routing takes place in two scenarios, as illustrated in Figure 1.2. Firstly, when the ticket is created and received by the system, and the first processing group needs to be assigned. In some systems, this can be manually selected by the end-user during ticket submission. For experienced users, being specific on who should process their tickets is an ease decision. But for new and inexperienced users, making such decision may be challenging. The second scenario happens when the current processing group is not able to resolve the ticket. A next group will be chosen by the current group, depending on their diagnosis of the problem and knowledge about the other groups. Though many service providers follow a

¹⁵The work in this chapter has been published in [147].

similar ticket processing workflow, some may have variations in their systems, *e.g.*, deployment of dispatchers who makes centralized routing decisions.

In both scenarios, human routing requires considerable level of expertise. The chance of finding the resolver is affected by an individual’s previous experience, interpretation of the problem, as well as the familiarity with other expert groups. Routing a ticket to a wrong group would result in longer turnaround time, higher cost, and customer dissatisfaction. This calls for intelligence solutions to aid the routing process, leveraging archived tickets and routing paths. In the previous chapter, we propose a unified routing framework, leveraging manually crafted features from different feature types. Our experiments show that the features that model the interactive relationship between the ticket and expert group representations, are the most contributive to the performance gain in routing.

Recent studies show neural network models are effective in modeling text similarities [145, 148–152]. These models automatically generate features from input data, and are able to scale with large amount number of training instances. In this chapter, we leverage deep neural networks for automatic feature generation and compute a relevance score for a ticket and candidate resolver pair. The joint relevance score is a weighted sum of two individual scores, computed in a multi-view architecture. Specifically, in the text-view of both ticket and expert group, we use a classic semantic text matching model to obtain the text relevance. The output from graph-view is expert group representation vector in the same size as the ticket vector for relevance computation. The input are pre-trained embeddings using routing information as network structure, and ticket text as node features. The joint scores are used to rank the candidate resolvers at every routing step. To the best of our knowledge, this is the first attempt to apply deep neural network models for the ticket routing problem, incorporating both text and routing network information.

In the remaining of the chapter, we discuss related works in ticket routing and multi-view information matching in Section 6.2. Section 6.3 details the views and components. Experimental setup and results are presented in Section 6.4. Lastly, we conclude the chapter in Section 6.5.

6.2 Related Work

6.2.1 Ticket Routing

The pioneering work of Shao *et al.* [5, 10] modeled ticket routing problem using graphical Markov models. In their formulation, each expert group is represented by a node in a graph. Directed edges exist between two groups with previous ticket transfers, with normalized frequencies as weights. For example, the probability of a ticket to be routed from group A to B , *i.e.*, $P(B|A)$ is the estimated by the frequency of $A \rightarrow B$ over all tickets originated from A . Furthermore, the authors applied frequent pattern mining techniques to find common paths in routing sequences from archived tickets. They showed that, comparing to individual nodes (*e.g.*, $P(C|B)$), using extracted paths collectively as a super-node (*e.g.*, $P(C|\{A, B\})$) improved routing performance, evaluated by MSTR score. Miao *et al.* [6] proposed generative models to predict the conditional probabilities of each possible transfer is taking place given the terms in a ticket. Both models are only applicable after an initial group cannot resolve the ticket. In comparison, our proposed approach is able to handle both initial group assignment and inter-group transfers.

To make a complete routing solution, Sun *et al.* [80] proposed a content-aware model with multiple steps. The authors implemented a filtering step to reduce the number of candidate groups in routing, based on ticket content similarities. Similarly, Xu and He [134] proposed another Two-stage Expert Routing (TER) model, different from [80] only in how the initial group is identified. As a preprocessing step, expert group representation vectors are trained using their solved tickets with randomly sampled negative tickets. The representation method was proposed by Han *et al.* [30], assuming a group only handles tickets with matching skills, and low dimensional group vectors are learned from resolved tickets. In [134], the first step is to determine the initial group, by taking the nearest neighboring groups in a distributed representation space. Thereafter, the probabilistic transition models are used for ticket transfer. Both works used disjointed and decoupled features for initial group assignment and ticket transfer, which could lead to information loss.

In the previous chapter, we present a unified framework for ticket routing (UFTR), built on the feature-based learning-to-rank paradigm. When matching an expert

group to a ticket, the framework incorporates hand-crafted features from four types of information. The ticket features are generated from the ticket text. The group features are characteristics of the group based on their resolved tickets; the ticket-group features are relevance and interactive features computed based on both ticket and group. Lastly, the group-group features are transition probabilities between the current routing sequence and candidate groups. These features can be computed offline and easily extensible. Compared to previous methods, UFTR applies the same model and steps for first both ticket routing scenarios. In this research, we aim to utilize most of the information available to our problem, and leverage the neural network models for automatic feature extraction.

There are also works [31, 82, 85] dedicated to analysis of the contributing factors in the routing phenomenon. Particularly, in [82], the authors concluded that the main factors for routing decision-making are individual group’s expertise and awareness of others’ profiles. While the former is relatively straightforward to obtain, the latter is ambiguous in our problem setting. In another work, Ma *et al.* [153] showed the existence of a theoretical shortest path when applying decentralized search to ticket routing, which is useful in profiling the overall network performance. However, the proposed approach requires the problem areas and difficulty level of a ticket to be known, making it inapplicable to us.

6.2.2 Multi-view deep neural network model

Deep neural network models have been increasingly sophisticated and popular for many applications in Natural Language Processing (NLP). Without any expert network structure, ticket routing can be considered as a text matching problems in information retrieval. Deep Structured Semantic Model (DSSM) [148] is a text matching network structure, generating hidden representations for two inputs in identical pipelines. Then, a relevance score (*e.g.*, cosine similarity) between the two representations is computed. DSSM and its variations are widely adopted models for many text matching applications, such as information retrieval [148–150], recommendation [145, 151, 154, 155], and chatbot [156, 157]. Also working with tickets, Zhou *et al.* [23] proposed a system that matches a ticket to its resolution, using a siamese network architecture, similar to DSSM. Most of these works are

FIGURE 6.1: Multi-view DeepRouting computes a joint relevance score by combining relevance scores from different views.

homogeneous architecture, implementing the same network structures for both inputs. Due to the differences between ticket text and expert network structures, we implement a heterogeneous variation of DSSM.

Mitra *et al.* [152] introduced a duet architecture, using two different deep neural networks to jointly learn both local and distributed representation models from a $\langle \text{text query, document} \rangle$ pairs for document ranking. The networks for local and distributed models are different. The joint relevant score is the sum of scores from each model. On the other hand, Elkahky *et al.* [145] extended DSSM to a multi-view architecture, for recommending items in online marketplaces. The authors modeled item information from different domains using different networks and train the models jointly. In the multi-view architecture, one of the views (user-view in [145]) is shared, while auxiliary items views are modeled individually. In this work, we follow the multi-view architecture in [145] to obtain a joint relevance score for a ticket and a candidate group, score of a resolver should be higher than other groups.

6.3 System Architecture

DeepRouting architecture is designed for multi-view matching given a ticket and an expert group. As illustrated in Figure 6.1, it contains two modules: text matching and graph matching. The text-view takes raw text as input and generates a dense representation vector. Text matching module computes cosine similarity between the vectors for an input ticket and group, respectively. An expert group is

FIGURE 6.2: Overview of text-view processing steps with name labels on the left. Labels on the right indicate the shape of outputs from each step.

represented by a collection of tickets it has resolved. Meanwhile, graph matching part computes the cosine similarity between the ticket text-view and a graph-view vector for a group. The graph-view takes a pretrained graph embedding vector as input and generates a vector with the same dimension as the text-view. Thereafter, the joint relevance score is the weighted sum of scores from both modules. Lastly, the relevance score is multiplied by a group's resolver probability, estimated by the frequency it was the resolver in training set.

6.3.1 Text View

To model the text relevance between a ticket and an expert group, we represent the group by a collection of tickets it resolved previously. We follow the deep neural network architecture proposed by [148, 149] for text semantic modeling. The steps and output data shapes from each step are illustrated in Figure 6.2. Note that, text in tickets is noisy, sparse, and highly domain specific. For this reason, pretrained text embedding methods, such as Word2Vec [116] or GloVe [117] would suffer from high out-of-vocabulary (OOV) issue and poor performance. In this regard, we employ *word hashing* [148], a character-based tokenization reduces the vocabulary size while preserving the trigram information. We first apply a word-based trigram tokenizer, followed by a character-based trigram tokenizer for each

word. The advantage of using such methods is to better handle technical jargon and misspellings words. Word hashing reduces the number bag-of-words features by an order of magnitude. The vectors of 3 words are concatenated to capture multiword expressions.

Next, features from concatenated vectors are extracted by,

$$\mathbf{u}_i = \text{ReLU}(\mathbf{W}_c^\top \mathbf{x}_i), \text{ for } i = 1, \dots, n \quad (6.1)$$

where $\mathbf{W}_c$ is a linear projection matrix for feature space transformation, shared across all input. ReLU is an activation function, *i.e.*, $\text{ReLU}(x) = \max(0, x)$.

To reduce variable number of feature vectors to a fixed size ticket-level feature vector, we perform dimension-wise max-pooling. Specifically,

$$v_j = \max_{i=1, \dots, n} \{u_j^i\}, u_j \in \mathbf{u} \quad (6.2)$$

Lastly, we use a fully-connected layer to extract the high-level semantic embedding,

$$y = \text{ReLU}(\mathbf{W}_f^\top \mathbf{v}) \quad (6.3)$$

Text-view processes are applied to a ticket and a group text input, and the relevance between them is computed as the cosine similarity,

$$R_{text}(\tau, g) = \text{cosine}(y_\tau, y_g) = \frac{y_\tau^\top y_g}{\|y_\tau\| \|y_g\|} \quad (6.4)$$

6.3.2 Graph View

Naturally, an expert network has graph structures and characteristics. It contains expert groups as nodes and the edges are estimated by inter-group ticket transfers from archived tickets, as discussed previously in Section 5.4.2. Previous works in ticket routing explored such information, using graphical models [5, 6, 10, 31, 134] or extract graph specific features [133] for routing models. However, many of them either considered the graph to be loosely coupled with ticket text, or ignored the text entirely. Moreover, comprehensive graph analytics are computationally costly.

Graph embedding is an effective and efficient method to convert a graph into a low dimensional space, while preserving its information [158–160]. Different types of graph embedding include node embedding, edge embedding, and subgraph embedding. In our problem setting, we reckon that the first type is applicable. Specifically, node embedding aims to represent graph nodes in low and fixed dimensional embedding space, using graph structure information and edge weights. In addition, to incorporate node features as input, we choose the Graph Convolutional Network (GCN) [161] over other embedding methods. Specifically, it considers a graph as an adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, where n is the total number of nodes. Meanwhile, $\mathbf{D}$ is its degree matrix, such that $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$. The node features are concatenated in $\mathbf{X} \in \mathbb{R}^{n \times m}$, where m is the number of features. To learn a representation vector for each node, GCN propagates through multiple convolutional layers to incorporate neighborhood information in multiple hops. The values of a layer $\mathbf{H}^{(l+1)} \in \mathbb{R}^{n \times D}$ is derived as:

$$\mathbf{H}^{(l+1)} = \text{ReLU} \left(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^{(l)} \mathbf{W}^{(l)} \right), \quad (6.5)$$

where, $\tilde{\mathbf{A}}$ is the sum of $\mathbf{A}$ and identity matrix I_N , $\mathbf{W}^{(l)}$ is a weight matrix for layer l and $\tilde{\mathbf{D}}$ is the diagonal node degree matrix of $\mathbf{A}$. *ReLU* is the activation function introducing non-linearity to the layer output. The parameter values are trained using gradient descent.

In our setting, we consider the expert graph as a homogeneous graph. We use a technical lexicon with 40k terms and phrase as node features. They are automatically generated by semi-supervised learning similar to [99]. After training, each node is represented in a 300-dimensional vector. To compare with the semantic ticket text-view vector, we use a fully-connected layers to project the graph embedding to a vector z_g and compute the cosine similarity between z_g and y_τ .

6.3.3 Multi-view DeepRouting

For a pair of ticket and group input, the text matching score and the graph matching score are combined using,

$$R(\tau, g) = \alpha R_{\text{text}}(\tau, g) + (1 - \alpha) R_{\text{graph}}(\tau, g) \quad (6.6)$$

where α is a ratio, indicating the significance of text matching. We find the value α in validation set. During training, the objective is to maximize the likelihood of resolver being identified among non-resolver groups, for across the training tickets. This is equivalent to minimizing the loss function,

$$\mathcal{L} = -\log \prod_{(\tau, g^+)} P(g^+|\tau) \quad (6.7)$$

where the probability of the resolver given a ticket is,

$$p(g^+|\tau) = \frac{\exp(\beta_{g^+} R(\tau, g^+))}{\sum_{g \in \mathcal{G}} \exp(\beta_g R(\tau, g))} \quad (6.8)$$

where β is a prior probability for a group to be the resolver, estimated from training tickets.

The neural network parameters are trained using Stochastic Gradient Descent with ADADELTA [162] optimizer to dynamically adjust learning rate during training. Each mini-batch consists of 64 positive ticket-group pairs. For each positive pair, 49 negative groups are selected, including the non-resolvers in their routing paths and the remainings are randomly sampled. Initially, parameter weights are randomly set. Training takes around 50 hours on a server with Nvidia V100 GPUs.

6.4 Experiment

To evaluate DeepRouting's performance, we conduct two sets of experiments on a proprietary data set. Firstly, we evaluate the effectiveness of our model in ranking the true resolver among other groups. Secondly, we simulate the routing process in Assistive Routing, and compare the performance of our model with human and algorithmic baselines. Baselines are selected individually for different evaluations. The data used for experiment is identical to the data used in the previous chapter. The overall data distributions are shown in Figure 5.2.

From the data set, we select tickets with length between 20 to 150 terms, assuming tickets that are too short or too long are outliers. Among the remaining tickets, we filter again based on the number of tickets their resolver have resolved. Specifically, we exclude groups that have solved less than 50 or more than 5,000 tickets. From

the selected tickets, we randomly select 55,000 tickets for training and 5,000 for validation. For each positive pair, 49 negative groups are selected, by including the non-resolvers in its routing path and random sampling for the remaining. For each expert group, 100 tickets are randomly sampled from all tickets it has resolved. From each of routing steps 1 to 4, we sample 500 from the selected tickets for testing. In summary, we have 55,000 tickets for training, 5,000 tickets for validation, and 2000 tickets for test. We preprocess and manually examined the selected tickets to ensure that any personal information, or any information that could be used to infer user’s identity, are removed.

6.4.1 Resolver Ranking

During operation, ticket routing is equivalent to resolver ranking at each routing step. In the first experiment, we compare the multi-view DeepRouting model with five baseline models for a single step resolver ranking.

BM25 [163] is a classic model for scoring documents with respect to their relevance given a query in information retrieval. We use term vectors to represent each ticket as a query and each expert group a document, which is represented as a collection of tickets it has resolved in training set.

FastText-Embedding [140] represents a family of word embedding methods that are the state-of-the-art for many natural language applications. They use neural network to learn a dense vector representation for each word in the training corpus. FastText uses sub-word information when learning word representations to handle out-of-vocabulary words. We first train 300 dimensional word embeddings using all tickets in the training set. Subsequently, a ticket vector is the inverse document frequency (IDF) weighted average vector of all its unique word vectors. An expert group vector is represented in the similar way. The relevance score between a ticket and a group is the cosine similarity of their vector representations.

FastText-Classification [164] represents text classification supervised models. In our setup, each ticket is a training instance while its resolver is the label. There are many text classification models, we choose FastText because it outperforms

TABLE 6.1: Performance for the resolver ranking. The best model is in bold-face, while the second best case is underlined.

	HR@1	@3	@5	@10
BM25	0.635	0.728	0.773	0.885
FastText _{Embedding}	0.433	0.597	0.669	0.787
FastText _{Classification}	0.656	0.756	0.858	0.906
UFTR _{Pointwise}	0.687	0.785	0.927	0.948
UFTR _{Pairwise}	<u>0.719</u>	<u>0.825</u>	<u>0.942</u>	<u>0.975</u>
DeepRouting _{Text}	0.665	0.784	0.895	0.918
DeepRouting _{Graph}	0.603	0.742	0.835	0.881
DeepRouting _{MultiView}	0.721	0.844	0.958	0.981

many other models during our experiments. During testing, a list of prediction are generated, ranked by probability scores.

UFTR-Pointwise, UFTR-Pairwise [133] belong to a ranking framework using multiple types of hand-crafted features. The feature types are Ticket, Group, Ticket-Group, and Group-Group. Note that the original Group-Group features include the transfer probabilities between a current group to a candidate group. Since we only focus on one step resolver ranking in this experiment, only the probability of a candidate being the resolver is used. For pointwise version, a Random Forest Regressor (RFR) model is trained using training set with randomly selected negative samples with 1:1 ratio. During testing, a score is generated for a ticket and each candidate resolver for ranking. The pairwise model is a classification model, determining if a group is likely to rank above the other group. Since we are only interested in resolver’s ranking, during testing, we compare the ground truth resolver with each of the other groups to determine its ranking.

DeepRouting-Text, DeepRouting-Graph, DeepRouting-DualView are different settings of DeepRouting. Text and Graph are single-view versions, only using text and graph representation of expert groups, respectively. Due to the change in network architecture, all models are trained independently on training data, for fair comparison.

Performance for resolver ranking is evaluated by Hit Rate (HR) in a *leave-one-out* test. Particularly, for each test ticket with a ground truth resolver, 49 other groups are randomly sampled to form the candidate resolver list. Each model outputs a

ranking of the resolver in the 50 groups. HR is computed at position 1, 3, 5, and 10.

Shown in Table 1, multi-view DeepRouting outperforms the other models across the board. Particularly, it achieves 0.981 in HR@10. Closely behind is UFTR-Pairwise model. Both top performers incorporate multiple views of information when representing the expert groups. The difference lies in how features are generated. In UFTR, features are manually crafted, while they are automatically generated by neural networks in DeepRouting. Also, both models consider the relative differences of the ground truth and negative groups during training. UFTR-Pairwise considers the relative position, whereby DeepRouting considers the posterior probability of ground truth group among multiple negative samples in the loss function. Noticeably, BM25 is a strong baseline, performing better than two other models. FastText-Embedding and DeepRouting-Graph rely on abstract representation of expert groups from the input, but they are the least performing models. Input for both models could be too abstract to contain valuable information about the expert group. In comparison, FastText-Classification and DeepRouting-Text are similar in terms of using text as input. Both could produce a hidden representation for tickets during the training process and comparable in performance.

6.4.2 Assistive Routing

Many previous works [6, 10, 80, 134] evaluated their approaches based solely on computed routing results. For example, a system generates the most probable resolver at every step, until the true resolver was found. With the routing path, a MSTR score was computed based on the length of the sequence. In Chapter 3, we argued for more information from human routing decision to be utilized in evaluation. We proposed an Assistive Routing strategy for ticket routing evaluation, to better simulates ticket routing systems in operation, while measuring the performance against the ground truth path, instead of only resolver. Specifically, the system under test generates a list candidate resolvers at each step. A reviewer in the current group is supposed to determine if the resolver is in the list. Otherwise, they should make their own routing decision.

In this experiment, we assume the top- k candidate groups for a ticket are to be reviewed by a pseudo-reviewer. If the true resolver is in the list, the reviewer could

correctly route the ticket to it and the ticket is resolved. Otherwise, the ticket is routed to the next group along the ground truth routing path. The process repeats until the resolver is reached. Each routing step is scored by the distance from its proposed group to the resolver. The best score a test ticket could achieve is 1, when the resolver is proposed in the first step; the score decreases exponentially for every extra step it requires. The overall performance for a test set is the average scores of each test cases, *i.e.*, Mean Average Distance to Resolver (MADR). Formally,

$$MADR(\mathcal{T}) = \frac{\sum_{t \in \mathcal{T}} \phi(t)}{|\mathcal{T}|}, \text{ where } \phi(t) = \frac{\sum_{g \in \hat{S}_t} \varphi(g)}{|\hat{S}_t|} \quad (6.9)$$

where $\varphi(g) = \frac{1}{2^{\Delta(g, g_{resolver})}}$ is the scoring function for a group g , and Δ represents “distance between”.

We compare MADR scores of DeepRouting with multiple baselines, including the **Human Routing** score, which is computed from the ground truth routing paths in test set. As discussed earlier, many previous works are only partially relevant in our problem setting. To compare with complete routing frameworks, we choose Two-stage Expert Recommendation (TER) [134] and Unified Framework for Ticket Routing (UFTR) with different settings as baselines. Besides TER-FM, TER-FMS, and TER-VMS presented in Section 5.6, we add a new baseline here, inspired by Miao [6].

TER-GGT is Generative Greedy Transfer proposed by Miao *et al.* [6]. The routing probability of an expert group g_i to another group g_{i+1} is,

$$P(g_{i+1} | \tau, g_i) = \frac{P(\tau | e_{i,i}) P(g_{i+1} | g_i)}{Z(t, g_i)} \quad (6.10)$$

where $e_{i,i+1}$ characterizes the directional edge goes from g_i to g_{i+1} . $P(\tau | e_{i,i+1}) = \prod_{w_k \in \tau} P(w_k | e_{i,i+1})^{\#(w_k, \tau)}$ is the product of probabilities of individual word tokens $w_k \in \tau$ being randomly drawn from all tickets routed along $e_{i,i+1}$. $f(w_k, \tau)$ is an indicator function, equals to 1 if w_k presents in τ ; 0 otherwise. $P(g_{i+1} | g_i)$ is the prior probability of group g_i sending a ticket to g_{i+1} , estimated from archived routings. $Z(\tau, g_i) = \sum_{g_{i+1} \in \mathcal{G}} P(\tau | e_{i,i+1}) P(g_{i+1} | g_i)$ is the sum of probabilities to all routing options. Essentially, the formulation in Equation 6.10 is a softmax function where the edge weights from g_i to all possible nodes are normalized.

TABLE 6.2: Performance comparison in Assistive Routing. The best model is in bold-face, while the second best case is underlined. The models use different information for routing, indicated by $\dagger$ for text and $\sharp$ for graph.

	MADR@1	@3	@5	@10
Human Routing	0.701			
TER-FM $\sharp$	0.701	0.742	0.754	0.771
TER-FMS $\sharp$	0.706	0.750	0.781	0.828
TER-VMS $\sharp$	0.709	0.780	0.804	0.874
TER-GGT $\dagger, \sharp$	0.711	0.805	0.883	0.924
UFTR $\dagger, \sharp$ Pointwise	0.755	0.863	0.913	0.985
UFTR $\dagger, \sharp$ Pairwise	<u>0.757</u>	<u>0.893</u>	<u>0.925</u>	<u>0.990</u>
DeepRouting $\dagger$ Text	0.719	0.818	0.905	0.958
DeepRouting $\sharp$ Graph	0.703	0.774	0.845	0.901
DeepRouting $\dagger, \sharp$ MultiView	0.760	0.913	0.945	0.994

UFTR-Pointwise and **UFTR-Pairwise** are described in Section 6.4.1. Similar to the settings in Resolver Ranking, we compare three different settings of DeepRouting with the baselines. For candidate resolver generation, we use the same information retrieval based with graph expansion method used in in [133], for all models. Note that TER-based models hold a fixed candidate set after the first stage, where as UFTR and DeepRouting models update candidates at every routing step.

The MADR scores at $k \in 1, 3, 5, 10$ for assistive routing are shown in Table 6.2. Overall, DeepRouting-DualView model outperforms other baselines, improving from human routing baseline by 8% and 40% in MADR@1 and @10, respectively. It also scores significantly higher than any of the TER models. Among the TER-based models, GGT uses text features and performs better than models using only graph information. The effects of text and graph features are also observable from single view DeepRouting models. Particularly, DeepRouting-Graph is behind DeepRouting-Text and barely comparable to the inferior TER models. These results show that, even graph embedding model considers edge information, the effectiveness in routing is limited. It could be due to the decoupling of graph embedding model and routing model trainings. In comparison, the count-based statistical features, though simple, are more expressive and do not require training. Nevertheless, text information has greater weights in solving the ticket routing problem in general. On the other hand, UFTR-based models are not far from the best DeepRouting model. Even the differences between pointwise and pairwise models are marginal. After all, having both text and graph features at the same

FIGURE 6.3: Number of human routing steps vs. number of Assistive Routing steps in (a) TER-GGT, (b) UFTR-Pairwise, and (c) DeepRouting-DualView. In each subplot, bars with the same colors are represent test tickets that have 1 to 4 steps by human routing. The height of a bar indicates the ratio of tickets that resolved by 1 to 4 steps in Assistive Routing, within its respective color. Note that, we are only interested in assistive routing performance, we exclude cases whereby the resolver is identified after a model failed to recommend after n steps. Therefore, that height of bars with the same color do not necessarily sum up to 1. (Best view in color).

time are advantageous over using either one of them, regardless of the modeling method.

Further, we evaluate the effectiveness of the proposed models in solving tickets with various difficulties, represented by the number of human routing steps. Particularly, we want to investigate how the models perform on tickets took different 1 to 4 human routing. Here, we only consider the top-1 recommendation at each step. They are effectively the fine-grained results from which the MADR@1 results are computed. Among three best model settings, our proposed model (Figure 6.3) is superior in multiple aspects. Firstly, bars in the farthest row for each color are the tallest in Figure 6.3(c). This shows that DeepRouting-DualView managed to solve the highest ratio of tickets within the first Assistive Routing step. In comparison, UFTR-Pairwise has the first step resolved ratios slightly lower, while TER-GGT resolved the least. Secondly, for rows other than the farthest, DeepRouting-DualView has the lowest bars compared to the other two models, showing DeepRouting-DualView is more likely to propose the resolver in the first Assistive Routing step. Lastly, for tickets that required 3 to 4 human routing steps (green and purple bars), DeepRouting-DualView resolve the most ratio of tickets in aggregate. As the length of human routing sequence is an implicit indication of a ticket’s difficulty, it shows that DeepRouting-DualView is able to handle difficult tickets more effectively.

6.5 Conclusion

In this chapter, we propose a deep neural network approach for ticket routing in an expert network. We apply classic text matching and graph embedding in a multi-view architecture to match a ticket with expert groups. The main improvement from the other unified routing framework is that, features are automatic extracted by neural network models. Empirical results show superior performance of the proposed approach in both resolver ranking and Assistive Routing. The results also show that text similarity is more effective compared to routing graph modeling in ticket routing. A valuable future research direction is to investigate if simplifying the network structure would improve the overall routing performance. As for the technical choices, we propose to experiment with more text matching frameworks, leveraging both neural networks and hand-crafted features. We would also like

to investigate an end-to-end architecture integrating the graph node embedding during the training process. Lastly, the generality and robustness of the framework need future examination with data from different systems.

Chapter 7

Conclusions and Future Work

In this thesis, we investigate how IT service providers can use text contents and historical routing paths to improve their ticket routing efficiency. We first introduce a novel metric for ticket routing and Assistive Routing evaluation framework. The new metric considers partial matching of a routing path with the ground truth, awarding a non-resolver group according to its distance to the resolver in a path. We justified why we believe it is a reasonable metric to use to supplement existing metrics. Next, we looked into how to extract software product names and version information from user-generated ticket for more informative ticket representation for routing. After analyzing manually annotated real-life tickets, we proposed an extraction and linking models, using manually annotated tickets and domain-specific features. In experiments, we showed linear models with the proposed features can deliver better and more consistent results, compared with state-of-the-art neural network baseline models, especially for the dataset with sparse labels. Subsequently, we formulate the ticket routing problem as a learning-to-rank problem and propose two frameworks as solutions. Compared to previous multi-stage approaches, both of our proposed solutions can address initial expert group assignment and ticket transfer uniformly. The first framework uses hand-crafted features from four types of information, namely, ticket, expert group, ticket-group, and group-group. The experiments show that using the pairwise model outperforms the baselines as well as using pointwise ranking in Assistive Routing. Through the ablation study, we find that the most important feature group is ticket-group. Such finding was significant for use in proposing the second solution, a multi-view deep neural network approach with the focus in modeling ticket-group similarities.

Specifically, we employed a classic deep semantic model to represent text in tickets and groups when modeling their similarities. The groups are also represented by pre-trained graph embeddings. We showed that the deep neural network approach outperforms baselines and our first framework. We also found that using graph embedding to represent expert groups are less effective. This is consistent with Bartolini *et al.* [88] finding, such that the performance gain from using routing paths information is limited. In an expert network, as overall routing performance improves, inter-group ticket transfer reduces, resulting in a sparse graph with a few edges. Such findings could be helpful for organizations to revise their expert network setup, in terms to improve overall performance, such as merging groups that transfer tickets frequently.

The evaluation framework we proposed is effectively a “human-in-the-loop” simulation. Most of the similar applications adopt automatic evaluation approaches, only comparing the system output with the ground truth label. This training-predict paradigm is over-simplified for real-life applications. For example, it is not reasonable to follow a routing system to keep predicting incorrectly without intervention. In our settings, we want to capture human decision-making patterns in terms of ticket routing. Not only for the final resolver but also at every routing step, even for incorrect ones. In our evaluation, when the top- k routing recommendations do not include the resolver, we take the routing decision made by humans and continue. In this way, we simulate the entire ticket routing process leveraging most of the information available. With such an evaluation framework, our routing solutions are more relevant and feasible to the real application implementation. On the other hand, our proposed routing solutions are flexible and scalable. The unified routing model can incorporate results from previous or future ticket analysis directly as features. While the deep neural network framework is compatible with more sophisticated network architectures and scalable to more training data. However, like any model-based solutions, to implementation in another support organization, modification and adaptation are unavoidable.

Lastly, we discuss potential future directions for this research into two scopes, ticket routing and intelligent support in general.

Ticket routing. We have briefly introduced some ideas to improve the proposed frameworks. Specifically, we see great potential in the application of deep neural networks. More sophisticated models are emerging from research communities

more frequently than ever. As we have explored in this research, NLP models specialized in text modeling and similarity comparison are especially relevant to our task. It is also interesting to explore neural network-based IR techniques on relevant document retrieval and ranking. In the meanwhile, we propose to carry out more evaluation with additional metrics and detailed error analysis. They will benefit our understandings of different techniques and help us in model selection.

Besides the rapidly evolving techniques, ticket routing can be further benefited by integrating more diverse data sources. In this research, we are only permitted with limited access to historical tickets. The effectiveness of our solutions could be improved with more data enrichment. For example, the mappings between customers to products are very useful information for ticket routing. Using such enrichment to our solutions in candidate selection may effectively reduce the candidate space, thus improving the routing quality. However, the main challenge is not technical in nature. The management should adopt a unified data policy and governance.

Intelligent support. The ticket routing problem emerged from the standard support pipeline, which depends on a large number of experts who are only experts in their areas. The problem could be eased by changing the support workflow. The intelligent support, in the essence, is a search engine with the technical knowledge to assist the user with their support needs. This is already implemented in popular open search engines, such as Google. For domain-specific applications, some support organizations have implemented an additional search step on behalf of the user, when a ticket is created. The output from the search may be the answer to the question, a document for reference, or a similar ticket asked by other users. This expects the support organizations to have guidelines on solution documentation and knowledge management, also clean up the knowledge base regularly. Research in Question Answering (QA) is dedicated to the same use cases. An interesting direction in QA is to understand what the customer is asking for and construct the answer from multiple sources. When the customer can find the answer themselves, ticket routing is no longer a problem itself. In the foreseeable future, computers with AI may be able to control, diagnosis and fix themselves and each other. By that time, support jobs for humans are likely to stop existing, along with ticket and the problem of ticket routing.

List of Author's Publications

Journal Articles

- **Jianglei Han**, Jing Li, and Aixin Sun, “UFTR: A Unified Framework for Ticket Routing,” Empirical Software Engineering, (In communication).

Conference Proceedings

- **Jianglei Han**, and Aixin Sun, “Mean Average Distance to Resolver: An Evaluation Metric for Ticket Routing in Expert Network,” in IEEE International Conference in Software Maintenance and Evolution, 2017.
- **Jianglei Han**, Ka Hian Goh, Aixin Sun, and Mohammad Akbari, “Towards Effective Extraction and Linking of Software Mentions from User-Generated Support Tickets,” in Conference on Information and Knowledge Management, 2018.
- **Jianglei Han**, and Aixin Sun, “DeepRouting: A Deep Neural Network Approach for Ticket Routing in Expert Network,” IEEE Conference in Service Computing, 2020.

Bibliography

- [1] Rahul Potharaju, Navendu Jain, and Cristina Nita-Rotaru. Juggling the jigsaw: Towards automated problem inference from network trouble tickets. In *USENIX Symposium on Networked Systems Design and Implementation*, pages 127–141, 2013. [xvii](#), [xix](#), [12](#), [14](#), [15](#), [18](#), [19](#), [61](#)
- [2] Qing Wang, Wubai Zhou, Chunqiu Zeng, Tao Li, Larisa Shwartz, and Genady Ya Grabarnik. Constructing the knowledge base for cognitive it service management. In *IEEE International Conference on Services Computing*, pages 410–417, 2017. [xvii](#), [12](#), [14](#), [15](#), [18](#), [19](#), [59](#)
- [3] AXELOS. *ITIL® Foundation, ITIL 4 edition*. TSO (The Stationery Office), 2019. [xix](#), [1](#), [2](#)
- [4] John S Justeson and Slava M Katz. Technical terminology: some linguistic properties and an algorithm for identification in text. *Natural language engineering*, 1(1):9–27, 1995. [xix](#), [14](#)
- [5] Qihong Shao, Yi Chen, Shu Tao, Xifeng Yan, and Nikos Anerousis. Efficient ticket routing by resolution sequence mining. In *KDD*, pages 605–613, 2008. [xix](#), [30](#), [31](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [39](#), [43](#), [46](#), [82](#), [83](#), [85](#), [95](#), [98](#), [107](#), [111](#)
- [6] Gengxin Miao, Louise E Moser, Xifeng Yan, Shu Tao, Yi Chen, and Nikos Anerousis. Generative models for ticket resolution in expert networks. In *KDD*, pages 733–742, 2010. [xix](#), [6](#), [29](#), [30](#), [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [44](#), [46](#), [47](#), [81](#), [82](#), [83](#), [85](#), [95](#), [98](#), [107](#), [111](#), [116](#), [117](#)
- [7] Shivali Agarwal, Renuka Sindhgatta, and Bikram Sengupta. SmartDispatch: enabling efficient ticket dispatch in an IT service environment. In *KDD*, pages 1393–1401, 2012. [6](#), [12](#), [25](#), [27](#), [29](#), [30](#), [45](#)
- [8] Chunqiu Zeng, Wubai Zhou, Tao Li, Larisa Shwartz, and Genady Ya Grabarnik. Knowledge guided hierarchical multi-label classification over ticket data. *IEEE Transactions on Network and Service Management*, 14(2):246–260, June 2017. [12](#), [13](#), [14](#), [21](#), [26](#), [27](#)
- [9] Jianglei Han and Mohammad Akbari. Vertical domain text classification: Towards understanding IT tickets using deep neural networks. In *AAAI Conference on Artificial Intelligence*, 2018. [6](#), [59](#)

- [10] Qihong Shao, Yi Chen, Shu Tao, Xifeng Yan, and Nikos Anerousis. Easyticket: a ticket routing recommendation engine for enterprise problem resolution. *VLDB*, 1(2):1436–1439, 2008. [6](#), [34](#), [43](#), [81](#), [82](#), [90](#), [94](#), [107](#), [111](#), [116](#)
- [11] Giuseppe A Di Lucca, Massimiliano Di Penta, and Sara Gradara. An approach to classify software maintenance requests. In *ICSM*, 2002. [12](#), [29](#), [45](#)
- [12] Rajeev Gupta, K Hima Prasad, and Mukesh Mohania. Automating itsm incident management process. In *ICAC*, pages 141–150, 2008. [29](#), [30](#), [45](#)
- [13] E E Jan, J Ni, N Ge, and N Ayachitula. A statistical machine learning approach for ticket mining in it service delivery. In *IM*, 2013. [12](#), [15](#)
- [14] X Wei, A Sailer, and R Mahindru. Automatic structuring of IT problem ticket data for enhanced problem resolution. In *IM*, 2007. [12](#), [13](#), [17](#)
- [15] Shivali Agarwal, Vishalaksh Aggarwal, Arjun R Akula, Gargi Banerjee Dasgupta, and Giriprasad Sridhara. Automatic problem extraction and analysis from unstructured text in IT tickets. *IBM Journal of Research and Development*, 61(1):4–41, 2017. [12](#), [13](#), [14](#), [15](#), [19](#), [23](#), [59](#), [61](#)
- [16] Rongda Zhu, Yu Deng, Soumitra Sarkar, Kaoutar El Maghraoui, Harigovind V Ramasamy, and Alan Bivens. Towards More Effective Solution Retrieval in IT Support Services Using Systems Log. In *ICSOC*, 2016. [15](#)
- [17] Andrii Maksai, Jasmina Bogojenska, and Dorothea Wiesmann. Hierarchical incident ticket classification with minimal supervision. In *ICDM*, pages 923–928. IEEE Computer Society, December 2014. [12](#), [25](#)
- [18] Ioana Giurgiu, Jasmina Bogojenska, Sergii Nikolaiev, George Stark, and Dorothea Wiesmann. Analysis of labor efforts and their impact factors to solve server incidents in datacenters. In *CCGrid*, pages 424–433. IEEE, September 2016. [12](#), [17](#)
- [19] Karthikeyan Ponnalagu. Ontology-driven root-cause analytics for user-reported symptoms in managed IT systems. *IBM Journal of Research and Development*, 61(1):5–53, 2017. [12](#), [14](#), [18](#), [59](#)
- [20] Monika Gupta, Allahbaksh Asadullah, Srinivas Padmanabhuni, and Alexander Serebrenik. Reducing user input requests to improve IT support ticket resolution process. *Empirical Software Engineering*, 23(3):1664–1703, 2018. [12](#)
- [21] T L Acorn and S H Walden. Smart: Support management automated reasoning technology for compaq customer service. In *IAAI*, 1992. [13](#), [14](#)

- [22] S Roy, D P Muni, JJYT Yan, and N Budhiraja. Clustering and labeling it maintenance tickets. *ICSOC*, 2016. [13](#), [14](#), [22](#)
- [23] Wubai Zhou, Wei Xue, Ramesh Baral, Qing Wang, Chunqiu Zeng, Tao Li, Jian Xu, Zheng Liu, Larisa Shwartz, and Genady Ya Grabarnik. Star: A system for ticket analysis and resolution. In *KDD*, pages 2181–2190, 2017. [13](#), [19](#), [20](#), [108](#)
- [24] Mirela Madalina Botezatu, Jasmina Bogojenska, Ioana Giurgiu, Hagen Völzer, and Dorothea Wiesmann. Multi-view incident ticket clustering for optimal ticket dispatching. In *KDD*, pages 1711–1720, 2015. [13](#), [14](#), [22](#), [23](#), [24](#)
- [25] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008. [13](#)
- [26] A Medem and M I Akodjenou. Troubleminer: Mining network trouble tickets. In *IM*, 2009. [13](#), [22](#)
- [27] Ea-Ee Jan, Kuan-Yu Chen, and Tsuyoshi Idé. A probabilistic concept annotation for it service desk tickets. In *ESAIR*, pages 21–23, New York, New York, USA, 2014. ACM Press. [14](#), [15](#), [23](#)
- [28] S Agarwal, G Sridhara, and G Dasgupta. Automated Quality Assessment of Unstructured Resolution Text in IT Service Systems. In *ICSOC*, 2016. [14](#), [20](#)
- [29] Ann Taylor, Mitchell Marcus, and Beatrice Santorini. The penn treebank: an overview. In *Treebanks*, pages 5–22. Springer, 2003. [14](#)
- [30] Fangqiu Han, Shulong Tan, Huan Sun, Mudhakar Srivatsa, Deng Cai, and Xifeng Yan. Distributed representations of expertise. In *SIAM Data Mining*, pages 531–539, 2016. [14](#), [33](#), [83](#), [98](#), [107](#)
- [31] Huan Sun, Mudhakar Srivatsa, Shulong Tan, Yang Li, Lance M Kaplan, Shu Tao, and Xifeng Yan. Analyzing expert behaviors in collaborative networks. In *KDD*, pages 1486–1495, 2014. [14](#), [33](#), [34](#), [44](#), [83](#), [108](#), [111](#)
- [32] Chunqiu Zeng, Tao Li, Larisa Shwartz, and Genady Grabarnik. Hierarchical multi-label classification over ticket data using contextual loss. In *NOMS*, 2014. [14](#), [21](#), [26](#), [27](#)
- [33] Kayhan Moharrerri, Jayashree Ramanathan, and Rajiv Ramnath. Recommendations for achieving service levels within large-scale resolution service networks. In *ACM India Conference*, pages 37–46, 2015. [14](#), [21](#), [24](#), [27](#), [31](#), [34](#)
- [34] Kayhan Moharrerri, Jayashree Ramanathan, and Rajiv Ramnath. Motivating dynamic features for resolution time estimation within IT operations management. In *IEEE Big Data*, pages 2103–2108, 2016. [32](#), [45](#), [81](#)

- [35] Kayhan Moharrerri, Jayashree Ramanathan, and Rajiv Ramnath. Probabilistic sequence modeling for trustworthy IT servicing by collective expert networks. In *IEEE COMPSAC*, volume 1, pages 379–389, 2016. [21](#), [24](#), [27](#), [31](#), [32](#), [45](#)
- [36] Rahul Potharaju, Joseph Chan, Luhui Hu, Cristina Nita-Rotaru, Mingshi Wang, Liyuan Zhang, and Navendu Jain. Confseer: Leveraging customer support knowledge bases for automated misconfiguration detection. *Proceedings of the VLDB Endowment*, 8(12), 2015. [14](#), [19](#)
- [37] Wubai Zhou, Tao Li, Larisa Shwartz, and Genady Ya Grabarnik. Recommending ticket resolution using feature adaptation. In *International Conference on Network and Service Management*, pages 15–21, 2015. [14](#), [59](#)
- [38] Yong-Bin Kang, Arkady Zaslavsky, and Shonali Krishnaswamy. A Knowledge-rich Similarity Measure for Improving IT Incident Resolution Process. In *SAC*, pages 1781–1788, 2010. [15](#)
- [39] L Tang, T Li, and L Shwartz. Identifying missed monitoring alerts based on unstructured incident tickets. In *International Conference on Network and Service Management*, 2013. [15](#), [24](#), [27](#)
- [40] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003. [15](#)
- [41] Wubai Zhou, Liang Tang, Tao Li, Larisa Shwartz, and Genady Ya Grabarnik. Resolution recommendation for event tickets in service management. In *IFIP/IEEE Symposium on Integrated Network Management*, pages 287–295, 2015. [15](#), [59](#)
- [42] Yu Deng, KE Maghraoui, TD Griffin, V Agarwal, SG Tamilselvam, RD Sharnagat, TH Alexander, NE Gómez, CM Cramer, A Bivens, et al. Advanced search system for IT support services. *IBM Journal of Research and Development*, 61(1):3–27, 2017. [15](#), [19](#), [59](#)
- [43] Niklas Goby, Tobias Brandt, Stefan Feuerriegel, and Dirk Neumann. Business Intelligence for Business Processes - the Case of IT Incident Management. In *ECIS*, 2016. [15](#), [29](#)
- [44] Liangjie Hong and Brian D Davison. Empirical study of topic modeling in twitter. In *Proceedings of the first workshop on social media analytics*, pages 80–88. ACM, 2010. [16](#)
- [45] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391, 1990. [16](#)
- [46] Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57. ACM, 1999. [16](#)

- [47] G Miao, Z Guan, L E Moser, X Yan, and S Tao. Latent association analysis of document pairs. In *KDD*, 2012. 16
- [48] V Shimpi, M Natu, V Sadaphal, and V Kulkarni. Problem identification by mining trouble tickets. In *COMAD*, 2014. 17
- [49] R Liu and J Lee. IT incident management by analyzing incident relations. In *International Conference in Service-Oriented Computing*, 2012. 17
- [50] R Liu, Q Li, F Li, L Mei, and J Lee. Big Data architecture for IT incident management. *Service Operations and Logistics*, 2014. 17
- [51] Shuo Yang, Lei Zou, Zhongyuan Wang, Jun Yan, and Ji-Rong Wen. Efficiently answering technical questions — a knowledge graph approach. In *AAAI Conference on Artificial Intelligence*, 2017. 19, 59
- [52] Vishalaksh Aggarwal, Shivali Agarwal, Gaargi B Dasgupta, Giriprasad Sridhara, and Vijay E. ReAct: A System for Recommending Actions for Rapid Resolution of IT Service Incidents. In *IEEE International Conference on Services Computing*, pages 1–8, 2016. 19, 59
- [53] A Watanabe, K Ishibashi, and T Toyono. Workflow extraction for service operation using multiple unstructured trouble tickets. *NOMS*, 2016. 19
- [54] Qing Wang, Tao Li, SS Iyengar, Larisa Shwartz, and Genady Ya Grabarnik. Online IT ticket automation recommendation using hierarchical multi-armed bandit algorithms. In *Proceedings of the 2018 SIAM International Conference on Data Mining*, pages 657–665, 2018. 20
- [55] D Majumdar, R Catherine, and S Ikbali. Privacy protected knowledge management in services with emphasis on quality data. In *CIKM*, 2011. 20
- [56] Maja Vukovic, Jim Laredo, and Valentina Salapura. h-iq - human intelligence for quality of service delivery data. In *SCC*, 2012. 20
- [57] Ta Hsin Li, Rong Liu, Noi Sukaviriya, Ying Li, Jeaha Yang, Michael Sandin, and Juhnyoung Lee. Incident Ticket Analytics for IT Application Management Services. In *IEEE International Conference on Services Computing*, pages 568–574, 2014. 22, 59
- [58] Derek Lin, Rashmi Raghu, Vivek Ramamurthy, Jin Yu, Regunathan Radhakrishnan, and Joseph Fernandez. Unveiling clusters of events for alert and incident management in large-scale enterprise it. In *KDD*, pages 1630–1639, New York, New York, USA, 2014. ACM Press. 22
- [59] Patricia Marcu, Genady Grabarnik, Laura Z Luan, Daniela Rosu, Larisa Shwartz, and Christopher Ward. Towards an optimized model of incident ticket correlation. In *IM*, 2009. 22, 23

- [60] Stanisław Osiński, Jerzy Stefanowski, and Dawid Weiss. Lingo: Search results clustering algorithm based on singular value decomposition. In *Intelligent information processing and web mining*, pages 359–368. Springer, 2004. [23](#)
- [61] P Dhoolia, P Chugh, P Costa, N Gantayat, M Gupta, N Kambhatla, R Kumar, S Mani, P Mitra, C Rogerson, and M Saxena. A cognitive system for business and technical support: A case study. *IBM Journal of Research and Development*, 61(1):7:74–7:85, March 2017. [23](#)
- [62] S Mani, K Sankaranarayanan, and V S Sinha. Panning requirement nuggets in stream of software maintenance tickets. In *FSE*, 2014. [23](#)
- [63] S Kikuchi. Prediction of workloads in incident management based on incident ticket updating history. In *International Conference on Utility and Cloud Computing*, 2015. [24](#), [27](#)
- [64] Yixin Diao, Hani Jamjoom, and David Loewenstern. Rule-based problem classification in IT service management. In *IEEE Cloud Computing*, pages 221–228, 2009. [24](#), [27](#)
- [65] G B Dasgupta, T K Nayak, A R Akula, and S Agarwal. Towards auto-remediation in services delivery: Context-based classification of noisy and unstructured tickets. In *ICSOC*, 2014. [25](#), [27](#)
- [66] I Giurgiu, D Wiesmann, J Bogojeska, D Lanyi, G Stark, R B Wallace, M M Pereira, and A A Hidalgo. On the adoption and impact of predictive analytics for server incident reduction. *IBM Journal of Research and Development*, 61(1):9:98–9:109, March 2017. [25](#), [27](#)
- [67] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987. [25](#)
- [68] Cristina Kadar, Dorothea Wiesmann, Jose Iria, Dirk Husemann, and Mario Lucic. Automatic Classification of Change Requests for Improved IT Service Quality. In *SRII*, pages 430–439. IEEE, November 2010. [25](#), [27](#)
- [69] David D Lewis and William A Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–12. Springer-Verlag New York, Inc., 1994. [25](#)
- [70] F Wang, J Sun, T Li, and N Anerousis. Two heads better than one: Metric+ active learning and its applications for IT service classification. In *ICDM*, pages 1022–1027. IEEE, 2009. [25](#), [27](#)
- [71] Daniel Borchmann, Rafael Peñaloza, and Wenqian Wang. Classifying software bug reports using methods from formal concept analysis. *Studia Universitatis Babeş-Bolyai Informatica*, 59:10–27, June 2014. Supplemental proceedings of the 12th International Conference on Formal Concept Analysis (ICFCA’14). [26](#), [27](#)

- [72] Wenqian Wang. Error-tolerant direct bases. Master's thesis, Technische Universität Dresden, Dresden, Germany, 2013. [26](#), [27](#)
- [73] Bharath Hariharan, Lihi Zelnik-Manor, Manik Varma, and Svn Vishwanathan. Large scale max-margin multi-label classification with priors. In *ICML*, pages 423–430, 2010. [26](#)
- [74] Lijuan Cai and Thomas Hofmann. Hierarchical document categorization with support vector machines. In *CIKM*, pages 78–87, 2004. [26](#)
- [75] Nicolò Cesa-Bianchi, Claudio Gentile, and Luca Zaniboni. Incremental algorithms for hierarchical classification. *Journal of Machine Learning Research*, 7(Jan):31–54, 2006. [26](#)
- [76] Wei Bi and James T Kwok. Hierarchical multilabel classification with minimum bayes risk. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 101–110. IEEE, 2012. [26](#)
- [77] Yang Song, Anca Sailer, and Hidayatullah Shaikh. Hierarchical Online Problem Classification for IT Support Services. *IEEE Transactions on Services Computing*, pages 345–357, 2012. [27](#)
- [78] Jasmina Bogojeska, David Lanyi, Ioana Giurgiu, George Stark, and Dorothea Wiesmann. Classifying server behavior and predicting impact of modernization actions. In *CNSM*, 2013. [27](#)
- [79] Asheq Khan, Hani Jamjoom, and Jimeng Sun. AIM-HI: a framework for request routing in large-scale IT global service delivery. *IBM Journal of Research and Development*, 53(6):4, 2009. [30](#), [44](#), [45](#), [47](#), [82](#)
- [80] Peng Sun, Shu Tao, Xifeng Yan, Nikos Anerousis, and Yi Chen. Content-aware resolution sequence mining for ticket routing. In *Business Process Management*, pages 243–259, 2010. [33](#), [34](#), [35](#), [36](#), [37](#), [38](#), [43](#), [44](#), [47](#), [82](#), [107](#), [116](#)
- [81] Liang Ma, Mudhakar Srivatsa, Derya Cansever, Xifeng Yan, Sue Kase, and Michelle Vanni. Query answering efficiency in expert networks under decentralized search. In *CIKM*, pages 2119–2124, 2016. [33](#), [44](#), [46](#), [56](#)
- [82] Gengxin Miao, Shu Tao, Winnie Cheng, Randy Moulic, Louise E Moser, David Lo, and Xifeng Yan. Understanding task-driven information flow in collaborative networks. In *WWW*, pages 849–858, 2012. [33](#), [34](#), [35](#), [44](#), [46](#), [83](#), [108](#)
- [83] L. M. Camarinha-Matos and H. Afsarmanesh. The emerging discipline of collaborative networks. In *Working Conf. on Virtual Enterprises*, pages 3–16, 2004. [33](#)
- [84] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003. [33](#)

- [85] Yi Chen, Shu Tao, Xifeng Yan, Nikos Anerousis, and Qihong Shao. Assessing expertise awareness in resolution networks. In *ASONAM*, pages 128–135, 2010. [34](#), [35](#), [44](#), [46](#), [83](#), [108](#)
- [86] T M Edgington, T S Raghu, and A S Vinze. Using process mining to identify coordination patterns in IT service management. *Decision Support Systems*, 49(2):175–186, 2010. [34](#)
- [87] C. Bartolini, C. Stefanelli, and M. Tortonesi. SYMIAN: Analysis and performance improvement of the IT incident management process. *IEEE Transaction on Network and Service Management*, 2010. [34](#)
- [88] C. Bartolini, C. Stefanelli, and M. Tortonesi. Synthetic incident generation in the reenactment of IT support organization behavior. In *IM*, 2013. [34](#), [39](#), [124](#)
- [89] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *ICDE*, 1995. [34](#)
- [90] G Jeong, S Kim, and T Zimmermann. Improving bug triage with bug tossing graphs. In *ESEC-FSE*, 2009. [38](#)
- [91] P Bhattacharya, I Neamtiu, and C R Shelton. Automated, highly-accurate, bug assignment using machine learning and tossing graphs. *Journal of Systems and Software*, 2012. [38](#)
- [92] Xin Xia, David Lo, Ying Ding, Jafar M Al-Kofahi, Tien N Nguyen, and Xinyu Wang. Improving automated bug triaging with specialized topic model. *IEEE Transactions on Software Engineering*, 43(3):272–297, 2016. [38](#)
- [93] H R Motahari-Nezhad and C Bartolini. IT support conversation manager: A conversation-centered approach and tool for managing best practice IT processes. In *EDCC*, 2010. [38](#)
- [94] Hamid Reza Motahari-Nezhad and Claudio Bartolini. Next best step and expert recommendation for collaborative processes in IT service management. In *Business Process Management*, pages 50–61, 2011. [38](#), [45](#), [81](#), [82](#)
- [95] Jianglei Han and Aixin Sun. Mean average distance to resolver: An evaluation metric for ticket routing in expert network. In *IEEE ICSME*, pages 594–602, 2017. [41](#)
- [96] Liang Ma. Query answering via decentralized search. Technical report, 2016. URL [QueryAnsweringviaDecentralizedSearch](#). [44](#)
- [97] Akila Venkatesan and Zayaraz Gothandapani. Novel metrics for bug triage. *Journal of Software*, 9(12):1–6, December 2014. [44](#)
- [98] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 43–52, 1998. [51](#)

- [99] Jianglei Han, Ka Hian Goh, Aixin Sun, and Mohammad Akbari. Towards effective extraction and linking of software mentions from user-generated support tickets. In *CIKM*, pages 2263–2271, 2018. [59](#), [85](#), [94](#), [112](#)
- [100] Xuelian Pan, Erjia Yan, Qianqian Wang, and Weina Hua. Assessing the impact of software on science: A bootstrapped learning of software entities in full-text papers. *Journal of Informetrics*, 9(4):860–871, 2015. [60](#)
- [101] Deheng Ye, Zhenchang Xing, Chee Yong Foo, Zi Qun Ang, Jing Li, and Nachiket Kapre. Software-specific named entity recognition in software engineering social content. In *IEEE Conference on Software Analysis, Evolution, and Reengineering*, volume 1, pages 90–101, 2016. [60](#), [66](#), [67](#), [69](#)
- [102] Deheng Ye, Zhenchang Xing, Chee Yong Foo, Jing Li, and Nachiket Kapre. Learning to extract api mentions from informal natural language discussions. In *International Conference on Software Maintenance and Evolution*, pages 389–399, 2016. [60](#)
- [103] Duangmanee Pew Putthividhya and Junling Hu. Bootstrapped named entity recognition for product attribute extraction. In *Conference on Empirical Methods in Natural Language Processing*, pages 1557–1567, 2011. [60](#)
- [104] Gabor Melli and Christian Romming. An overview of the cprod1 contest on consumer product recognition within user generated postings and normalization against a large product catalog. In *IEEE International Conference on Data Mining Workshops*, 2012. [60](#)
- [105] Sen Wu, Zhanpeng Fang, and Jie Tang. Accurate product name recognition from user generated content. In *IEEE International Conference on Data Mining Workshops*, pages 874–877, 2012. [60](#)
- [106] Yangjie Yao and Aixin Sun. Mobile phone name extraction from internet forums: a semi-supervised approach. *World Wide Web*, 19(5):783–805, 2016. [60](#), [61](#), [66](#), [67](#), [69](#), [73](#), [77](#)
- [107] Henry S Vieira, Altigran S da Silva, Pável Calado, Marco Cristo, and Edleno S de Moura. Towards the effective linking of social media contents to products in e-commerce catalogs. In *International Conference on Information and Knowledge Management*, pages 1049–1058, 2016. [61](#), [66](#), [67](#), [73](#), [74](#), [77](#)
- [108] Ea-Ee Jan, Kuan-Yu Chen, and Tsuyoshi Idé. Probabilistic text analytics framework for information technology service desk tickets. In *IFIP/IEEE Symposium on Integrated Network Management*, pages 870–873, 2015. [61](#)
- [109] Ea-Ee Jan, Jian Ni, Niyu Ge, Naga Ayachitula, and Xiaolan Zhang. A statistical machine learning approach for ticket mining in it service delivery. In *IFIP/IEEE Symposium on Integrated Network Management*, pages 541–546, 2013. [61](#)

- [110] Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. Brat: a web-based tool for nlp-assisted text annotation. In *European Chapter of the Association for Computational Linguistics*, pages 102–107, 2012. [62](#)
- [111] Shen Wei, Wang Jianyong, and Han Jiawei. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):443–460, 2015. [65](#)
- [112] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning*, pages 282 – 289, 2001. [66](#)
- [113] Charles Sutton and Andrew McCallum. An introduction to conditional random fields. *Foundations and Trends® in Machine Learning*, 4(4):267–373, 2012. [66](#), [74](#)
- [114] Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992. [67](#)
- [115] Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Conference on Computational Natural Language Learning*, pages 147–155, 2009. [67](#)
- [116] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv:1301.3781*, 2013. [67](#), [110](#)
- [117] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, 2014. [67](#), [110](#)
- [118] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017. ISSN 2307-387X. [67](#)
- [119] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Conference on Neural Information Processing Systems*, pages 3111–3119, 2013. [67](#)
- [120] Jiang Guo, Wanxiang Che, Haifeng Wang, and Ting Liu. Revisiting embedding features for simple semi-supervised learning. In *Conference on Empirical Methods in Natural Language Processing*, pages 110–120, 2014. [67](#)

- [121] Aria Haghighi and Dan Klein. Prototype-driven learning for sequence models. In *Conference of the North American Chapter of the Association for Computational Linguistics*, pages 320–327, 2006. [67](#)
- [122] Gerlof Bouma. Normalized (pointwise) mutual information in collocation extraction. *Proceedings of German Society for Computational Linguistics and Language Technology*, pages 31–40, 2009. [67](#)
- [123] Stefan Rüd, Massimiliano Ciaramita, Jens Müller, and Hinrich Schütze. Piggyback: Using search engines for robust cross-domain named entity recognition. In *Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 965–975, 2011. [68](#)
- [124] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. *Introduction to information retrieval*. Cambridge University Press, 2008. [68](#)
- [125] Chunyang Chen, Zhenchang Xing, and Ximing Wang. Unsupervised software-specific morphological forms inference from informal discussions. In *International Conference on Software Engineering*, pages 450–461, 2017. [69](#)
- [126] Nancy Chinchor and Beth Sundheim. Muc-5 evaluation metrics. In *Conference on Message Understanding*, pages 69–78, 1993. [72](#)
- [127] David Nadeau. *Semi-supervised named entity recognition: learning to recognize 100 entity types with little supervision*. PhD thesis, University of Ottawa, 2007. [72](#)
- [128] Chiu Jason P.C. and Eric Nichols. Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association for Computational Linguistics*, 4:357–370, 2016. [73](#)
- [129] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270, 2016. [73](#), [74](#)
- [130] Xuezhe Ma and Eduard Hovy. End-to-End Sequence Labeling via Bidirectional LSTM-CNNs-CRF. In *Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1064–1074, 2016. [73](#)
- [131] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Conference of the North American Chapter of the Association for Computational Linguistics*, 2018. [73](#)
- [132] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966. [73](#)

- [133] Jianglei Han, Jing Li, and Aixin Sun. Uftr: A unified framework for ticket routing. *Empirical Software Engineering*, In communication. [81](#), [111](#), [115](#), [118](#)
- [134] Jian Xu and Rouying He. Expert recommendation for trouble ticket routing. *Data & Knowledge Engineering*, 116:205–218, 2018. [81](#), [82](#), [83](#), [85](#), [98](#), [102](#), [107](#), [111](#), [116](#), [117](#)
- [135] Steve Cronen-Townsend, Yun Zhou, and W Bruce Croft. Predicting query performance. In *SIGIR*, pages 299–306, 2002. [86](#)
- [136] Krisztian Balog. *Entity-oriented Search*. Springer, 2018. [94](#)
- [137] Jingbo Shang, Liyuan Liu, Xiang Ren, Xiaotao Gu, Teng Ren, and Jiawei Han. Learning named entity tagger using domain-specific dictionary. In *EMNLP*, pages 2054–2064, 2018. [94](#)
- [138] Jon M Kleinberg. Hubs, authorities, and communities. *ACM computing surveys*, 31(4es):5, 1999. [94](#)
- [139] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999. [94](#)
- [140] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv:1607.04606*, 2016. [95](#), [114](#)
- [141] Tie-Yan Liu. *Learning to Rank for Information Retrieval*. Springer Science & Business Media, 2011. [95](#)
- [142] Yuan Tian, Dinusha Wijedasa, David Lo, and Claire Le Goues. Learning to rank for bug report assignee recommendation. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pages 1–10. IEEE, 2016. [95](#)
- [143] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. [97](#)
- [144] Qiang Wu, Christopher JC Burges, Krysta M Svore, and Jianfeng Gao. Adapting boosting for information retrieval measures. *Information Retrieval*, pages 254–270, 2010. [97](#)
- [145] Ali Mamdouh Elkahky, Yang Song, and Xiaodong He. A multi-view deep learning approach for cross domain user modeling in recommendation systems. In *WWW*, pages 278–288, 2015. [99](#), [106](#), [108](#), [109](#)
- [146] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. Fast matrix factorization for online recommendation with implicit feedback. In *SIGIR*, pages 549–558, 2016. [99](#)

- [147] Jianglei Han and Aixin Sun. Deeprouting: A deep neural network approach for ticket routing in expert network. In *IEEE Service Computing*, 2020. [105](#)
- [148] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry P. Heck. Learning deep structured semantic models for web search using clickthrough data. In *CIKM*, 2013. [106](#), [108](#), [110](#)
- [149] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. In *CIKM*, 2014. [110](#)
- [150] Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *WSDM*, pages 126–134, 2018. [108](#)
- [151] Jianfeng Gao, Patrick Pantel, Michael Gamon, Xiaodong He, and Li Deng. Modeling interestingness with deep neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2–13, 2014. [108](#)
- [152] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. Learning to match using local and distributed representations of text for web search. In *International Conference on World Wide Web*, pages 1291–1299, 2017. [106](#), [109](#)
- [153] Liang Ma, Mudhakar Srivatsa, Derya Cansever, Xifeng Yan, Sue Kase, and Michelle Vanni. Query answering efficiency in expert networks under decentralized search. In *International on Conference on Information and Knowledge Management*, pages 2119–2124, 2016. [108](#)
- [154] Dhruv Khattar, Vaibhav Kumar, Vasudeva Varma, and Manish Gupta. Weave&rec: A word embedding based 3-d convolutional network for news recommendation. In *CIKM*, pages 1855–1858, 2018. [108](#)
- [155] Xusheng Luo, Yonghua Yang, Kenny Qili Zhu, Yu Gong, and Keping Yang. Conceptualize and infer user needs in e-commerce. In *CIKM*, pages 2517–2525, 2019. [108](#)
- [156] Heung-Yeung Shum, Xiao-dong He, and Di Li. From eliza to xiaoice: challenges and opportunities with social chatbots. *Frontiers of Information Technology & Electronic Engineering*, 19(1):10–26, 2018. [108](#)
- [157] Jianfeng Gao, Michel Galley, Lihong Li, et al. Neural approaches to conversational ai. *Foundations and Trends® in Information Retrieval*, 13(2-3): 127–298, 2019. [108](#)
- [158] HongYun Cai, Vincent Wenchen Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30:1616–1637, 2017. [112](#)

-
- [159] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *ArXiv*, abs/1705.02801, 2018.
 - [160] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y. Chang. Network representation learning with rich text information. In *IJCAI*, 2015. [112](#)
 - [161] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations, ICLR*, 2017. [112](#)
 - [162] Matthew D Zeiler. Adadelata: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012. [113](#)
 - [163] Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009. [114](#)
 - [164] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 427–431, 2017. [114](#)