Towards Effective Extraction and Linking of Software Mentions from User-Generated Support Tickets

Jianglei Han[1,2], Ka Hian Goh[1,2], Aixin Sun[2], and Mohammad Akbari[1,3]
[1]SAP Leonardo ML, Singapore.
[2]School of Computer Science and Engineering, Nanyang Technological University, Singapore
[3]Department of Computer Science, University College London, United Kingdom
ray.han@sap.com;kgoh022@e.ntu.edu.sg;axsun@ntu.edu.sg;m.akbari@ucl.ac.uk

ABSTRACT

Software support tickets contain short and noisy text from users. Software products are represented by various surface forms and informal abbreviations created by users. Automatically identifying software mentions from tickets and determining the official names (and versions) are helpful for many downstream applications, *e.g.*, routing the support tickets to the right team of experts supporting the software. In this work, we study the problem of *software product name extraction* and *linking* from support tickets. We first analyze a collection of annotated tickets to understand the language patterns. Second, we design features using multiple in-domain and web knowledge sources, for the extraction and linking with linear models. Experiments on four datasets show better and more consistent results of our methods compared to neural network baselines.

1 INTRODUCTION

A European software company offers over a thousand applications in its product portfolio, covering a wide variety of business processes. These products are installed and used by hundreds of thousands of business customers around the world. A few thousand support engineers are employed and trained to provide technical assistance to such large customer base around-the-clock. Their jobs include receiving and analyzing support tickets from customers, before assisting them. During an investigation, one of the first clues to look for is to which software and version the problem is affecting. Some issues only affect specific versions of a product, while some incidents occur only after a version upgrade. From a ticket, extracting software mentions and correctly linking them to the product and version would benefit the whole process of software support. Examples are automatic ticket analysis and understanding [1, 18], advanced ticket search [9] and classification [12], root cause analysis [27], resolution recommendation [2, 44, 45] , and in-domain knowledge-base construction [37, 40].

Software support tickets contain structured customer related information and free text. The latter is user-generated, domain-specific, sometimes ambiguous. Many nouns are homonyms, and may refer to multiple objects or products. Figure 1 shows a sample

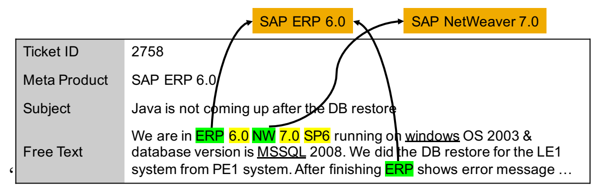

Figure 1: An example ticket, with product and version mentions and the official names of the linked products.

labeled ticket, in which “NW" is an abbreviation of `SAP NetWeaver`. Without mentioning an explicit version, it could be used to substitute to any version of the `SAP NetWeaver` family. Hence, contextual information is useful in determining the correct product in the ticket. On the other hand, a product may have multiple aliases created by customers. For example, our analysis of customer tickets found that `SAP NetWeaver 7.4` has 21 different known aliases. Moreover, as the tickets are human-generated, grammatical and typographical errors are unavoidable.

Existing information extraction applications on IT tickets mainly relies on text syntax and structure patterns, *e.g.,* Part-of-Speech (POS) tags [1, 28, 37]. Studies on other domain-specific named entity extraction and linking problems mainly use local and contextual information, with limited external knowledge [30, 36, 39, 41]. In this paper, we present a solution for extracting and linking software product names. Our main contributions are:

- We analyze a collection of manually annotated tickets, for their language patterns, especially on homonyms and synonyms of software product mentions (Section 3).
- We explore multiple resources and techniques in designing a rich set of domain-specific features for software mention recognition. A linear chain Conditional Random Field (CRF) model is then applied (Section 4).
- We evaluate the CRF model against the state-of-the-art deep neural network models.Results show that linear CRF models with deliberately designed domain features outperform neural models (Section 5).

2 RELATED WORK

Software entity extraction has been studied to understand software usages. Pan *et al.* [24] propose a bootstrapping method to extract software entities from scientific publications using contextual pattern matching techniques. Compared to formal articles, software tickets are significantly less formal and noisier. A different set of features and knowledge sources are required to cater for of name

CIKM'18, 22-26 October 2018, Lingotto, Turin, Italy

variations and other irregularities in our problem setting. Ye *et al.* [42, 43] investigate software-related terminologies (*e.g.,* programming languages, platforms, and APIs) extraction from Stack Overflow, a question-answering platform. Our work differs from theirs because product names to be extracted do not fit into their categorization. Further, we also perform linking of the extracted mentions to a catalog of formal products.

In an e-commerce context, Putthividhya and Hu [30] 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 [20], Wu *et al.* [39] propose a hybrid framework for product mention recognition and linking to a product catalog with a large number of products. Yao and Sun [41] investigate mobile phone names extraction and normalization using a novel semi-supervised labeling scheme to generate training data at scale. The weakly labeled data are used to train a linear CRF model to recognize mobile phone names from online forums.

Focusing on the linking aspect of the problem, Vieira *et al.* [36] use a binary classifier with features generated by exploring similarities between a mention and the official name and its description. The classifier is used to determine if a pair of mention and official name is correctly linked. Instead of using version as a supplementary feature, we label versions alongside the product names. Our model is trained to recognize both name and version and use them for product linking.

Our research is also related to analytical and predictive IT support systems [1, 13, 14, 29]. Most of these systems 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 on software product name extraction and linking from support tickets, leveraging both contextual and external information sources.

Table 1: Mentions of ‘`SAP NetWeaver 7.4`’ (top table) and a list of different products “NW” has been linked to (bottom table). Mistaken and exceptional 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 NetWeaver; SAP Net Weaver; SAP NW7.4; Ntweaver; NetWeaver AS; NW7.40; SAP netweaver <u>SAP PO</u>; <u>BW</u>	1
“NW” links to	**Count**
`SAP NetWeaver 7.4`	22
`SAP NetWeaver 7.3`	16
`SAP NetWeaver 7.0`	11
`SAP NetWeaver Enterprise Search 7.3`; `SAP enhancement package 1 for SAP NetWeaver 7.3`; <u>`SAP Solution Manager 7.1`</u>	1

Table 2: 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

3 THE PROBLEM AND DATA ANALYSIS

Given a support ticket, the task is to recognize a set of product name mentions $M = \{m_1, \ldots, m_N\}$ from the ticket, and map the mentions to their official names in a pre-defined product catalog $\mathcal{P} = \{p_1, \ldots, p_{|\mathcal{P}|}\}$, *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. A product name mention can be **linkable** or **unlinkable**, depending on whether there exists a matching entry in the catalog. In Figure 1, “ERP” and “NW” are linkable mentions, while “windows” and “MYSQL” are unlinkable products as there are no matching names in the catalog, *i.e.,* not supported by the company.

3.1 Dataset Annotation

To analyze the language usage and patterns, we randomly sample and annotate a subset of support tickets from a production system. A total of 3,369 English tickets are sampled randomly from four business product areas: service (SV), basic (BC), finance (FI), and business intelligence (BI). These product areas are expected to be distinctive in terms of distributions of labels and terms. Table 2 summaries the datasets.

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, we adopt a “pair labeling” technique. 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 four datasets are mixed and shuffled during annotation to avoid bias from annotators. We deployed a Brat [34] server, a web-based interactive annotation tool for labeling data corpus. The tickets are labeled with the following:

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

Annotators may use **meta-product** information that comes with the ticket to infer the official names for linking. Meta-product is provided by the customer when creating a ticket, see Figure 1. In this example, 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* would be a useful clue for the correct linking.

Unlike previous works [36, 41] considering version mentions as part of the product names, we label them as a standalone type of entities. In software tickets, versions may precede the product (*e.g.,*

Table 3: 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
All	1,587	705	1,413

Table 4: 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

"*... 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***").

3.2 Dataset Analysis

Entities. The entity level summary of the annotated data is shown in Table 3. Noticeably, FI has fewer labeled entities compared to other datasets, despite having the similar number of tickets as others. Meanwhile, 85% of mentions are unigrams or bigrams.

Product names. Table 4 reports 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. The user selected meta-product may not be consistent with the mention-level product linking. Our analysis shows that among four labeled datasets, only 30% of the tickets have a mention link to the meta-product. Its usefulness is also limited when multiple products are mentioned in a ticket.

Versions. We propose to extract version mentions along with product names and use both for linking. As shown in Table 5, BI has the most proportion of tickets having at least one version mention and the highest percentage of versions related to product entities. Specifically, 96.5% of the linkable product mentions and 96.2% of the unlinkable product mentions are associated with at least a version. Except for dataset FI, all other datasets have a reasonable portion of tickets containing at least one version mention (39.6% to 45.3%). The majority of cases saw the version token within three tokens proceeding or following its entity.

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

Table 5: Tickets, linkable and unlinkable product mentions, that are with 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%)

Table 1 shows examples of surface forms for `'SAP NetWeaver 7.4'` and their frequencies in the annotated tickets. The exceptional cases are underlined. They may be used due to legacy reasons, or by mistake. We observe that some surface forms are a 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, the surface forms have different degrees of similarity with their official names.

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

Mapping the recognized product mentions to official names are not trivial. It is especially challenging to disambiguate the same family of products with different versions. Customers may use the same acronym for different versions of a product. Table 1 shows all official products which 'NW' is linked to and their frequencies. The top-3 most frequent usages are to mention `'SAP NetWeaver 7.x'`. Other references are sparse and mostly inaccurate. During linking process, particular attention is required for version mentions.

Observation 3. *Annotation is challenging even for domain experts without using background knowledge in the topics.*

Unfortunately, there is no structured and queryable knowledge base, or a complete formal name to alias dictionary for the target domain. A challenge is to obtain useful information from external and web resources to support the process. Meanwhile, much research efforts on entity linking focus on linking entity mentions to their corresponding entities in a knowledge base (*e.g.,* Wikipedia) [38]. Additional information from descriptions and inter-entity links can be leveraged during linking. However, in many domain-specific applications, a knowledge base is not available. For instance, in our case, only a product catalog is available. We therefore attempt to explore inter-mention features, especially between product and version mentions, to improve pairwise linking performance.

4 MENTION EXTRACTION AND LINKING

Our system consists of two main modules for mention extraction and linking, respectively. Training the mention extraction module takes two inputs: the labeled tickets, and the product catalog. Multiple domain-specific features are generated from the labeled and unlabeled tickets. Using these features, we train a CRF-based extraction module and a Linking module.

4.1 Product Names Extraction

Software name extraction is a domain-specific named entity extraction problem, which is a sequence labeling task technically. We employ a linear chain Conditional Random Fields (CRF) [15] model with the features to be discussed in Section 4.2. CRF has been the state-of-art model used in related tasks [36, 41, 42]. Compared to the increasingly popular neural network-based models, linear chain CRF model is more flexible in incorporating a large number of customized input features, being able to learn from a relatively small amount of sampled data, and assigning weights to each feature[35].

4.2 Features

We use unlabeled tickets and external resources to generate the following groups of features.

Basic Features. Following previous studies [36, 41], we use lexical and grammatical features as a base model. Specifically, we consider the current word itself, its lowercased forms, prefixes, suffixes, and word shape. Part-of-Speech (POS) tags and prefixes generated from Stanford CoreNLP tool[1] are used as features too for each word and its neighbors in a context window size of 5.

Word Clusters. Word clustering techniques, such as Brown Clustering [5], have proven to be effective in mitigating term sparsity in open-domain and domain-specific NER systems [31, 41, 42]. It assumes that similar words should appear in similar contexts. At each iteration, it 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 on 3 million unlabeled tickets, the vocabulary is organized in a binary hierarchical tree structure, and each word can be represented using a cluster ID in bitstring. We use the prefixes of 10, 12, 14, 16, 18, 20 bits to represent a word as features.

Word embedding [3, 21, 25] is a significant recent advancement in distributed text representation. Unlike Brown Clustering, word embedding produces dense vectors after the training step. Semantically similar words are close to each other in the embedding space. We apply three different word embedding models, namely Word2Vec [22], Glove [25] and FastText [3] on the same unlabeled corpus to generate word vectors 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 ID as word features.

Prototype Words. Guo *et al.* [10] compare different approaches in incorporating trained word vectors into a linear CRF model and propose a novel method based on *prototype-driven learning* [11]. It assumes that the semantically similar words should be tagged with the same label. We generate prototype features for each entity label by analyzing the collocation of labels and words. Specifically, the normalized pointwise mutual information (NPMI) [4] is computed for a label l and word w using $NPMI(l,w) = \frac{\lambda(l,w)}{-\ln p(l,w)}$, where $\lambda(l,w) = \ln \frac{p(l,w)}{p(l)p(w)}$ is the standard PMI. In labeled text corpus, for each entity label, we compute NPMI with every word in the vocabulary then rank the words in descending order by NPMI. The top 5 words for each label are shown in Table 6. The feature for the token label is set to 1 if the word is among the prototypes of the label.

[1]https://stanfordnlp.github.io/CoreNLP/

Table 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

Results from Web Search. Public accessible search engines can be useful in determining the validity of an entity or a phrase to a domain. For example, if a span of words is a valid software name or related to the field, the search engine will return more results from computer-related links. Similar to Rüd *et al.* [32], we explore results returned from a search engine[2] for domain relatedness of tokens. Specifically, we use a token and its surrounding words to form a query as input to a search engine and collect the top-k results ($k = 50$ in our experiment). Each result entry contains a title, a summary, and an URL. Subsequently, we compute the ratio of results having queries in the 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.

Despite of its usefulness, querying search engines at a large volume is not free of cost. For our experiments, we preselect the search queries using a combination of POS tag patterns, query likelihood score [33], and heuristics. Query likelihood score is the joint probability of neighboring tokens, obtained using a language model trained on unlabeled tickets. Eventually, the query likelihood scores and the ratios computed from search engine results, are rounded to the nearest 0.1 and used as categorical features.

Dictionary Construction. Previous works [41, 42] pre-define a dictionary to improve the NER performance. We propose an approach to create dictionary automatically using a combination of semantic similarity and heuristics. Chen *et al.* [6] propose an unsupervised method to build software-specific dictionary of synonyms and acronyms, using distributed word similarities. Inspired by their approach, we design a dictionary generation algorithm (Algorithm 1) using mentions labeled in training data as seed names.

Specifically, a mention dictionary is first built from training data. We observe that a mention m could contain overlapping tokens with the official product name p. We therefore consider each token of p as candidates. Similar to the rules in [41] in Line 5, we design the following rules to generate candidates from p.

(1) p less the token that is a company name or the version, *e.g.,* `'SAP NetWeaver 7.4'` ⟶ `'NetWeaver 7.4'`, `'NetWeaver'`.
(2) All capital letters in p less the token that is a company name or the version, *e.g.,* `'SAP NetWeaver 7.4'` ⟶ `'NW 7.4'`, `'NW'`.
(3) Capitalized first letters of all tokens in p less the token that is a company name or the version *e.g.,* `'SAP BusinessObjects Business Intelligence platform 4.2'` ⟶ `'BOBJ 4.2'`, `'BOBI'`

[2]https://azure.microsoft.com/en-gb/services/cognitive-services/bing-web-search/

Algorithm 1: Dictionary generation

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

Output : C, list of names of p

1 $C \leftarrow \emptyset$;
2 **foreach** *ticket* $t \in \mathcal{T}_{Train}$ **do**
3   **foreach** $m \mapsto p$ *in* t **do**
4     $C \leftarrow Tokenize(p) \cup RuleGenerator(p)$;
5     $N \leftarrow NearestNeighbor(m \cup C, E)$;
6     $C \leftarrow C \cup \{m\} \cup \{N\} \cup \{R\} \cup LowerCase(C)$\$;
7   **end**
8 **end**

In Line 5, the nearest neighbors are obtained from a trained word embedding model. Specifically, the FastText model enables construction of word vectors for unseen words in training corpus on the fly. To enable the iterative extension of the dictionary, we choose FastText model for its generation. A binary feature is generated for a candidate token exists in the dictionary.

We also crawl software names from crowd-sourced knowledge base Wikipedia[3]. Though the names are neither official nor complete, it is freely accessible with reasonable quality. We post-process the list by converting all names to lowercase and remove the numbers and use it as another external resource for the extraction.

4.3 Product Name Linking

The output of the extraction module will be the input 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. Note that the version mention v and meta-product t may not be always available for every mention m.

We employ a pairwise linking model. 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 the linear combination of features, *i.e.,* $\phi(m, p) = \sum_i \lambda_i f_i(e, m)$. In specific to our problem setting, an additional entity v and a ticket level product t, are also included to the model.

We employ a Support Vector Regression (SVR) model to estimate the probability ϕ, using the features detailed in Table 7. In training data, the labeled $\langle m, p \rangle$ pairs are positive samples. A negative pair $\langle m, p\prime \rangle$ is generated with a random $p\prime \in \mathcal{P} \wedge p\prime \neq p$ for each positive sample. 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.

[3] https://en.wikipedia.org/wiki/List_of_SAP_products

Table 7: Features for linking a mention m with version v, to an official product p, with ticket-level meta-product t. $[\cdot]$ is the linear combination of embedding vectors.

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.

5 EXPERIMENTS

We now evaluate the product name extraction and linking modules in isolation. We then examine outputs from the end-to-end system with respect to error analysis.

Experimental Setup. For each labeled dataset detailed in Section 3, we randomly split the data into 70/10/20 percents as training, development, and testing tickets.

Evaluation Metrics. We use *Precision* (*Pr*), *Recall* (*Re*), and $F1$ measure for both modules. The NER module is first evaluated at entity level with strict matching for the entity type. That is, 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) share task [7] to provide a lenient version of *Pr*, *Re*, and $F1$, considering partial matching [23].

Comparing system outputs with the golden labels, MUC accounts for four 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 $Precision_{MUC} = \frac{\alpha+\beta}{\gamma}$ and $Recall_{MUC} = \frac{\alpha+\beta}{\delta}$. Lastly, $F1_{MUC} = \frac{2 \times Precision_{MUC} \times Recall_{MUC}}{Precision_{MUC} + Recall_{MUC}}$.

5.1 Baseline methods

In recent years, deep neural network models have been increasingly popular in information extraction applications. It is interesting to compare the performance of the non-neural CRF model using our proposed features with neural models. To this end, we select four popular deep neural network models for named entity extraction as baselines [8, 16, 19, 26]. These models are different in their encoding and output components. Except Chiu *et al.* [8], the

Table 8: Linear Chain CRF model with the proposed features, compared with neural network-based baseline models. MUC is a lenient measure considering partial matching with golden label. For each row and each metric, the best score is in boldface.

Entity	CNN-LSTM-SOFTMAX			LSTM-LSTM-CRF			CNN-LSTM-CRF			GRU-GRU-CRF			**CRF**		
Measure	*Pr*	*Re*	*F*1	*Pr*	*Re*	*F*1	*Pr*	*Re*	*F*1	*Pr*	*Re*	*F*1	*Pr*	*Re*	*F*1
Dataset: SV															
Linkable	0.800	0.899	0.847	**0.876**	0.930	**0.902**	0.859	**0.946**	0.900	0.845	0.930	0.886	0.858	0.938	0.896
Unlinkable	**1.000**	**0.500**	**0.667**	0.000	0.000	0.000	**1.000**	**0.500**	**0.667**	**1.000**	**0.500**	**0.667**	**1.000**	**0.500**	**0.667**
Version	0.766	**0.900**	0.828	0.850	0.850	0.850	**0.872**	0.850	**0.861**	**0.872**	0.850	**0.861**	0.846	0.825	0.835
Average	0.855	0.766	0.780	0.863	**0.890**	**0.876**	**0.910**	0.765	0.809	0.906	0.760	0.804	0.901	0.754	0.799
MUC	0.803	0.906	0.852	**0.873**	0.904	0.888	0.865	0.921	0.892	0.863	0.918	0.889	0.869	**0.934**	**0.899**
Dataset: BC															
Linkable	0.721	**0.771**	0.745	0.767	0.756	0.761	0.784	0.664	0.719	0.808	0.740	0.773	**0.874**	0.740	**0.802**
Unlinkable	0.500	**0.635**	0.560	0.600	0.529	0.562	**0.623**	0.447	0.520	0.620	0.576	**0.598**	0.569	0.482	0.522
Version	0.781	**0.824**	0.802	0.872	**0.824**	0.847	0.842	0.703	0.766	0.835	0.780	0.807	**0.986**	0.769	**0.864**
Average	0.668	**0.743**	0.702	0.747	0.703	0.724	0.750	0.605	0.669	0.755	0.699	0.726	**0.810**	0.664	**0.729**
MUC	0.682	**0.764**	0.720	0.767	0.725	0.745	0.780	0.630	0.697	0.776	0.718	0.746	**0.825**	0.682	**0.747**
Dataset: FI															
Linkable	0.600	0.692	0.643	0.600	0.692	0.643	0.615	0.615	0.615	0.615	0.615	0.615	**0.769**	**0.769**	**0.769**
Unlinkable	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Version	0.714	0.455	0.556	0.750	0.545	0.632	0.800	0.364	0.500	0.667	0.182	0.286	**1.000**	**0.545**	**0.706**
Average	0.657	0.455	0.556	0.675	0.619	0.637	0.708	0.490	0.558	0.641	0.399	0.451	**0.885**	**0.657**	**0.738**
MUC	0.659	0.604	0.630	0.674	0.646	0.660	0.694	0.521	0.595	0.656	0.437	0.525	**0.842**	**0.667**	**0.744**
Dataset: BI															
Linkable	0.769	0.843	0.805	**0.861**	0.747	0.800	0.859	0.735	0.792	0.851	0.759	0.802	**0.861**	0.819	**0.839**
Unlinkable	0.684	0.783	0.730	0.750	0.783	0.766	0.750	0.739	0.744	0.732	0.754	0.743	**0.862**	**0.812**	**0.836**
Version	0.916	0.884	0.899	0.885	0.895	0.89	0.858	**0.913**	0.884	**0.922**	0.890	**0.905**	0.901	0.901	0.901
Average	0.789	0.837	0.811	0.832	0.808	0.819	0.822	0.796	0.807	0.835	0.801	0.817	**0.874**	**0.844**	**0.859**
MUC	0.839	**0.870**	0.854	0.862	0.846	0.854	0.846	0.841	0.844	0.876	0.841	0.858	**0.886**	0.864	**0.875**
Mean MUC	0.745	0.786	0.764	0.794	0.780	0.786	0.796	0.728	0.757	0.792	0.728	0.754	**0.855**	**0.787**	**0.816**

other models all use a CRF output layer. We represent these models by their [character encoder]-[word encoder]-[output] triplets for simplicity. These models are **CNN**[4]**-LSTM**[5]**-SOFTMAX** [8], **LSTM-LSTM-CRF** [16], **CNN-LSTM-CRF** [19] and **GRU**[6]**-GRU-CRF** [26]. Noted that both LSTM and GRU models are bidirectional, to encode contextual information from both left and right side of the current token. All four baselines models are claimed to be outperforming linear models on open domain NER tasks. However, the details on each are beyond the scope of this paper.

The performance of neural models is subjective to parameters tuning. We adopt the same common parameters across the models, 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 development set and evaluated on their respective testing set.

For linking module, we compared with **Fuzzy string matching** baselines using different queries, as well as the linking methods from related works, **GLEN** [41], and **ProdLink** [36]. Fuzzy string matching is to compute Levenshtein Distance [17] between query string q and each p_i. Three variations of q are evaluated: $q = m$, $q = m + v$, and $q = m + v + t$, where + means 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 [41] is a rule-based linking method designed specifically for mobile phone names normalization. We modify the approach using a candidate voting with confidence score computed in a similar way. 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 4.2. p with the most congregated vote is linked to m. We heuristically set k to 200.

ProdLink [36] uses features extracted from m and the product description of p to train a binary classifier for linking. Subsequently, a string similarity score is computed to select the most likely linking object. Since we do not have the detailed description of p in our catalog, we use p to generate the closest token-based feature set outlined in [36] as a baseline.

[4]Convolutional Neural Network: A kind neural network for feature extraction.
[5]Long Short Term Memory. A variation of the Reccurent Neural Network (RNN) model.
[6]Gated Recurrent Unit. Similar to LSTM but with fewer trainable parameters.

Table 9: 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**
- Dictionaries	0.899	0.705	0.634	0.848
- Word clusters	0.884	0.732	0.714	0.870
- Search results	0.881	0.737	0.683	0.880
- Prototypes	0.880	0.739	0.698	0.870
Base model	0.859	0.695	0.667	0.850

5.2 Experimental Results

Figure 8 reports the detailed results, having datasets in horizontal blocks and methods in vertical blocks. Within a block, each row corresponds to an entity type, including the average and MUC metrics. The scores of neural network models fluctuate in different datasets, due to label sparsity and data distribution. In comparison, the performance of CRF model with rich features is much more consistent, outperforming the baselines in MUC metrics. Notably, in dataset FI, where the labels are sparse, CRF model exceeds by as much as 25% and 13% compared to the best neural model in precision and F1, respectively. The average MUC metrics across all four datasets show the clear advantage of the feature-rich linear chain CRF model in our experiments.

Among the neural models, the simple CNN-LSTM-SOFTMAX model using a linear output layer shows better recall but slightly poorer precision in some datasets. A question raised is how the CRF layer would affect a neural model for NER from a noisy text. In [35], the author states that it is recommended to use binned features instead of real-valued features as input to a CRF model for more performance gain. In neural models, the output from the word encoding/hidden layer is used directly as input to the CRF layer for tagging output [16]. Performance improvements are reported when applied on news articles. It could be the noise in user-generated text like software tickets that limit the effectiveness of the CRF layer. On the other hand, using CNN for character feature extraction exhibits slight advantage compared to using a LSTM encoder. Overall, the neural network models show some potential in this task. We will leave the detailed investigation in neural network models for domain-specific NER to future work, with more labeled data.

To understand the impact of individual feature groups in our CRF model, we conduct ablation study on MUC $F1$ scores. The results after removing dictionaries, word clusters, search results, and prototypes are shown in Table 9. The results in the last row are from a CRF model only using the basic features. Models without using dictionary features observe the most significant decline in MUC $F1$. The other feature groups have less influence on the overall performance. The exact impact varies on different datasets.

In Table 10, we show the most useful features from the trained CRF models on each dataset, per entity type. The weights corresponds to the relative influence of a feature on the result of a model. The dictionary features are dominant for linkable product names across datasets. Prototype features are more effective for identifying unlinkable product names. While the surface form mentions are the most important features for version extraction. Word substring, word shape, and word cluster features are useful for all entity types. In comparison, the significance of the search result features are less permanent, even compared with the query likelihood feature which is used alongside to quantify the quality of a phrase. In overall, the features generated from external resources improve the MUC F1 scores on the majority of datasets.

Table 11 compares the linking performance of our method using different types of queries, with baseline methods. The results show the advantage of including 'version' (*i.e.,* M2), compared to only using m (M1) in the query. Most notably on dataset BI, incorporating version information gives 74% and 120% increase in precision and recall, respectively compared to the lowest fuzzy matching baseline. Improvements are observed on other datasets too. It confirms the usefulness of using version entities for product linking in our task. In contrast, the impact of having ticket-level meta-product t in the model (M3) is less conclusive. These results are not surprising as only about 30% of the tickets have a mention associate to meta-product. After all, M2 and M3 almost dominate the top two scores in all metrics across four datasets. Fuzzy string matching methods show some reasonable precision but low in recall. GLEN has good potential, using only statistical acronym linkages.

5.3 Error analysis

To study the end-to-end system, we use the output from extraction module as the input to the linking module. Since the specific version-product relation is missing, we assume any version token in a window of flexible size is associated to a product mention, which is at position 0. Specifically, the left boundary of the window is $\max(-2, pos_m)$ and the right boundary is $\min(6, pos_m)$, where pos_m is the relative position of another product mention. When multiple tokens are present in the valid window, all are considered relevant to it. In Figure 2, we use three test tickets with results produced by our system to illustrate typical positive outcomes and errors. Specifically, we focus on the three types of errors.

Error Type 1: Wrong entity. In Ticket 106, a product mention "Solution Manager" is correctly recognized by NER module. Despite the absence of any version token, 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 entity.

Error Type 3: Missing entity Our NER system failed to recognize mentions from ticket 196. The term "Sfin" is a rare acronym of `'SAP Simple Finance'` in training. For such case, pattern matching techniques that capture the immediate preceding words can be applied to complement the NER system.

6 CONCLUSION

In this work, we investigate software product name extraction and linking problem from support tickets. We analyze the language patterns by annotating tickets from production support systems,

Table 10: The most effective CRF features with weights.

	SV Feature	SV Weight	BC Feature	BC Weight	FI Feature	FI Weight	BI Feature	BI 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
	word[:1]	0.510	query likelihood	1.617	word[:1]	0.845	prototype	0.754
Unlinkable	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
	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
Version	mentions	3.256	mentions	3.547	mentions	1.211	mentions	3.024
	word.endssdigit	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 11: Linking module results. For each column, the best score is in boldface and the second best is underlined.

Dataset	SV			BC			FI			BI		
Method	*Pr*	*Re*	*F1*	*Pr*	*Re*	*F1*	*Pr*	*Re*	*F1*	*Pr*	*Re*	*F1*
Fuzzy Matching $q = m$	0.874	0.193	0.316	0.341	0.144	0.202	0.047	0.081	0.059	0.174	0.163	0.168
Fuzzy Matching $q = (m, v)$	$\underline{0.921}$	0.273	0.421	0.535	0.151	0.236	0.155	0.153	0.154	0.188	0.113	0.141
Fuzzy Matching $q = (m, v, t)$	0.606	0.174	0.270	0.242	0.050	0.083	0.011	0.083	0.019	0.011	0.011	0.011
GLEN [41]	0.872	0.303	0.450	0.581	0.505	0.540	0.515	0.622	0.563	0.670	0.456	0.543
ProdLink [36]	0.119	0.126	0.118	0.503	0.385	0.341	$\underline{0.588}$	$\underline{0.692}$	$\underline{0.631}$	0.284	0.232	0.252
Our method M1: $q = m$	0.856	$\underline{0.669}$	0.748	0.723	0.392	0.441	$\underline{0.588}$	$\underline{0.692}$	$\underline{0.631}$	0.433	0.232	0.270
Our method M2: $q = (m, v)$	**0.932**	**0.685**	**0.783**	$\underline{0.708}$	$\underline{0.508}$	$\underline{0.557}$	**0.703**	**0.769**	**0.734**	**0.753**	$\underline{0.512}$	$\underline{0.588}$
Our method M3: $q = (m, v, t)$	0.916	0.661	$\underline{0.757}$	**0.768**	**0.538**	**0.607**	$\underline{0.588}$	$\underline{0.692}$	$\underline{0.631}$	$\underline{0.749}$	**0.537**	**0.624**

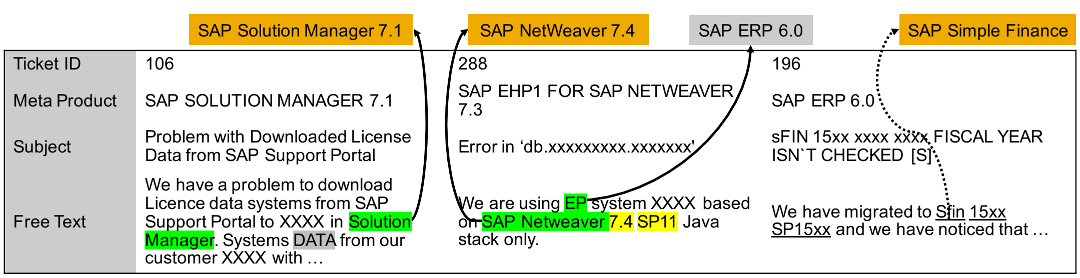

Figure 2: Results from the end-to-end system on three tickets. True positive results from product and version extraction are highlighted in green and yellow respectively. False negative entities are underlined. Positive linkings are in solid curves and orange boxes, while the false negative linkings are dashed curves. False positive linking is grey.

then use the insights drawn to design features for extraction and linking models. We demonstrate the effectiveness of our features in both extractions and linking modules. Our models outperform deep learning based baselines and are more consistent across datasets, even with sparse labels. For future work, we plan to investigate further into the increasingly maturing neural network models for domain-specific information extraction, combining the advantages of linear models and neural models and performing extraction and linking simultaneously. We are also interested in exploring semi-supervised methods leveraging unlabeled data for information extraction from support tickets.

ACKNOWLEDGMENTS

The first author is sponsored by SAP Industrial Ph.D Program, partially funded by the Economic Development Board and the National Research Foundation of Singapore. The authors would like to thank David Molinera for his assistance in data preparation.

REFERENCES

[1] Shivali Agarwal, Vishalaksh Aggarwal, Arjun R Akula, Gargi Banerjee Dasgupta, and Giriprasad Sridhara. 2017. Automatic problem extraction and analysis from unstructured text in IT tickets. *IBM Journal of Research and Development* 61, 1 (2017), 4–41.

[2] Vishalaksh Aggarwal, Shivali Agarwal, Gaargi B Dasgupta, Giriprasad Sridhara, and Vijay E. 2016. ReAct: A System for Recommending Actions for Rapid Resolution of IT Service Incidents. In *IEEE International Conference on Services Computing*. 1–8.

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics* 5 (2017), 135–146.

[4] Gerlof Bouma. 2009. Normalized (pointwise) mutual information in collocation extraction. *Proceedings of German Society for Computational Linguistics and Language Technology* (2009), 31–40.

[5] Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. 1992. Class-based n-gram models of natural language. *Computational linguistics* 18, 4 (1992), 467–479.

[6] Chunyang Chen, Zhenchang Xing, and Ximing Wang. 2017. Unsupervised software-specific morphological forms inference from informal discussions. In *International Conference on Software Engineering*. 450–461.

[7] Nancy Chinchor and Beth Sundheim. 1993. MUC-5 evaluation metrics. In *Conference on Message Understanding*. 69–78.

[8] J.P.C. Chiu and E Nichols. 2016. Named entity recognition with bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics* 4 (2016), 357–370.

[9] Yu Deng, KE Maghraoui, TD Griffin, V Agarwal, SG Tamilselvam, RD Sharnagat, TH Alexander, NE Gómez, CM Cramer, A Bivens, and others. 2017. Advanced search system for IT support services. *IBM Journal of Research and Development* 61, 1 (2017), 3–27.

[10] Jiang Guo, Wanxiang Che, Haifeng Wang, and Ting Liu. 2014. Revisiting Embedding Features for Simple Semi-supervised Learning.. In *Conference on Empirical Methods in Natural Language Processing*. 110–120.

[11] Aria Haghighi and Dan Klein. 2006. Prototype-driven learning for sequence models. In *Conference of the North American Chapter of the Association for Computational Linguistics*. 320–327.

[12] Jianglei Han and Mohammad Akbari. 2018. Vertical Domain Text Classification: Towards Understanding IT Tickets Using Deep Neural Networks. In *AAAI Conference on Artificial Intelligence*.

[13] Ea-Ee Jan, Kuan-Yu Chen, and Tsuyoshi Idé. 2015. Probabilistic text analytics framework for information technology service desk tickets. In *IFIP/IEEE Symposium on Integrated Network Management*. 870–873.

[14] Ea-Ee Jan, Jian Ni, Niyu Ge, Naga Ayachitula, and Xiaolan Zhang. 2013. A statistical machine learning approach for ticket mining in IT service delivery. In *IFIP/IEEE Symposium on Integrated Network Management*. 541–546.

[15] John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning*. 282 – 289.

[16] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural Architectures for Named Entity Recognition. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 260–270.

[17] Vladimir I Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, Vol. 10. 707–710.

[18] Ta Hsin Li, Rong Liu, Noi Sukaviriya, Ying Li, Jeaha Yang, Michael Sandin, and Juhnyoung Lee. 2014. Incident Ticket Analytics for IT Application Management Services. In *IEEE International Conference on Services Computing*. 568–574.

[19] Xuezhe Ma and Eduard Hovy. 2016. End-to-End Sequence Labeling via Bidirectional LSTM-CNNs-CRF. In *Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. 1064–1074.

[20] Gabor Melli and Christian Romming. 2012. 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*.

[21] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv:1301.3781* (2013).

[22] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Conference on Neural Information Processing Systems*. 3111–3119.

[23] David Nadeau. 2007. *Semi-supervised named entity recognition: learning to recognize 100 entity types with little supervision.* Ph.D. Dissertation. University of Ottawa.

[24] Xuelian Pan, Erjia Yan, Qianqian Wang, and Weina Hua. 2015. Assessing the impact of software on science: A bootstrapped learning of software entities in full-text papers. *Journal of Informetrics* 9, 4 (2015), 860–871.

[25] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Conference on Empirical Methods in Natural Language Processing*. 1532–1543.

[26] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Conference of the North American Chapter of the Association for Computational Linguistics*.

[27] Karthikeyan Ponnalagu. 2017. Ontology-driven root-cause analytics for user-reported symptoms in managed IT systems. *IBM Journal of Research and Development* 61, 1 (2017), 5–53.

[28] Rahul Potharaju, Joseph Chan, Luhui Hu, Cristina Nita-Rotaru, Mingshi Wang, Liyuan Zhang, and Navendu Jain. 2015. ConfSeer: Leveraging Customer Support Knowledge Bases for Automated Misconfiguration Detection. *Proceedings of the VLDB Endowment* 8, 12 (2015).

[29] Rahul Potharaju, Navendu Jain, and Cristina Nita-Rotaru. 2013. Juggling the Jigsaw: Towards Automated Problem Inference from Network Trouble Tickets.. In *USENIX Symposium on Networked Systems Design and Implementation*. 127–141.

[30] Duangmanee Pew Putthividhya and Junling Hu. 2011. Bootstrapped named entity recognition for product attribute extraction. In *Conference on Empirical Methods in Natural Language Processing*. 1557–1567.

[31] Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Conference on Computational Natural Language Learning*. 147–155.

[32] Stefan Rüd, Massimiliano Ciaramita, Jens Müller, and Hinrich Schütze. 2011. 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*. 965–975.

[33] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. 2008. *Introduction to information retrieval.* Cambridge University Press.

[34] Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. 2012. BRAT: a web-based tool for NLP-assisted text annotation. In *European Chapter of the Association for Computational Linguistics*. 102–107.

[35] Charles Sutton and Andrew McCallum. 2012. An introduction to conditional random fields. *Foundations and Trends® in Machine Learning* 4, 4 (2012), 267–373.

[36] Henry S Vieira, Altigran S da Silva, Pável Calado, Marco Cristo, and Edleno S de Moura. 2016. Towards the Effective Linking of Social Media Contents to Products in E-Commerce Catalogs. In *International Conference on Information and Knowledge Management*. 1049–1058.

[37] Qing Wang, Wubai Zhou, Chunqiu Zeng, Tao Li, Larisa Shwartz, and Genady Ya Grabarnik. 2017. Constructing the knowledge base for cognitive IT service management. In *IEEE International Conference on Services Computing*. 410–417.

[38] Shen Wei, Wang Jianyong, and Han Jiawei. 2015. Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions. *IEEE Transactions on Knowledge and Data Engineering* 27, 2 (2015), 443–460.

[39] Sen Wu, Zhanpeng Fang, and Jie Tang. 2012. Accurate product name recognition from user generated content. In *IEEE International Conference on Data Mining Workshops*. 874–877.

[40] Shuo Yang, Lei Zou, Zhongyuan Wang, Jun Yan, and Ji-Rong Wen. 2017. Efficiently Answering Technical Questions âĂŤ A Knowledge Graph Approach. In *AAAI Conference on Artificial Intelligence*.

[41] Yangjie Yao and Aixin Sun. 2016. Mobile phone name extraction from internet forums: a semi-supervised approach. *World Wide Web* 19, 5 (2016), 783–805.

[42] Deheng Ye, Zhenchang Xing, Chee Yong Foo, Zi Qun Ang, Jing Li, and Nachiket Kapre. 2016. Software-specific named entity recognition in software engineering social content. In *IEEE Conference on Software Analysis, Evolution, and Reengineering*, Vol. 1. 90–101.

[43] Deheng Ye, Zhenchang Xing, Chee Yong Foo, Jing Li, and Nachiket Kapre. 2016. Learning to extract api mentions from informal natural language discussions. In *International Conference on Software Maintenance and Evolution*. 389–399.

[44] Wubai Zhou, Tao Li, Larisa Shwartz, and Genady Ya Grabarnik. 2015. Recommending ticket resolution using feature adaptation. In *International Conference on Network and Service Management*. 15–21.

[45] Wubai Zhou, Liang Tang, Tao Li, Larisa Shwartz, and Genady Ya Grabarnik. 2015. Resolution recommendation for event tickets in service management. In *IFIP/IEEE Symposium on Integrated Network Management*. 287–295.