Hi, How Can I Help You? Automating Enterprise IT Support Help Desks

Senthil Mani,[1] **Neelamadhav Gantayat,**[1] **Rahul Aralikatte,**[1] **Monika Gupta**[1]
{sentmani, neelamadhav, rahul.a.r, monikgup}@in.ibm.com

Sampath Dechu,[1] **Anush Sankaran,**[1] **Shreya Khare**[1]
{sampath.dechu, anussank, shkahre4}@in.ibm.com

Barry Mitchell,[2] **Hemamalini Subramanian,**[2] **Hema Venkatarangan**[2]
bcm@us.ibm.com, {hesubram, hema.sudarshan}@in.ibm.com
[1]IBM Reseach AI [2]IBM Global Business Services

Abstract

Question answering is one of the primary challenges of natural language understanding. In realizing such a system, providing complex long answers to questions is a challenging task as opposed to factoid answering as the former needs context disambiguation. The different methods explored in the literature can be broadly classified into three categories namely: 1) classification based, 2) knowledge graph based and 3) retrieval based. Individually, none of them address the need of an enterprise wide assistance system for an IT support and maintenance domain. In this domain, the variance of answers is large ranging from factoid to structured operating procedures; the knowledge is present across heterogeneous data sources like application specific documentation, ticket management systems and any single technique for a general purpose assistance is unable to scale for such a landscape. To address this, we have built a cognitive platform with capabilities adopted for this domain. Further, we have built a general purpose question answering system leveraging the platform that can be instantiated for multiple products, technologies in the support domain. The system uses a novel hybrid answering model that orchestrates across a deep learning classifier, a knowledge graph based context disambiguation module and a sophisticated bag-of-words search system. This orchestration performs context switching for a provided question and also does a smooth hand-off of the question to a human expert if none of the automated techniques can provide a confident answer. This system has been deployed across 675 internal enterprise IT support and maintenance projects.

1 Introduction

One of the long-term goals of AI is to have a system answering complex questions asked in natural language. The term question answering (QA) has a very broad interpretation enabling almost any problem to be formulated as one. The problem of QA originated decades ago with research focused on retrieving answers from structured databases. AI has aided the growth of QA systems in more challenging settings with better natural data understanding. QA systems have undergone a revolution in inferring from various information sources such as unstructured web content, large-scale document corpus and provide more cognitive responses (Fang et al. 2016).

Copyright © 2018, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Based on the response provided, QA systems can be broadly classified into two groups: i) factoid based systems, which generally answer ‘wh’ questions. The answer is typically an entity with one or few words (Xu et al. 2015) (Oh et al. 2016), ii) non-factoid based systems providing complex long answers or a procedural explanation to the question.

Research in factoid QA systems is very mature (Kolomiyets and Moens 2011) with lots of open domain systems available such as AskHermes (Cao et al. 2011), AnswerBus (Zheng 2002), QANUS (Ng and Kan 2015), YodaQA (Baudiš and Šedivỳ 2015), Sirius (Hauswald et al. 2015), and DEQA (Lehmann et al. 2012). Publicly available large-scale knowledge bases, such as *FreeBase*, are manually curated to learn and benchmark factoid based QA systems (Yao and Van Durme 2014) (Bordes et al. 2015).

However, going beyond factoid QA systems, i.e. retrieving long answers is extremely challenging both in defining what constitutes an answer chunk and in evaluating the retrieval performance (Yang et al. 2016). In such non-factoid systems, generating syntactically correct long procedural answers is challenging, compelling the use of classification or search based approaches for such tasks (Clark et al. 2016) (Mitra and Baral 2015).

In enterprise IT, one of the main sources of revenue is providing support to clients. In 2012, the IT support industry revenue in the U.S. alone was estimated to be $212 billion (Shapiro 2014). This is an area where even simple automations can bring about a huge financial impact. (Chen, Chiang, and Storey 2012) studied how intelligent systems can make this sector more profitable. In this work, we present a platform which is developed and deployed inside IBM to share the load of IT support agents.

This system is a classical case of a QA system where the answers are not factoid but are long explanatory answers which often contain multiple steps describing the solution to a problem. The success of such a system in enterprise depends on many factors like: i) accuracy of the answers ii) speed of answer generation/retrieval iii) ability to rapidly ingest new data.

Consider this example from the banking domain: "*How can I get a card?*". It is the responsibility of a system to respond to such incomplete/generic questions. To do this, we need a robust technique which will just not try to answer the question right away but asks for more information to build a clear context. Any single technique will not be able to accomplish this and therefore an ensemble of different techniques are required to understand the question, identify the gaps in the information given by the user and then answer the question in a meaningful manner. To cater to such situations, we have built a platform which can do the following with minimal human assistance.

1. **Knowledge ingestion:** The system has the ability to assimilate knowledge from multiple sources. The sources currently supported are unstructured documents (*.docx, .pptx, .pdf, .html*), images, audio and video recordings.
2. **Dynamic Disambiguation:** If the input question is incomplete and/or extra information is required for answering the question with high confidence, the system asks a probing question to the user rather than providing a generic answer
3. **Orchestration:** There are multiple components in the system which can provide the answer in different situations. Therefore, we have built a separate orchestrator which decides which component to invoke under what circumstances to maximize the likelihood of getting a correct answer
4. **Effective inclusion of human-in-the-loop**: Even with such a system, there are scenarios when a question remains un-answered. In such cases, the orchestration mechanism automatically delegates the question to a human agent who can then carry forward the conversation initiated by the system
5. **Continuous learning:** The system continuously monitors and learns from i) the user's feedback after every dialog turn and ii) the conversations between the user and the system/agent to improve its knowledge store

Further, once the answer has been identified, if there is a back-end fix which needs to be performed to completely resolve the problem, the platform automatically executes an appropriate automaton with the user's permission. For example, if the user was unable to log into a system, the platform, after narrowing down to a high confidence solution, asks the user whether it should reset her password. On receiving an affirmative response, the reset process is performed, hence automating the end-to-end IT technical support process. This platform is currently being used by close to 675 internal enterprise IT support and maintenance projects.

2 System architecture

Our platform consists of three major components: (i) an **knowledge curation** component which converts **raw data** in various formats into structured **knowledge**, (ii) **cognitive** component which learns and indexes the answer for a question through multiple techniques, and (iii) the **assistant** component which interacts with the end user and orchestrates across the cognitive component to fetch the answer. In this section, we discuss these components in detail

2.1 Knowledge Curation

Enterprise knowledge is dispersed across multiple system (incident management systems, document management systems etc.) and formats (html, pdf, ppt, doc etc.). In some cases, the knowledge resides only with the SMEs. To build a question answering system, ingestion of this knowledge is fundamental. The knowledge curation component addresses this heterogeneity by enabling a crowd-sourced platform for curating QA pairs manually, mining incident logs to extract QA pairs, and understanding and extracting content from various structured and unstructured documents.

Incident Analysis The success of a question answering system largely depends on understanding what are the frequently asked questions by the users and being able to address these questions. Past incidents in incident management systems can be mined to understand this insight. However, incidents contain this data (the problem reported and resolution provided) in an unstructured format. Further the quality of this data is human dependant and also prone to human error.

We implemented a clustering approach to mine frequently occurring problem categories by grouping individual incidents to sets of clusters. We then extracted the most frequent problem in the cluster as the question and the corresponding resolution (if available) as the relevant answer. However, in some cases, we had to rely on manual curation of the content to create a relevant QA pair from an identified problem cluster.

Crowd Sourced Curation Much of the knowledge in resolving commonly occurring user issues, is present with the SMEs who have been working in this domain. To bootstrap the system quickly with clean, noise-free QA pairs, we enabled a crowd sourced QA authoring component in our platform. SMEs could use this component to author the answers for the commonly occurring user issues.

Document Ingestion The purpose of document ingestion is to break content into atomic chunks such that each chunk explains about one concept. The document ingestion pipeline handles *.doc*, *.docx*, *.ppt*, *.pptx*, and *.pdf* files in two major steps. In the first step, Apache POI[1] is used to check the formatting of the document i.e. the content is structured with headings and sub-headings. If they are not formatted in this way, we use some structural heuristics and Latent Semantic Indexing (Hofmann 1999) to induce formatting into the documents. The structured document is then converted into HTML using Apache TIKA[2]. Post processing is performed on the HTML chunks to fix any discrepancies or incorrect chunking. The processed chunks are then indexed using Apache Lucene[3] to enable fast retrieval. This pipeline maintains the styles, formatting, tables, images, etc in the

[1]https://poi.apache.org/
[2]https://tika.apache.org/
[3]http://lucene.apache.org/

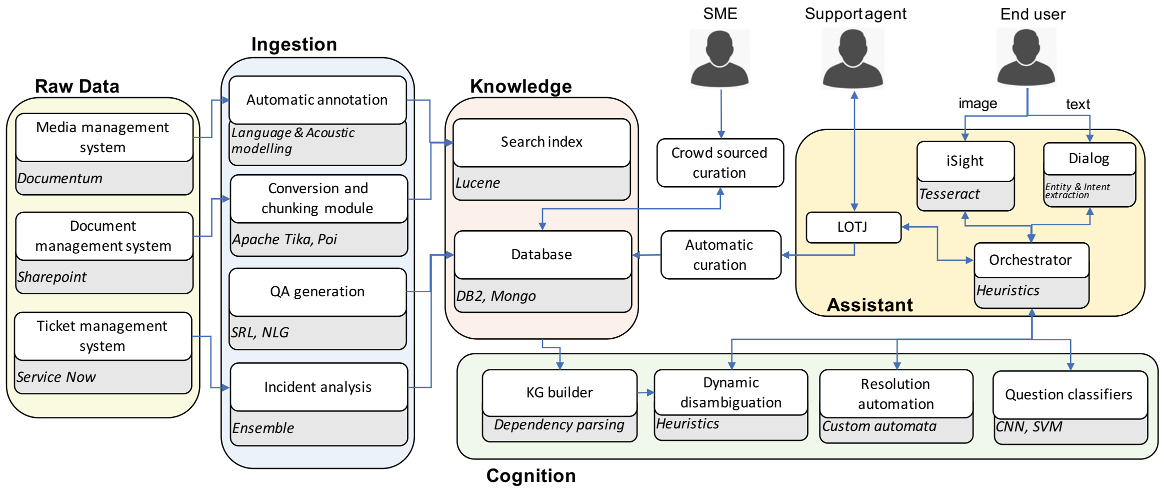


Figure 1: Overall architecture diagram of our platform

documents so that they can be displayed in an aesthetically pleasing manner to the user.

Multi-Media Ingestor In an enterprise, there exist typically knowledge transfer sessions about applications in recorded video formats. These are sessions conducted by an outgoing vendor to the service provider receiving the maintenance contract of the client IT systems. However, knowledge contained in such videos remain inaccessible because it is laborious for the support agent to get the relevant information, as these videos stretch over couple of hours of knowledge transfer sessions. Further, these videos generally show a screen capture of a running application or programs and the related audio contains the relevant correlated information.

Our ingestor component is an ensemble of audio processing techniques which enables support agent to retrieve relevant video snippets based on a query. The main aim of this component is to convert the information contained in the video to a richly annotated text format so that it can be indexed in the same way as documents. The implementation details are listed below.

1. **Noise Removal:** Adaptive filtering (Sambur 1978) is used to remove the noise present in the audio channel which may have seeped in due to the recording environment, microphone processing algorithms, etc.
2. **Speech To Text:** The filtered audio is then converted into time stamped text using Automatic Speech Recognition (ASR). With the application of deep learning techniques, ASR performance has improved dramatically with word error rate dropping to about 6% from 14% a few years ago (Deng, Hinton, and Kingsbury 2013). We use our proprietary speech to text service which has been shown to have a state-of-the-art word-error rate of 5.5% (Saon et al. 2017). We use WaveNet (Oord et al. 2016) and a custom LSTM for language modelling, and an ensemble of three neural networks for acoustic modelling viz., i) an LSTM with multiple feature inputs, ii) an LSTM trained with speaker-adversarial multi-task learning and iii) a residual net (ResNet) with 25 convolutional layers and time-dilated convolutions. This process produces word-level time stamped text *without punctuations* from the audio content.
3. **Transcript Enrichment:** The transcripts obtained from the previous step lack punctuation, and sentence level time-stamps. Lack of punctuations limits the use of these transcripts for subsequent machine processing like machine translation, question answering etc. It is impossible to perform phrase level (contiguous and non-contiguous words) or topic based searches if we cannot identify or segment the obtained transcript into sentences. Therefore, we trained a bi-directional recurrent neural network (BRNN) model with attention similar to (Tilk and Alumäe 2016). It takes in a sequence of words with no punctuation and outputs punctuated text.
4. **Indexing and Search** Finally, the transcripts are indexed with Lucene. This allows us to indirectly search the content of videos using its transcripts. We use Lucene's highlights and Extended Dismax Query Parser (edismax) to retrieve contiguous and non-contiguous text (and hence video) fragments respectively. This allows the user to have a seamless experience of playing the video from the exact moment where a topic of interest starts.

Question & Answer Generation One of the important goals of our system is to reduce the effort of manual curation. Hence, it is essential to identify QA pairs directly from the documents and videos that are ingested for a project. Figure 2a shows the various sub-components of a QA pair generation pipeline. Currently, we create questions from 'factual' sentences and 'mention' sentences. *Factual sentences* are those which contain one or more atomic facts. For eg: "A credit card is a standalone lending instrument.". *Mention sentences* are those which talk about a location such as "the following table...", "the above figure...", etc. We briefly explain each subcomponent of this pipeline below.

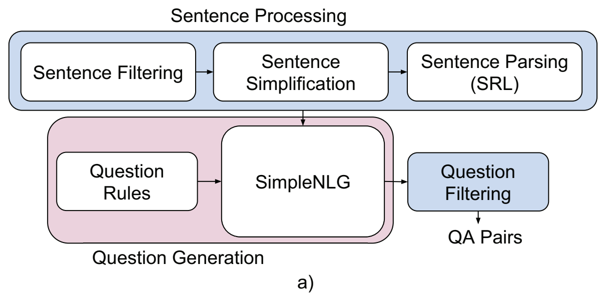
1. **Sentence filtering:** We identify important sentences from a chunk of text which are candidates for QA pair generation. We achieve this by first creating an extractive summary using TextRank algorithm (Mihalcea and Tarau 2004) retaining 10% of the sentences. Further, we filter the sentence using a domain specific glossary constructed automatically by using tf-idf scores from the domain corpus.
2. **Sentence simplification:** The candidate sentences can sometime be complex in nature. Such sentences are broken into simpler ones using (Heilman and Smith 2010), such that the truth value of the simpler sentences are always true.
3. **Sentence parsing:** The candidate set of sentences are parsed using Semantic Role Labeling (SRL) (Pradhan et al. 2004) to get the constituents. For each predicate in a sentence, further identified constituents which are either a semantic role (agent, patient, instrument, etc.) or an adjunct (location, manner, temporal etc.). We implemented the SRL using a SVM (Cortes and Vapnik 1995) trained on the WSJ corpora from the PTB dataset (Marcus et al. 1994). These semantic labels are used to form grammatically correct questions following the rules described next.
4. **Question formation rules:** We identify the question type (Who, What, How, When etc.) to be generated for each constituent of a predicate based on the combination of semantic role and the named entity class . For e.g if the word is a constituent, the role is an agent and named entity class is a person, then we generate a *Who* question. We constructed 42 such static rules (similar to (Mannem, Prasad, and Joshi 2010)) based on the combination of constituent, role and named entity class.
5. **Question Generation:** For each candidate sentence, we use SimpleNLG[4] an open-source library for generating natural language question based on thier SRL annotations and the type of question identified in the previous steps.
6. **Question Filtering:** The questions generated can sometimes be grammatically incorrect or too generic and not relevant to the domain. We use some heuristics such as length of the question, presence of a noun phrase, etc. to retain high quality questions and discard others.

2.2 Cognitive Component

In this section, we talk about the core cognitive component of the platform. This layer uses the knowledge ingested in the previous stages to answer questions from the users. There are three components in this layer, each of which takes a different approach to tackle the user's questions.

- **QA classifiers**: We can look at QA as a question classification problem. One or more questions in the ingested QA pairs can have the same answer. The first question for which a new answer is written/ingested becomes a 'primary question' and all other questions which have the same answer become it's 'alternate questions'. Each primary question and it's corresponding alternate questions form a class. The purpose of these classifiers is to classify incoming questions into one of these classes. A simple SVM and an advanced CNN classifier are built to suit the needs of different users
- **Dynamic Disambiguation:** Sometimes the users' questions are not complete. Additional information/context is required before the classifiers can classify the question correctly with a high confidence score. A sophisticated disambiguation module is built to deal with such cases of incomplete data. A knowledge graph is built using the ingested data and is used by the disambiguation module to identify the gaps in the user's question and ask for relevant information
- **Search**: If a question cannot be classified with confidence even after getting missing information from the user, a simple tf-idf and bag of words based language model is used to search for similar sentences and retrieve them

[4]https://github.com/simplenlg



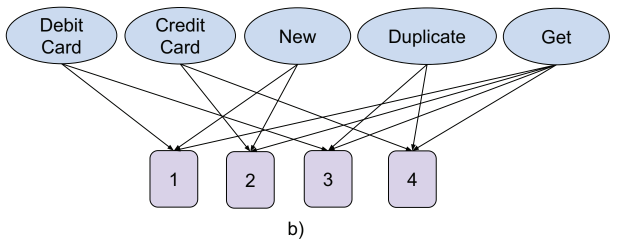


Figure 2: a) Question generation pipeline. b) Knowledge graph created on the resultant questions.

QA Classifiers There are two types of classifiers in our platform which can be configured for a project based on users' specific needs: i) Dependency based CNN classifier (DCNN) and ii) Support Vector Classifier (SVC). There is a trade off between time taken to train and accuracy between these two classifiers. If there is need at the project level, to have an incoming question/answer pair to be available immediately as part of the classifier (especially when the new answer is introducing a new class), then SVC is preferred as it can be trained in a couple of minutes. However, if the project is comfortable with a delay in the incoming question/answer pair's availability in the classifier, then DCNN is preferred. The details of the classifiers are explained below.

1. **Dependency based CNN Classifier** We used CNNs be-

cause (Ruder, Ghaffari, and Breslin 2016) showed that CNNs give better classification accuracy on short pieces of text when compared to RNNs. The DCNN classifier is similar to the model proposed in (Kim 2014), but it also considers the ancestor and sibling words in a sentence's dependency tree. Here, we consider two types of convolutions: *Convolutions of ancestor paths* and *Convolutions on Siblings*. For a detailed explanation of these convolutions, please see (Ma et al. 2015).

We use max-over-time pooling (Kim 2014) to get the maximum activation over the feature maps. In DCNNs, we want the maximum activation from the feature map across the whole dependency tree (whole sentence). This is also called 'max-over-tree' pooling (Ma et al. 2015). Thus, each filter outputs only one feature vector after pooling. The final representation of a sentence will be many such features, one from each of the filters in the network. These feature vectors are finally passed on to a fully connected layer for classification.

The model we built is similar to what is done in (Ma et al. 2015). We concatenated the feature maps obtained from ancestor path and sibling convolutions with the sequential n-gram representation. The concatenation with the sequential n-gram representation was required since the questions that the model is being trained on may contain grammatical flaws which will result in parsing errors during dependency tree construction, but these parsing errors do not affect the sequential representation.

2. **Support Vector Classifier** A standard Support Vector Classifier with a linear kernel is used with a one-vs-rest scheme for multi-class support. We trained it with L2 penalty and hinge loss. We built it using the $liblinear$ library (Chang and Lin 2011) written in C. This makes it extremely fast to train and deploy.

Dynamic Disambiguation Consider an example from the banking support domain, where a customer asks a question like 'How do I get a card?'. This is a very generic question and the QA classifier classifies it into multiple semantically related classes such as:

1. How can I get a new debit card?
2. What is the process to get a new credit card?
3. I want a duplicate debit card
4. How do I get a duplicate credit card?

The confidence scores of these classifications are not high as all these questions are similar to each other. Therefore, when the highest confidence from the classifiers go below a configurable threshold, the dynamic disambiguation module is invoked. The user is asked to clarify the intent with a question like 'Do you need a Debit Card or Credit Card?'. Based on the response, further disambiguating question like 'Do you want to apply for a new card or a duplicate card?' is posted. Once these relevant information are obtained, the context becomes clear and the classifier can easily identify the relevant QA pair and display the correct answer.

To perform such disambiguations, we constructed a knowledge graph (KG) on the fly using the top-k QA pair returned by the classifiers. The KG is a bipartite graph where all the concepts like noun phrases, verb phrases, etc. form one group and the references to the questions form another group as shown in 2b. These concepts are extracted using dependency parsing (De Marneffe and Manning 2008b) (De Marneffe and Manning 2008a). Similar concept buckets are identified from the first group and one question is asked for each bucket. The questions are usually very simple and hence are template based. Once the user clarifies the concept(s) she is interested in, the answer which is connected to those concept(s) is shown.

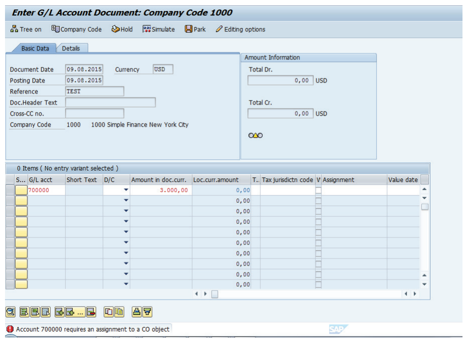
Search A tf-idf based search system is built on top of Apache Lucene. Lucene offers an $O(n)$ search time, where the text chunks (from the document and AV ingestion pipelines) are indexed using a simple bag-of-words scheme. For an input question, the ten most similar documents/videos along with the closely related chunks are extracted by checking for maximum cosine similarities on their tf-idf representations.

Resolution Automation Sometimes it is not enough to just provide a solution to a user's problem. For eg., if the user is facing an issue during login, it would be a great user experience if the password is reset by the system directly, rather than giving out the instructions for the user to do the same. Therefore our platform supports an automaton to be attached to an answer. When displaying such an answer, the user is alerted to the possibility of the system taking some corrective action on their behalf. If the user agrees, the automaton is executed by the system and the problem is resolved.

2.3 Assistant

The main purpose of the Assistance layer is to hide the complexities of the system from the users and provide them with an easy to use interface. The user can interact with our platform in two ways: text and images. The user can ask a question or can upload a screenshot of the error he is facing. The output of the system is always rich text. This layer contains five components, namely:

1. **iSight**: If the user chooses to upload an error screenshot, this component extracts information about the error and certain UI artifacts which may help in resolving the error
2. **Dialog**: This is a standard dialog module which takes in user's natural language inputs and converts it into a structured and annotated format understandable by the orchestrator. It also takes in output from the orchestrator and generates natural language text for the user to read
3. **Orchestractor**: As the name suggests, this component orchestrates among all the components of the brain to get a single answer and show it to the user
4. **LOTJ**: This **L**earning **O**n **T**he **J**ob component constantly listens to the interactions between the user, the orchestrator and the support agent to identify new tidbits of information which is ingested into the system as new knowledge usable in future conversations



Enter G/L Account Document: Company Code 1000

Tree on · Company Code · Hold · Simulate · Park · Editing options

Basic Data · Details

Document Date	09.08.2015	Currency	USD
Posting Date	09.08.2015		
Reference	TEST		
Doc.Header Text			
Cross-CC no.			
Company Code	1000	1000 Simple Finance New York City	

Amount Information

Total Dr. 0,00 USD

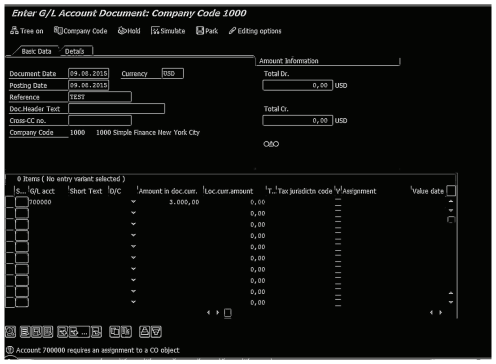
Total Cr. 0,00 USD

0 Items (No entry variant selected)

S...	G/L acct	Short Text	D/C	Amount in doc.curr.	Loc.curr.amount	T..	Tax jurisdictn code	V	Assignment	Value date
	700000			3.000,00	0,00					
					0,00					
					0,00					
					0,00					
					0,00					
					0,00					
					0,00					
					0,00					
					0,00					
					0,00					

Account 700000 requires an assignment to a CO object

Figure 3: An example screenshot image of a SAP system having an error message displayed. The user query input to find a solution for the error was, "*How to fix error in CO object?*".



Enter G/L Account Document: Company Code 1000

Tree on · Company Code · Hold · Simulate · Park · Editing options

Basic Data · Details

Document Date	09.08.2015	Currency	USD
Posting Date	09.08.2015		
Reference	TEST		
Doc.Header Text			
Cross-CC no.			
Company Code	1000	1000 Simple Finance New York City	

Amount Information

Total Dr. 0,00 USD

Total Cr. 0,00 USD

0 Items (No entry variant selected)

S...	G/L acct	Short Text	D/C	Amount in doc.curr.	Loc.curr.amount	T..	Tax jurisdictn code	V	Assignment	Value date
	700000			3.000,00	0,00					
					0,00					
					0,00					
					0,00					
					0,00					
					0,00					
					0,00					
					0,00					
					0,00					
					0,00					

Account 700000 requires an assignment to a CO object

Figure 4: The resulting image of the proposed image preprocessing and enhancement pipeline on which the Tesseract OCR is performed.

iSight Sometimes it is easier for the users to upload the screenshot of an error they are facing rather than describing it in text. Consider Figure 3 which shows an error in an SAP system UI (the red fields). It may be difficult for a novice user to describe this error and hence might end up asking a generic question like "*How to fix error in CO object?*". This will result in a unnecessary disambiguation process in this context, which can be avoided if the system accepts the error screen shot directly and parses them to identify the errors accurately.

We used the Tesseract OCR (Smith 2007) library to extract text from an image. Upon initial testing, we found out that the performance of Tesseract's text extraction is influenced by, (i) font size, (ii) font color, (iii) background color, and (iv) font type. Hence, we built the following image preprocessing pipeline to enhance the text in screenshot images: i) Converted RGB color to grayscale for withering the effect of font and background colors ii) Removed the effect of font types such as bold and italics, by sharpening the text using two levels of Difference of Gaussians iii) Normalized the font size by enhancing the resolution of the images (super-resolution) using bicubic interpolation iv) converted the image into a binary format by performing adaptive thresholding using Otsu's algorithm and v) completed the binarization loss through one level of dialation using a square structuring element.

The resulting enhanced screenshot image is shown in Figure 4 from which text is extracted using Tesseract. A general purpose English dictionary along with a domain specific vocabulary built using ingested documents is used to correct spellings in the OCR extracted text. Further, a simple classifier is used to recognize the application to which this UI screenshot belongs to and heuristics are applied to extract the error message and meta information. This is then restructured appropriately and fed into the orchestrator for further processing.

Dialog This component does a two way conversion between natural language and a structured data format understandable by the system. It extracts the intent in the user's question and the relevant entities the question is talking about. These intents are entities are then processed and enriched using a fuzy matching algorithm (Baeza-Yates and G. Navarro 1999) before being sent to the orchestrator.

Orchestrator The orchestrator component acts as a middleman between the user and the platform. It communicates with all the components in the cognitive component to get relevant answers and decides what to show to the user. It also gathers feedback from the user at every step to identify it's own shortcomings. The major steps taken by the orchestrator when a new conversation starts with a user is as follows: i) The user either uploads a screenshot or enters a question which is then pre-processed to a structured format and given to the orchestrator ii) The orchestrator initially contacts the classifiers to get top-k answers for the question asked and their confidence scores iii) If the confidence of an answer is greater than a threshold, the answer is shown. Else, the dynamic disambiguation module is called and the orchestrator asks counter questions to the user to clarify the context iv) After clarification, if the confidence of the classifiers increase above the threshold, the answer is shown to the user. Else, the search module is invoked vi) If the high confidence answer shown to the user has a automaton attached to it, the orchestrator takes the user's permission and executes the automaton vii) At any time, if the user is not happy with the answer provided, the orchestrator does a smooth hand-off to a human agent

Learning on the job The purpose of this module is to identify new QA pairs which can be harvested from interactions with the system. It does so in two ways: i) whenever the search module is invoked and the user provides a positive feedback on a document/AV chunk, the LOTJ module associates the question asked by the user to this chunk and forms a new QA pair and updates the associated weight based on the frequency. If this weight becomes greater than a pre-defined threshold, the QA pair is ingested into the system ii) when the user is dissatisfied with the answers given by the system and a hand-off is made to the human agent, the module listens to the conversation between the user and the agent to identify potential QA pairs, updates the associated weights and the same process is followed as before.

Metric	Value
Projects	675
Users	5800
Avg. users / project	86
Total QA Pairs	306930
Avg. QA pairs / project	454.7
Queries per day	453
Avg. time saved / user	455 min/year
Questions answered by	
a) Classifiers	106033
b) Dynamic Disambiguation	12590
c) Search	113351

Table 1: Usage summary of the solutions in production

At any point in time, the support agents can see the potential QA pairs created by the LOTJ module. They can then edit them if required and approve/reject them for immediate ingestion. Thus, this module provides a non-intrusive way of accumulating new knowledge. This, along with ingested knowledge is cleaned to remove all sensitive client-specific data to bootstrap new projects quickly with minimal effort.

3 Business Impact

Our platform has enabled the deployment of multiple general purpose assistance solutions to reduce the time taken to resolve incidents, automate the resolution of incidents and reduce incident volumes resulting in support agent productivity improvement. Some of the solutions built on our platform are: (i) *Employee Assist*: Assistance to business users; helping them tackle day-to-day issues they face while interacting with various applications (like HR portal, Travel portal, etc) and there by reducing the overall volume of incidents raised (ii) *Agent Assist*: Assistance to support agents (L1/L2) in helping them resolve incidents faster (iii) *Coding Assist*: Assistance to SAP ABAP programmers to help improve their productivity in development of custom ABAP modules (iv) *Dynamic Automation*: Automating the execution of typical service requests orchestrating across robotic process automatons, thereby reducing the need for a human in the loop.

These solutions have been deployed across multiple accounts and helped significantly in revenue generation and savings. Table 1 summarizes current usage of these solutions in a production environment.

Here are some actual quotes from the user community who have actively adopted our solutions:

1. *"Using Agent Assist we are able to resolve tickets faster. This one implementation has given a lot of mindshare with the client and is a key ingredient in our contract renewal proposal with the client"*
2. *"Over 30+ support agents were able to use the solution to resolve 30% of the tickets. Saving time on ticket resolution means helping end users quicker, and reducing the effort and cost required to support the application in question. Win-win!'*
3. *"We are thrilled to have a successful commercial deployment of the automation solution. This kind of automation is becoming a fundamental way GBS delivers services, a definite stepping stone towards self healing applications and wider process integration"*

The project initially started out with 5 researchers working for 6 months to create a prototype before a dedicated team of developers came onboard. Today, we have a development team of around 30 people including testers as part our services business units who manage this product following an agile development methodology. There are monthly releases consisting of new enhancements and bug fixes, following our own devops pipeline to build, test and deploy the solutions to production. The solutions built on our platform are being used across the world in countries like USA, India, Brazil, etc.

4 Conclusion

We have developed a cognitive QA platform aimed at automating enterprise help desks for IT support and maintenance. A generic orchestration framework is designed that integrates diverse QA methods and performs context switching in real-time to combine the advantages of the individual methods. We also discuss knowledge ingestion, training & configuration, the user experience and the continuous learning & adaptation aspects of this system. We also emphasize on how including a human expert in the loop guarantees a high-quality user experience. Currently, the QA system provides support to 675 enterprise projects, thereby improving support agent productivity while reducing the turnaround time for incident resolution and also attracting a huge revenue.

References

Baeza-Yates, and G. Navarro, R. 1999. Faster approximate string matching. *Algorithmica* 23(2):127–158.

Baudiš, P., and Šedivỳ, J. 2015. Modeling of the question answering task in the yodaqa system. In *International Conference of the Cross-Language Evaluation Forum for European Languages*, 222–228.

Bordes, A.; Usunier, N.; Chopra, S.; and Weston, J. 2015. Large-scale simple question answering with memory networks. *arXiv preprint arXiv:1506.02075*.

Cao, Y.; Liu, F.; Simpson, P.; Antieau, L.; Bennett, A.; Cimino, J. J.; Ely, J.; and Yu, H. 2011. Askhermes: An online question answering system for complex clinical questions. *Journal of biomedical informatics* 44(2):277–288.

Chang, C.-C., and Lin, C.-J. 2011. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.* 2(3):27:1–27:27.

Chen, H.; Chiang, R. H.; and Storey, V. C. 2012. Business intelligence and analytics: From big data to big impact. *MIS quarterly* 36(4):1165–1188.

Clark, P.; Etzioni, O.; Khot, T.; Sabharwal, A.; Tafjord, O.; Turney, P.; and Khashabi, D. 2016. Combining retrieval, statistics, and inference to answer elementary science questions. *AAAI Conference on Artificial Intelligence*.

Cortes, C., and Vapnik, V. 1995. Support-vector networks. *Machine learning* 20(3):273–297.

De Marneffe, M.-C., and Manning, C. D. 2008a. Stanford typed dependencies manual. Technical report, Technical report, Stanford University.

De Marneffe, M.-C., and Manning, C. D. 2008b. The stanford typed dependencies representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, 1–8.

Deng, L.; Hinton, G.; and Kingsbury, B. 2013. New types of deep neural network learning for speech recognition and related applications: An overview. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, 8599–8603. IEEE.

Fang, H.; Wu, F.; Zhao, Z.; Duan, X.; Zhuang, Y.; and Ester, M. 2016. Community-based question answering via heterogeneous social network learning. In *AAAI Conference on Artificial Intelligence*.

Hauswald, J.; Laurenzano, M. A.; Zhang, Y.; Li, C.; Rovinski, A.; Khurana, A.; Dreslinski, R. G.; Mudge, T.; Petrucci, V.; Tang, L.; et al. 2015. Sirius: An open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers. In *ACM SIGPLAN Notices*, volume 50, 223–238.

Heilman, M., and Smith, N. A. 2010. Extracting simplified statements for factual question generation. In *Proceedings of QG2010: The Third Workshop on Ques-tion Generation*, 11.

Hofmann, T. 1999. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, 50–57. ACM.

Kim, Y. 2014. Convolutional neural networks for sentence classification. *CoRR* abs/1408.5882.

Kolomiyets, O., and Moens, M.-F. 2011. A survey on question answering technology from an information retrieval perspective. *Information Sciences* 181(24):5412–5434.

Lehmann, J.; Furche, T.; Grasso, G.; Ngomo, A.-C. N.; Schallhart, C.; Sellers, A.; Unger, C.; Bühmann, L.; Gerber, D.; Höffner, K.; et al. 2012. Deqa: deep web extraction for question answering. In *International Semantic Web Conference*, 131–147.

Ma, M.; Huang, L.; Zhou, B.; and Xiang, B. 2015. Tree-based convolution for sentence modeling. *CoRR* abs/1507.01839.

Mannem, P.; Prasad, R.; and Joshi, A. 2010. Question generation from paragraphs at upenn: Qgstec system description. In *Proceedings of QG2010: The Third Workshop on Question Generation*, 84–91.

Marcus, M.; Kim, G.; Marcinkiewicz, M. A.; MacIntyre, R.; Bies, A.; Ferguson, M.; Katz, K.; and Schasberger, B. 1994. The penn treebank: annotating predicate argument structure. In *Proceedings of the workshop on Human Language Technology*, 114–119. Association for Computational Linguistics.

Mihalcea, R., and Tarau, P. 2004. Textrank: Bringing order into texts. In Lin, D., and Wu, D., eds., *Proceedings of EMNLP 2004*, 404–411. Barcelona, Spain: Association for Computational Linguistics.

Mitra, A., and Baral, C. 2015. Addressing a question answering challenge by combining statistical methods with inductive rule learning and reasoning. *AAAI Conference on Artificial Intelligence*.

Ng, J.-P., and Kan, M.-Y. 2015. Qanus: An open-source question-answering platform. *arXiv preprint arXiv:1501.00311*.

Oh, J.-H.; Torisawa, K.; Hashimoto, C.; Iida, R.; Tanaka, M.; and Kloetzer, J. 2016. A semi-supervised learning approach to why-question answering. In *AAAI Conference on Artificial Intelligence*.

Oord, A. v. d.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A.; and Kavukcuoglu, K. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Pradhan, S. S.; Ward, W. H.; Hacioglu, K.; Martin, J. H.; and Jurafsky, D. 2004. Shallow semantic parsing using support vector machines. In *HLT-NAACL*, 233–240.

Ruder, S.; Ghaffari, P.; and Breslin, J. G. 2016. Character-level and multi-channel convolutional neural networks for large-scale authorship attribution. *ArXiv e-prints*.

Sambur, M. 1978. Adaptive noise canceling for speech signals. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26(5):419–423.

Saon, G.; Kurata, G.; Sercu, T.; Audhkhasi, K.; Thomas, S.; Dimitriadis, D.; Cui, X.; Ramabhadran, B.; Picheny, M.; Lim, L.; Roomi, B.; and Hall, P. 2017. English conversational telephone speech recognition by humans and machines. *CoRR* abs/1703.02136.

Shapiro, R. J. 2014. The us software industry as an engine for economic growth and employment. *Georgetown McDonough School of Business Research Paper* (2541673).

Smith, R. 2007. An overview of the tesseract ocr engine. In *International Conference on Document Analysis and Recognition*, volume 2, 629–633. IEEE.

Tilk, O., and Alumäe, T. 2016. Bidirectional recurrent neural network with attention mechanism for punctuation restoration. In *INTERSPEECH*, 3047–3051.

Xu, K.; Zhang, S.; Feng, Y.; Huang, S.; and Zhao, D. 2015. What is the longest river in the usa? semantic parsing for aggregation questions. In *AAAI Conference on Artificial Intelligence*, 4222–4223.

Yang, L.; Ai, Q.; Spina, D.; Chen, R.-C.; Pang, L.; Croft, W. B.; Guo, J.; and Scholer, F. 2016. Beyond factoid qa: Effective methods for non-factoid answer sentence retrieval. In *European Conference on Information Retrieval*, 115–128.

Yao, X., and Van Durme, B. 2014. Information extraction over structured data: Question answering with freebase. In *ACL (1)*, 956–966.

Zheng, Z. 2002. Answerbus question answering system. In *international conference on Human Language Technology Research*, 399–404.