Neev: A Cognitive Support Agent for Content Improvement in Hardware Tickets

Nishtha Madaan
IBM Research-India
nimadaan@in.ibm.com

Gautam Singh
IBM Research-India
gautamsi@in.ibm.com

Arun Kumar
IBM Research-India
kkarun@in.ibm.com

Gargi B Dasgupta
IBM Research-India
Gaargi.Dasgupta@in.ibm.com

***Abstract*—IT service providers differentiate themselves through offering after-sales support for hardware and software products. Thus, businesses, including large corporations, have intricate work-flows for servicing such support requests while reducing man-hours needed. These work-flows generally operate through a ticketing system for resolving customer issues. A lot of man-hours are spent in searching old tickets for correct problem and resolution for such issues. Support requests pertaining to enterprise hardware are more challenging than desktop support for end-user products. Enterprise hardware requires deeper diagnosis involving several systems and expertise of multiple agents. In this work we propose a cognitive agent, Neev, which helps in mitigating the problem in a three-fold fashion (1) retrieving a summary of relevant ticket text (2) Tagging the relevant parts as a part-of-the-problem or a part-of-the-solution (3) Focusing on the precise problem and solution. We evaluate the performance of our system using a rank-based metric where a ticket extraction is successful if the problem or solution occur in the top-n suggestions. We report the results for varying top-n values for both problem and solution on varying severity of the tickets. We find that the accuracy for problem extraction in top-1 is 62% and it reaches 86% and 94% for top-3 and top-5 cases, respectively. Furthermore, the accuracy for solution extraction reaches 62% and 88% for top-3 and top-8 cases, respectively.**

I. Introduction

Businesses not only compete to provide best-in-class hardware products but also in providing fast, targeted and low-cost after-sales services. After-sales support for hardware is more challenging and costly than for software as it requires on-site visits, troubleshooting and part replacements. To add to the challenge, there are a large number of client locations and part stocking locations which need to be managed to minimize time-distance costs. A customer or an on-site professional can report an after-sales requests through a ticketing system. Usually, the problem is reported via a web portal or through voice which are then converted into a ticket. For incidents reported through web, duplicate incidents are eliminated and a ticket is raised in ticket management system. For the incidents that are reported through phone, an agent raises a ticket in ticket management system and transcribes the conversation. There are two ways tickets are handled in ticket management system : 1) Automatic handling, where in a ticket is resolved by an automation engine. The back-end of this engine could be implemented using a library of predefined rules. On detecting a rule-match in the ticket, the resolution is obtained from the rule-library and returned to the ticket originator. 2) Manual handling, where an incoming ticket is initially assigned to a level-1 support agent. If ticket is not resolved at this stage it is escalated to a level-2 support agent and similarly to a level-3 support agent. Agents at all the three levels can additionally consult subject matter experts who may further take help from the product development teams to obtain their expertise in resolving the ticket. Moreover, they look for a fix in the past case-records or in product manuals before recommending a preliminary fix. Once the resolution is found the issue is fixed either on the phone or by dispatching a field agent and the ticket is closed.

The higher the level of support that is required to resolve a ticket, the more expensive support operations become. Thus there is a strong need for a cognitive agent which facilitates manual ticket resolution at initial levels of support rather than consulting the more experienced agents at higher levels. To this end, the cognitive agent should be able to reduce the time to go over each ticket and improve agent's understanding.

When a support agent initially reads through the problem, he performs a search on a myriad of support portals and attempts to reach the solution. The search results from such portals are appended to the ticket text. Further when ticket passes through various gateways while they are redirected from one level to other, an automated machine generated text is appended to the ticket. Further when agents consult subject matter experts and product development teams through emails or web-chats, they append these transcripts to the ticket text. This process of updating ticket at each intermediate step creates a confused mixture containing structured and unstructured pieces of information.

Multiple communications leads to multiple instances of same problem getting logged in the ticket. Also, same resolution might be repeatedly logged in the ticket while communicating either with the customer or with the field agent. Sometimes the customer might describe their own diagnosis which is recorded in the ticket and might not be useful. Hence, root cause analysis and problem resolution becomes a challenge.

The structure of this paper is as follows. Section 2 talks about the state- of-the-art and our contribution. In Section 3, we describe the ticket data set. Section 4 presents a bird-eye view of our system and then describes the details. In section 5, we walk the reader through an example ticket presenting our approach. Evaluation and results are contained in section 6. In section 7, we present the concluding remarks and future work.

Cognitive research today focuses on combining the best of man and machine. Cognitive agents can be classified in two different classes 1) which directly interact with the customer, 2) which assist a human agent in interacting with the customer. Our proposed cognitive agent, Neev, belongs to the latter category. It processes the tickets to extract ranked problems and resolution from tickets. Neev attempts to cut down the agent's time in reading the large amount of literature containing grammatical errors, noise and rephrasal of similar content. Less experienced agents would find themselves at ease in grasping the contents of the past records. Additionally this generates problem-solution pairs which can be used for curation and training Question-Answering (QA) systems.

II. Related Work and Our Contribution

There have been various efforts towards building interactive Cognitive systems with main focus around processing ticket data[2]. Chen et al. [6] have worked on trouble tickets to perform root cause analysis on tickets using topic modelling, clustering and IR techniques. They do not focus on finding exact problem and solution from a given ticket. Netsieve[10] attempts to extract frequent n-grams and filter them to get problem and solution patterns. Next, it uses these patterns to identify a multitude of symptom snippets from the ticket text. The challenge with such approach is that it doesn't pin point a specific problem reported by the customer. Our system caters to this specific issue by giving a ranked problem-solution pairs from a customer ticket. Netsieve, furthermore, does not make use of vectorised representations of words by automatically learning word embeddings [9]. A key feature of such word-embeddings is to have similar vector representations of similar words. In noisy free-form text, humans tend to write a problem in many ways as long as it means the same. There is a need for a cognitive agent to have a semantic understanding of the sentence. Word embeddings and dependency parsing [3] immediately bridge this gap and allow us to come up with a more general and semantic attributes of problem-like sentences and solution-like sentences.

Further, Meystre et al. have shown efforts made in problem extraction from documents in medical domain using natural language processing techniques [8]. They focus on extracting problem list for a patient from medical documents which are generally structured and less noisy than hardware tickets. Ajmera et al.'s work [1] attempts to find out short segments of problem text from a verbose description of a problem in case logs. These verbose descriptions are short customer complaints and, unlike a hardware ticket, do not contain a disorganised mix of machine text, misleading diagnosis and agent chats. Hen et al. [5] have worked towards extraction of frequently asked questions by combining data from software development discussion forums which mainly comprise of well-organised problems and replies. This is in stark contrast with the hardware tickets which are noisy and disorganised.

III. Dataset

Tickets contain a lot of machine generated content and agent written text including some structured snippets. Some of the

Type	Source	Example
Structured	Agent-written	*type/model: 1234-abc...*
	Agent-written	*serial : 654321 ...*
	Agent-written	*firmware: 87.6.5 ldots*
	Machine Generated	*+john, doe(acc)*
		-c/mxz/c–l**h/— ...*
Unstructured	Agent chats	*thanks! still waiting for the log data for review. ...*
		it is good idea for prepare standby parts for our customer and i had ...

TABLE I: Irrelevant structured and unstructured parts of ticket text

Type	Source	Example
Voice Call Transcript	Problem	spoke to Michael he said that the HMC is down he was doing an upgrade and the HMC went down its a possible drive failure he said its a 500 g b drive, client tried to reseat dirve,rebooted,nogo wants a CE onsite for drive replacement, in LPD no error.
Machine Generated	Problem	09/04 0804 SYS: AIX disk drive failure * *see tib*
Agent-Written	Solution	replace the drive, varified vary on status. verified tape initialized with nancy

TABLE II: Relevant Parts of Ticket Text

major structured fields include the text field, the scratch-pad field, customer name, severity, category, creation date, last modified and many more. In this paper, we will use the phrase *ticket text* to refer to the text field of the ticket. The text field contains the majority of diagnostic information and is largely unstructured. Table 1 represents a sample of irrelevant sentences found in the ticket. In the table we show information related to firmware, serial number, type and model as irrelevant because they do not play a role in identification of problem or solution within the ticket. Some sample relevant sentences are shown in Table 2.

IV. Design and Implementation

In this section, we first give a bird-eye view of our system architecture. Subsequently, we integrate various modules and describe in detail the functioning of the system.

A. Neev: High-level View of the System

Our system consists of 3 modules 1) Synonym Ingestion Module, 2) Noise Reducer Module, 3) QA Extractor Module. These are shown in Figure 1 and broadly described in this section. The effect of each module is depicted in a Venn diagram in Figure 2. The universal set shown corresponds to the ticket text. The next subset corresponds to the de-noised text obtained from the Noise Reducer Module. The partition line divides the relevant text into problem and solution using the high-recall QA extractor. The little circles (shown in blue)

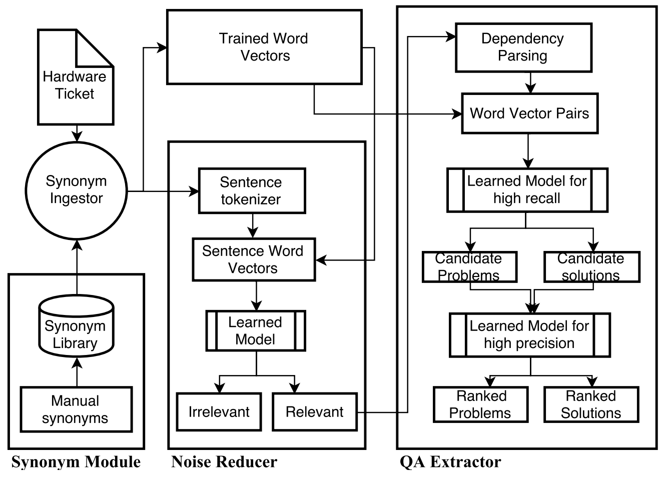

Fig. 1: Neev : Architecture Diagram.

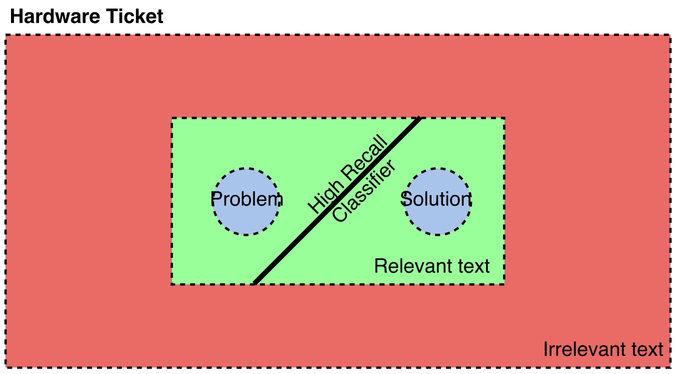

Fig. 2: Zeroing in towards Problem and Solution.

correspond to the precise problem and the solution embedded in the ticket text obtained using high-precision QA Extractor.

1) Synonym Ingestion Module

The main objective of this module is to incorporate domain knowledge to relate the words which are synonymous. In sentence classification, if synonymous words have different vector representations, then the classification performance might degrade. To avoid this, we need a module to identify the synonymous words and phrases and replace all of them by a same worded-representation. This will assign all synonyms with the same $n-$dimensional euclidean vector and assist the word-embedding learning process so it does not need to automatically learn such similarities.

2) Noise Reducer Module

The main objective of noise reducer module is to get a cleaner data. This step accomplishes the classification of sentences into the following two classes

- **Relevant:** By relevant we mean sentences that are either a part of the problem diagnosis trace or the solution text.
- **Irrelevant:** Remaining sentences are marked as irrelevant since they are unrelated to either problem or solution. These commonly include machine text, greetings and formalities, follow-up reminders and the like.

Since this data set contains a large portion of irrelevant text, a large ticket data set acts as a surrogate for the irrelevant text required for supervised learning. We will refer to this corpus as as the *ticket corpus*.
We use a large data set comprising of the troubleshooting guides which contain descriptions of a number of known error codes along with their problem and solution text. This data set, in a way, captures the essential textual features and the tone of problem and solution texts that we seek. Consequently, this data set is used as a surrogate for relevant text required for supervised learning. We will refer to this corpus as the *troubleshooting corpus*. Using these two corpora for relevant and irrelevant text, our training strategy consists of two parts. First, we learn euclidean vectors for every word found in the two corpora using the word2vec technique. This tool maps a large vocabulary onto a euclidean space of much smaller dimension such that semantically close words are also close in the new space. Second, we cluster the words occurring in both ticket and troubleshooting corpora in the new vector space using the k-means clustering algorithm [4]. Next, we employ a classification mechanism which classifies every sentence in a ticket by checking for the closeness of each sentence with the ticket corpus and the troubleshooting corpus. We assign it a label depending on whichever class is closer or more similar.

3) QA Extractor Module

The main objective of QA extractor module is to pin-point the problem and solution sentences from relevant ticket text obtained from the noise-reducer module. The occurrence of problems and solutions in ticket text do not follow any specific pattern or manual rule so any rule-based estimate does not take us consistently to the problem or solution of the ticket for a satisfactory fraction of tickets.

As mentioned earlier, the troubleshooting corpus which we use for training contains descriptions of error codes along with their problem text and solution text. This provides an ideal sample of problem and solution text required for supervised learning to discriminate problem from solution. Our training consists of 1) Re-using the word vectors computed for every word in the corpora using word2vec[9] tool, 2) Using Stanford dependency parser [3] to get dependency parse trees for the sentences in the problem and solution corpora, 3) Constructing word-pairs using every edge/relation in the dependency tree, 4) Constructing vectors from word-pairs extracted using the word vectors obtained using word2vec tool, 5) Clustering vectors from the word pairs occurring in problem text corpus as well as the solution text corpus and 6) creating an approximate probability distribution of word-pair vectors within the problem and the solution classes. Next, we have a classification mechanism which classifies every sentence in an incoming hardware ticket by checking for the closeness of each sentence of test ticket with the problem and the solution classes.
To increase the precision for problem and solution extraction, we build a next-level classifier which uses frequent word-word

Synset
disk partition, *volume*, file system
exp, disk expansion
media_error, medium error, sector error
hard drive, hard disk, *HDD*, disk drive
memory, RAM, DIMM
computer, server, system
BBU, battery backup unit, *battery*
utility power, alternating power, *AC*

TABLE III: A Sample Synonym List

Algorithm 1 Noise Reducer Module: Training Phase

```
FUNCTION:
INPUT:
    $S_{relevant}$ the set of relevant sentences
    $T(w)$, look-up table for vector representation of any word $w$
OUTPUT:
$C_{relevant}$, the set of relevant cluster-centroids

for $s \in S_{relevant}$ do
  for $w_i \in s$ do
    Look up $\mathbf{v_i} \leftarrow T(w_i)$
    Add to the vector set $V_{relevant} \leftarrow \mathbf{v_i}$
  end for
end for
RETURN $C_{relevant} \leftarrow kMeans(V_{relevant}, K)$
    where $K$ is a design parameter
```

dependencies (or edges) in the problem and solution text. For this high-precision classifier, the training data comes from a small set of 900 tickets which are agent-annotated with the problem and solution snippets.

B. Neev : Detailed Working of the System

This section describes a detailed step-by-step workflow of our system.

1) Synonym Ingestion Module

In the ingestion phase, we perform case-insensitive search to find any phrase that is available in the synonym list. Next, we replace the matching phrase with the head phrase of the corresponding synset.

2) Noise Reducer Module

This module performs training to returns the likelihood of a word belonging to the relevant class. Algorithm 1 describes how we use k-means to obtain clusters of vectors belonging to relevant and irrelevant texts, respectively. We place a spherical Gaussian centered on each of the cluster centroids to obtain an approximate probability distribution for each class. Equations 1 and 2 describe how we construct an approximate probability distributions for the relevant and the irrelevant classes, respectively.

$$P_{approx}(\mathbf{v}|Relevant) = \frac{1}{K} \sum_{\mathbf{c_i} \in C_{relevant}} exp(-c||\mathbf{v} - \mathbf{c_i}||^2) \tag{1}$$

$$P_{approx}(\mathbf{v}|Irrelevant) = \frac{1}{K} \sum_{\mathbf{c_i} \in C_{irrelevant}} exp(-c||\mathbf{v} - \mathbf{c_i}||^2) \tag{2}$$

where $\mathbf{v}$ refers to any word vector in some sentence in the test ticket at run time. $C_{relevant}$ refers to the set of k-means cluster centroids obtained after clustering step performed on the relevant text. $C_{irrelevant}$ is the irrelevant counter-part of the same notation. K is the size of the sets of cluster centroids in the respective equations. Using the probability for a word in a sentence, we calculate the probability of a sentence belonging to relevant and irrelevant text by adding up the log-likelihoods for the respective classes for every word in the sentence.

$$log(P(s|Relevant)) = \sum_{\mathbf{w} \in s} log(P_{approx}(\mathbf{w}|Relevant)) \tag{3}$$

$$log(P(s|Irrelevant)) = \sum_{\mathbf{w} \in s} log(P_{approx}(\mathbf{w}|Irrelevant)) \tag{4}$$

Here, $\mathbf{w}$ refers to some word in some sentence s. This approach is successful in removing the noise such as machine generated text, logs from numerous support portals and other non-useful agent conversations. We are left with high recall of relevant text by removing 75% of the ticket text.

3) QA Extraction Module - High Recall

In this module, we train a probability distribution for the problem and the solution classes. We assume the presence

Algorithm 2 QA Extraction - High Recall Classifier : Training Phase

```
INPUT:
    $S_p$, set of problem-like sentences
    $T(w)$, look-up table for vector representation of any word $w$
OUTPUT:
    $C_{problem}$, the set of cluster-centroids of problem edges

for each sentence $s \in S_p$ do
  $d = dependencyParse(s)$
  for each edge $(w_i, w_j) \in d$ do
    $\mathbf{v_i} = T(w_i)$
    $\mathbf{v_j} = T(w_j)$
    $\mathbf{v_{i,j}} = (\mathbf{v_i}, \mathbf{v_j})$
    Add to the vector set $V_{problem} \leftarrow \mathbf{v_{i,j}}$
  end for
end for
RETURN $C_{problem} \leftarrow kMeans(V_{problem}, K)$
    $K$ is the design parameter
```

of a dependency parser to obtain a dependency tree depicting

Algorithm 3 Generating Bag of Edge-Vectors for a Sentence

```
INPUT:
    $T(w)$, look-up table for vector representation of any word $w$
    $s$, a sentence in the de-noised ticket
OUTPUT:
    $V_{parse}$, a bag of edge-vectors in the sentence

$d = dependencyParse(s)$
for each edge $(w_i, w_j) \in d$ do
    $\mathbf{v_i} = T(w_i)$
    $\mathbf{v_j} = T(w_j)$
    $\mathbf{v_{edge}} = (\mathbf{v_i}, \mathbf{v_j})$
    Add to the vector set $V_{parse} \leftarrow \mathbf{v_{edge}}$
end for
RETURN $V_{parse}$
```

relationship between dependent words in a sentence. For any sentence s, let the dependency parser $dependencyParse(s)$ return a tree represented as (V_s, E_s) where V_s is the set of words/nodes in the sentence and E_s is the set of edges $\{(w_i, w_j, relation) | \forall w_i, w_j \in s\}$ representing relations between dependent words. The dependency tree starts at a dummy root node $ROOT$. In our case, we use the Stanford dependency parser [3].

We consider each sentence as a bag of edges where the directed edges are obtained from the dependency-tree. We obtain a vector for every edge by concatenating the word vectors of the two joining words. Thus, our sentence is represented by a bag of edge-vectors. Thus, in the training step, we try to obtain approximate probability distribution of edge-vectors within the two classes namely, the problem class and the solution class.

We first perform k-means clustering separately on the edge-vectors belonging to the problem class and the solution class. This pipeline is depicted in Algorithm 2 for the problem class. Next, we place a spherical Gaussian centered on each of the cluster centroids to obtain an approximate probability distribution for each class. We introduce a design parameter to determine the radius of these Gaussian. Also, since we use k-means for clustering, hence k is also a design parameter. k determines the number of clusters desired in the model. These distributions can be shown as follows

$$P_{approx}(\mathbf{v}|Problem) = \frac{1}{K} \sum_{\mathbf{c_i} \in C_{problem}} exp(-c||\mathbf{v} - \mathbf{c_i}||^2) \tag{5}$$

$$P_{approx}(\mathbf{v}|Solution) = \frac{1}{K} \sum_{\mathbf{c_i} \in C_{solution}} exp(-c||\mathbf{v} - \mathbf{c_i}||^2) \tag{6}$$

where $\mathbf{v}$ is the edge-vector, $C_{problem}$ is the set of edge-vector cluster centroids for the problem class, $C_{solution}$ is the set of edge-vector cluster centroids for the solution class, c is the design parameter and K is the number of clusters obtained in the clustering.

We can use these distributions to infer the class for a given sentence. To achieve this, we obtain the bag-of-edge-vectors for the sentence using the dependency parser as shown in Algorithm 3. We sum the log-likelihoods of the vectors in the bag to assign a belongingness score for the two classes. The algorithm for inferring the class for a sentence in a ticket is given in Algorithm 3. Equation 7 gives the log-likelihood of the sentence s being a problem. Equation 8 gives the log-likelihood of the sentence s being a solution.

$$\sum_{\mathbf{v}_{edge} \in dependencyParse(s)} log(P_{approx}(\mathbf{v}_{edge}|Problem)) \tag{7}$$

$$\sum_{\mathbf{v}_{edge} \in dependencyParse(s)} log(P_{approx}(\mathbf{v}_{edge}|Solution)) \tag{8}$$

We assign the sentence s to the class with the higher score. Number of vectors in the training data approximately scales with the number of words in the problem and solution corpora. Hence, a simple look-up table to classify a sentence into one of two classes would be extremely memory inefficient and time-consuming. Thus, our model needs to be compressed while still retaining important features. To this end, clustering drastically reduces the model size and approximates the entire distribution of training data (for each class) using a small number of clusters. This, furthermore, brings down the time to compute the log-likelihood values.

4) QA Extraction Module - High Precision

This module implements a next level classifier to the earlier QA High Recall Module. The main motivation behind this is to get a narrower set of problems and solutions by increasing precision. We train on a small, 700 strong, agent-annotated QA set.

In the training phase, we perform dependency parsing on every problem in the training set. Further, we extract every word to word dependency (or edge) available in those dependency trees. We keep track of the most frequent word to word dependencies (or edges) in a frequency map $EdgeMap_{problem}$. In a way, the more frequent edges are deemed more important than less frequent ones in their ability to identify true problem or solution sentence. A separate frequency map $EdgeMap_{solution}$ is obtained for the solution text using the same approach. In Table 3, we show a few most frequent edges for the two cases of problem and solution, respectively. The algorithm for this approach is given in Algorithm 4.

For inferring the problem sentences from the ticket text, we first perform the dependency parsing on the ticket sentences flagged as problem by the high-recall classifier in the previous module. Next, we simply select the sentences which contain the dependency edges available in $EdgeMap_{problem}$. We rank the selected sentences on the basis of the frequency of corresponding edge in the frequency map $EdgeMap_{problem}$. The algorithm for the approach is presented in Algorithm 5. For the case of solution extraction, we perform the same approach using $EdgeMap_{solution}$.

Advantages of using dependency edges over n-grams is that

Most-frequent Problem Edges	Most-frequent Solution Edges
(drive,tape)	(dispatch,ce)
(need,replac)	(plan,action)
(releas,call)	(updat,firmwar)
(drive,hard)	(pleas,replac)
(call,server)	(need,replac)
(run,format)	(plan,dispatch)
(drive,disk)	(dispatch,call)
(fail,batteri)	(replac,drive)
(replac,to)	(replac,dimm)
(handl,for)	(check,updat)
(handl,replac)	(call,ce)
(batteri,fru)	(updat,to)
(releas,handl)	(replac,disk)
(handl,batteri)	(support,for)
(fail,drive)	(dispatch,case)
(replac,batteri)	(check,and)
(call,hmc)	(replac,drv)

TABLE IV: Most Frequent Edges for problem and solution texts.

the dependency edges are robust against varying phrases that mean the same thing. As an example, the phrases "need to replace drive" and "drive replacement needed" mean the same thing but would be captured as different n-grams. On the contrary, dependency edge (after stemming) *(replac,driv)* would occur in both phrases.

Algorithm 4 QA Extraction - High Precision Classifier : Training Phase

```
INPUT:
    $C_p$ = Problem corpus
    $\theta_p$ = frequency threshold
OUTPUT:
    $EdgeMap_{problem}(edge \to count)$ = Frequency map of problem edges
Map $EdgeMap_{problem}(edge \to count)$,
for each sentence $s \in C_p$ do
  $d = dependencyParse(s)$
  for for each edge $(w_1, w_2) \in d$ do
    Create edge from $[Stem(n_1), Stem(n_2)]$ where $Stem()$ stems the word to its morph form
    if $edge \notin EdgeMap_{problem}.Keys$ then
      $EdgeMap_{problem}(edge) \leftarrow 1$
    else
      $EdgeMap_{problem}(edge) + +$
    end if
  end for
  for each edge $\in EdgeMap_{problem}.Keys$ do
    if $EdgeMap_{problem}.get(edge) < \theta_p$ then
      $EdgeMap_{problem}.Remove(edge)$
    end if
  end for
end for
RETURN $EdgeMap_{problem}$
```

Algorithm 5 QA Extraction - High Precision Classifier : Inference

```
INPUT:
    $List_p$, list of high-recall problem candidates from ticket $\tau$
    $EdgeMap_{problem}$ trained frequency map of problem edges
OUTPUT:
    $S_p$, precise ranked candidate-list for problems

Map $\tau_{pmap}(edge \to count)$
for each sentence $s \in List_p$ do
  $d = dependencyParse(s)$
  for each edge $[w_1, w_2] \in d$ do
    if $edge \in EdgeMap_{problem}.Keys$ then
      $\tau_{pmap} \leftarrow ([w_1, w_2] \to EdgeMap_{problem}([w_1, w_2]))$
    end if
  end for
end for
$S_p \leftarrow$ most frequent keys in $\tau_{pmap}$ in decreasing order
RETURN $S_p$
```

REP OPENED NEW H/W CASE FOR THE ISSUE.
++++++++++++++++++++++++++++
H/W TICKET: XXXXX,XX,XXX
++++++++++++++++++++++++++++
H/W TEAM WILL BE WORKING ON H/W TICKET.
————————————————
CALLED JOHN DOE @ XXX-XXX-XXXX.
INFORMED CLIENT ABOUT THE QUERY.
-WAN, DOE -D/TXXXXXXX–LXXX/——-X3X3-dd/mm/yy-15:36–RT
——————
-WAN, DOE -D/TXXXXXXX–LXXX/——-X3X3-dd/mm/yy-15:36–RT
-WAN, DOE -D/TXXXXXXX–LXXX/——-X3X3-dd/mm/yy-15:36–RT
IT IS GOOD IDEA FOR PREPARE STANDBY PARTS FOR OUR CUSTOMER. AND I HAD ALREADY HAD MEETING WITH CUSTOMER THIS MONDAY. AND CUSTOMER ALSO HOPE COMPANY CAN PREPARE STANDBY PARTS
CAN YOU GIVE ME GUIDE HOW TO REQUEST STANDBY PARTS FOR THIS CASE. AND CUSTOMER PUSHED ME TO GIVE THEM THE ROOT CAUSE OF HIGH FAILURE RATE OF XXXX IO BACKPLANE AND ALSO NEED COMPANY TO GIVE THEM SOLUTIONS ——————
CLIENT AGREED TO THE SAME.
IS THAT ANY DEFECT OF 780 I/O BACKPLANE? IS THAT ANY ECA FOR IT?
WE STORED ONE 780 I/O BACKPLANE IN COMPANY INTERNAL IT CENTER
SINCE LAST TIME I/O BACKPLANE FAILED, COULD YOU PLEASE HELP TO ARRANGE ONE MORE
NEED YOUR KINDLY SUPPORT, THANKS! ONE FIRMWARE ENHANCEMENT HAS BEEN RELEASED IN XXX_XXX IT COULD
PROVIDE BENEFIT FOR POWERING ON FOR THIS ISSUE.
————————

Fig. 3: Raw Ticket Text

rep opened new h/w case for the issue. h/w team will be working on h/w ticket.
it is good idea for prepare standby parts for our customer and i had already had meeting with customer this monday and customer also hope
can you give me guide how to request standby parts for this case and customer pushed me to give them the root cause of high failure rate of
9179-mhb io backplane, and also need ibm give them the solutions. Is that any defect of 780 I/O backplane? Is that any ECA for it?
We stored one 780 I/O backplane in xxxx internal IT center since last time I/O backplane failed,
could you help to arrange one more in xxxxx?

Fig. 4: Denoised Ticket Text

V. A Walk-through Example

In this section, we show how our system works on a new incoming ticket. For representation, we take a sample obfuscated ticket from our ticket corpus and show how the pieces fit together. We take the manual synonym list given by Subject Matter Experts (SME) and perform the ingestion. After synonym ingestion, we filter out irrelevant text from the ticket. The denoised ticket is represented in Figure 4. We further segregate sentences in Figure 4 into problem-portion

RANK 1: WILL CHECK WITH THE xx TEAM TO GET THE XX PROGRESS

RANK 2: *IS THAT ANY DEFECT OF 780 I/O BACKPLANE? IS THAT ANY ECA FOR IT? WE STORED ONE 780 I/O BACKPLANE IN COMPANY INTERNAL IT CENTER SINCE LAST TIME I/O BACKPLANE FAILED*, COULD YOU PLEASE HELP TO ARRANGE ONE MORE NEED YOUR KINDLY SUPPORT, THANKS!

RANK 3: ALSO PLEASE UPDATE THE IMPACT IN XXX.

RANK 4: PLEASE RECORD THE XX AND XX OF REPLACED I/O PLANAR TO THIS TICKET THANKS!

Fig. 5: Problem Candidates for high recall classifier

RANK 1: WHAT CAUSED THE BOX TO CRASH? WHICH PART NEED TO BE REPLACED. PLEASE HELP TO ANALYZE THE LOG AND GIVE US ACTION PLAN, CUSTOMER HOPE WE FIX THE PROBLEM ASAP. THANKS FOR YOUR SUPPORT.

RANK 2: AGENT COLLECTED LOG AND WAIT CUSTOMER TO SEND IT TO ME, SO WE WILL UPLOAD THE FILE ASAP.

RANK 3: IT IS GOOD IDEA FOR *PREPARE STANDBY PARTS FOR OUR CUSTOMER. AND I HAD ALREADY HAD MEETING WITH CUSTOMER THIS MONDAY. AND CUSTOMER ALSO HOPE COMPANY CAN PREPARE STANDBY PARTS CAN YOU GIVE ME GUIDE HOW TO REQUEST STANDBY PARTS FOR THIS CASE. AND CUSTOMER PUSHED ME TO GIVE THEM THE ROOT CAUSE OF HIGH FAILURE RATE OF XXXX IO BACKPLANE* AND ALSO NEED COMPANY TO GIVE THEM SOLUTIONS.

RANK 4: PROJECT OFFICE WILL EVALUATE CUSTOMER SITUATION TO DECIDE IF STANDBY PARTS ARE NEEDED PLEASE WORK WITH XXX VIA XXX TO APPLY IF YOU REALLY NEED IT THE CAUSE OF OUTAGE WAS DUE TO DEFECTIVE I/O PLANAR BUT BEFORE FAILURE ANALYSIS IS COMPLETED WE CANNOT TELL WHICH COMPONENT ON PLANAR FAILED.

RANK 5: REVIEW THE LOG DATA, CODE XXXXXX POST AND SYSTEM WAS TERMINATED. THIS PROGRAM HIT THE GATEWAY XXXX- SYSTEM TERMINATED WITH CODE XXXX. PLEASE REPLACE THE DEFECT I/O BACKPLANE BY FOLLOWING THE ROUTE.

Fig. 6: Solution Candidates for high recall classifier

RANK 1: *IS THAT ANY DEFECT OF 780 I/O BACKPLANE? IS THAT ANY ECA FOR IT? WE STORED ONE 780 I/O BACKPLANE IN COMPANY INTERNAL IT CENTER SINCE LAST TIME I/O BACKPLANE FAILED*, COULD YOU PLEASE HELP TO ARRANGE ONE MORE NEED YOUR KINDLY SUPPORT, THANKS!

RANK 2: WILL CHECK WITH THE xx TEAM TO GET THE XX PROGRESS

Fig. 7: Problem Candidates for high precision classifier

RANK 1: IT IS GOOD IDEA FOR *PREPARE STANDBY PARTS FOR OUR CUSTOMER. AND I HAD ALREADY HAD MEETING WITH CUSTOMER THIS MONDAY. AND CUSTOMER ALSO HOPE COMPANY CAN PREPARE STANDBY PARTS CAN YOU GIVE ME GUIDE HOW TO REQUEST STANDBY PARTS FOR THIS CASE. AND CUSTOMER PUSHED ME TO GIVE THEM THE ROOT CAUSE OF HIGH FAILURE RATE OF XXXX IO BACKPLANE* AND ALSO NEED COMPANY TO GIVE THEM SOLUTIONS.

RANK 2: WHAT CAUSED THE BOX TO CRASH? WHICH PART NEED TO BE REPLACED. PLEASE HELP TO ANALYZE THE LOG AND GIVE US ACTION PLAN, CUSTOMER HOPE WE FIX THE PROBLEM ASAP. THANKS FOR YOUR SUPPORT.

Fig. 8: Solution Candidates for high precision classifier

and solution-portion in Figures 5 and 6, respectively, using high-recall QA extractor. To have a narrower set of problems and solutions, we use high-precision QA extractor and our ticket gets reduced as shown in Figure 7 and 8. This makes it easy for an agent to look through it and reach and validate the problem and solution quickly. Top ranking results are taken from the output of the high precision classifier and remaining are discarded.

VI. Evaluation and Results

A. Data Set

Our data set consists of 45,000 un-annotated hardware tickets retrieved over one month. Each ticket contains an average of 245 lines and an average of 2000 words in the unstructured portion. Moreover, we take 51,000 troubleshooting documents, where each document attempts to resolve a known error code. Such document give a verbal description of the problem and then provide a verbal description of how one might resolve the problem. We combine these two data sets to train the word-vectors using word2vec [9] technique. This results in a trained vocabulary which consists of 1,50,000 word-vectors. These large data sets enable us to learn the unsupervised features from our corpora.
Apart from these, we have 900 hardware tickets which have been annotated by the agents. Such tickets have tags marking the lines which contain the problem and the solution. We randomly use 700 such tickets for training and use the remaining 200 for testing and evaluation. Each ticket has a severity value associated with it. It tends to reflect how urgent the need for resolution is and irreparable the issue is.

B. Efficacy of Noise-Reduction Module

This module partitions the ticket text into relevant and irrelevant portions which lets a customer service agent read less data and get more informative results. As an experiment, we give 100 tickets to our system which returns 100 documents containing only the relevant text. These documents are then given to the SME's to evaluate the increase in efficiency. We compare the reading times per ticket before and after the noise-reduction. We observe a reduction in reading time by 70%. Moreover, the number of lines get reduced by 77% leaving only the relevant lines.

C. Efficacy of QA Extraction Module

1) Evaluation Metric

For validation of our ideas and system we experiment the approach using 200 tickets. These tickets are agent-annotated with problem and solution tags/snippets. Our system returns a ranked list of high-confidence problems and solutions. Thus, we evaluate our system's performance using a rank-based metric. In this metric, we compute accuracy by noting if the correct problem occurs in the top-n ranked results. We vary n over 1 through 5 for problems and 1 through 8 for solutions and report the accuracies in the Figure 9 and 10. For every test ticket, the results returned by our system are examined by SME's who manually mark the position (from the top) at which they found a correct problem and solution. These are used to compute the results shown in Table 2. Furthermore, using the same metric, we attempt to analyse the variation in accuracy with ticket severity. 60 tickets were identified with high severity 3 and 40 were identified with low severity 1 and 2. Other tickets had no severity assigned to them. The accuracy results for the two severity classes are presented in Figure 11 and Figure 12 for problem and solution extraction, respectively.
In Figure 9, we observe that the accuracy for problem extraction for top-1 result is 62% and it reaches 86% and 94% for top-3 and top-5 cases, respectively. In Figure 10, we observe that the accuracy for solution extraction is initially low but reaches 62% and 88% for top-3 and top-8 cases, respectively. It is noteworthy to see that problem extraction is more precise as compared to solution extraction. This is because problems are generally crisp and short. On the contrary, solutions are long and verbose. Agents might try out numerous approaches before arriving at a correct resolution. Moreover, a lot of ticket could be merely a follow-up monitoring of the progress after a

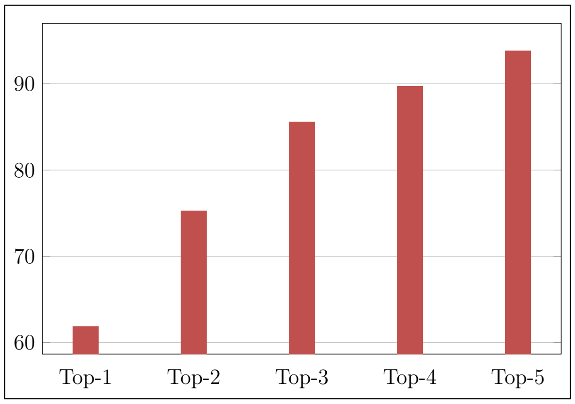

Fig. 9: Rank-based accuracy of our system for problem extraction

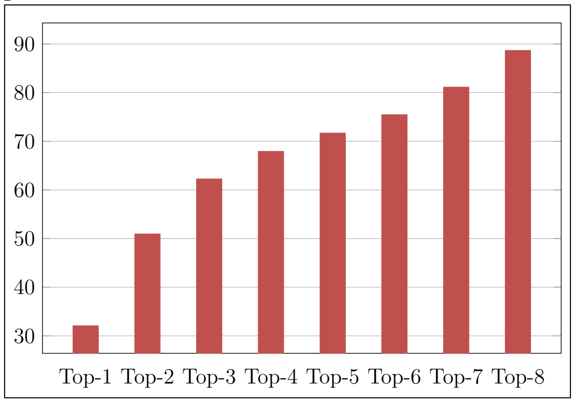

Fig. 10: Rank-based accuracy of our system for solution extraction

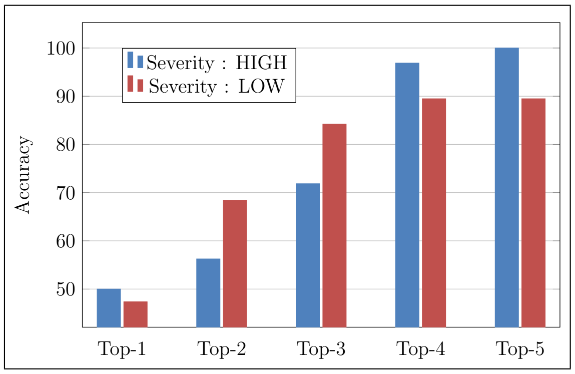

Fig. 11: Severity-wise accuracy of our system for problem extraction

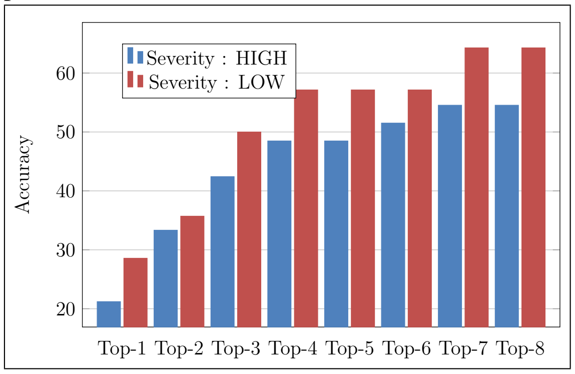

Fig. 12: Severity-wise accuracy of our system for solution extraction

resolution has been reached. In Figure 11, we observe that low-severity tickets offer greater accuracy of problem extraction within top-3 results in comparison to high-severity tickets. However, in top-5 results, both appear to perform at par. In Figure 12, we observe that low-severity tickets offer greater accuracy of solution extraction across the board.

VII. Conclusion and Future Work

This paper describes a cognitive agent to mine problems and solutions from hardware tickets. We overcome the challenge posed by our noisy hardware tickets by capturing the semantics of relevant text through dependencies among words and learning word embeddings and segregate the relevant and irrelevant parts of ticket. As future work, we plan to have an interactive QA service for customer service agents which helps them analyse different tickets and further give them a way to get frequent problems and solutions in a particular ticket corpus. Furthermore, the system can be used as a plugin and can be integrated with a curator to generate curated content. Such curated content finds use in training state-of-the-art QA Systems.

References

[1] Ajmera, Jitendra, et al. ”Automatic generation of question answer pairs from noisy case logs.” 2014 IEEE 30th International Conference on Data Engineering. IEEE, 2014.

[2] Chung, Hoojung, et al. ”A practical QA system in restricted domains.” Proceedings of the Workshop Question Answering in Restricted Domains, within ACL. 2004.

[3] De Marneffe, M. C., & Manning, C. D. (2008, August). The Stanford typed dependencies representation.In Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation (pp. 1-8). Association for Computational Linguistics.

[4] Hartigan, John A., and Manchek A. Wong. ”Algorithm AS 136: A k-means clustering algorithm.” Journal of the Royal Statistical Society. Series C (Applied Statistics) 28.1 (1979): 100-108.

[5] Hen, Stefan, Martin Monperrus, and Mira Mezini. ”Semi-automatically extracting FAQs to improve accessibility of software development knowledge.” Proceedings of the 34th International Conference on Software Engineering. IEEE Press, 2012.

[6] Jan, Ea-Ee, Kuan-Yu Chen, and Tsuyoshi Id. ”Probabilistic text analytics framework for information technology service desk tickets.” Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on. IEEE, 2015.

[7] Justeson, John S., and Slava M. Katz. ”Technical terminology: some linguistic properties and an algorithm for identification in text.” Natural language engineering 1.01 (1995): 9-27.

[8] Meystre, Stphane, and Peter J. Haug. ”Natural language processing to extract medical problems from electronic clinical documents: performance evaluation.” Journal of biomedical informatics 39.6 (2006): 589-599.

[9] Mikolov, Tomas, et al. ”Efficient estimation of word representations in vector space.” arXiv preprint arXiv:1301.3781 (2013).

[10] Potharaju, Rahul, Navendu Jain, and Cristina Nita-Rotaru. ”Juggling the jigsaw: Towards automated problem inference from network trouble tickets.” Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13). 2013.

[11] Shao, Qihong, et al. ”Efficient ticket routing by resolution sequence mining.” Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2008.