A NATURAL LANGUAGE INTERFACE FOR SEARCH AND RECOMMENDATIONS OF DIGITAL ENTERTAINMENT MEDIA

Sashikumar Venkataraman and Nizam Ahmed Mohaideen

Rovi Corporation, USA

ABSTRACT

We describe an integrated platform that combines a search and recommendations system of digital media with a novel conversation interface that enables users to use natural-language conversation for performing a variety of tasks on the digital content and information retrieval relating to meta-content. This advanced platform is built over a knowledge graph that consists of millions of tagged entities, along with structured relationships and popularities crawled and ingested from multiple sources, continuously evolving over time.

The voice system uses a unique extensible architecture that combines natural language processing (NLP) techniques with named entity recognition of the knowledge graph determining both intent as well as entities extracted from user queries. Relationships between entities in the knowledge graph aid in identifying the right entities thereby creating meaningful, contextual, and temporally relevant interpretations.

INTRODUCTION

In the last couple of years, there has been a rapid growth of second screen devices such as mobiles and tablets to drive digital entertainment systems. At the same time, we are also witnessing a robust technology from the speech recognition community for converting voice audio to spoken text. The fusion of these two is now making voice interfaces and natural language a viable and practical solution for several tasks in the living room; many of these tasks would have involved inputting text on more cumbersome and text-input constraint devices such as TV remotes and mobiles. It is just a matter of time now for voice interfaces to become the default interface for several devices in the future digital home.

In this paper we describe a natural-language interface to perform a variety of functionalities pertaining to video and music content in relation to end-user tasks pertaining to digital media content. Examples of such tasks or intents involve retrieving search results and getting personalized recommendations, driving common TV-control commands, getting more information from the media meta-content and answering trivia questions pertaining to them, and checking availability of the content in a channel lineup or on-demand catalog. Along with the rich set of query intents, the conversation system supports queries that involve entities spanning a comprehensive knowledge graph.

Examples of such queries are "Show me some old Bond movies without Roger Moore" or "Who played Tony Montana in Scarface?"

This problem of building conversational question answering (QA) systems has been a hot topic in industry and academia for several years (1, 2). A QA system aims at providing precise textual answers to specific natural language users' queries rather than typical search engine results that give a set of matching documents. Of late, many of these systems are based on ontology wherein the knowledge-based data has a structured organization defined by an ontology (3, 4). Users could raise questions in natural language and the system will return accurate answers to users directly after question analyzing, information retrieval and answer extraction. Ontology knowledge base provides a convenient way to incorporate semantic understanding of user queries, but the natural language needs to be mapped to the query statement of ontology. Examples of such ontologybased QA are AquaLog (5), QASYO (6) and more recently Siri by Apple. AquaLog is a semiautomatic QA system that combines natural-language processing, ontologies, logic, and information retrieval technologies. QASYO is a QA system built over Yago that integrates the ontology of WordNet with the facts derived from Wikipedia. In all these systems, the input natural-language query is first translated to an intermediate representation compatible with the ontology and this intermediate query is then used to find the final results.

In the current work, we use the ontology based on the Rovi Knowledge Graph (RKG) that incorporates factual information of all 'known' or 'named' things. This includes countless movies and TV shows, music albums and songs, as well as countless known people such as actors, musicians, celebrities, music bands, known companies and business establishments, places, sports teams, tournaments and players, etc. All the facts pertaining to these entities are crawled from multiple sites such as Wikipedia, Freebase, and many others and correlated so as to create a unique smart tag (with a unique identifier) to represent each entity in the RKG. Two entities can stand in a relation and there are multiple kinds of structured relationships that exist in the RKG such as movie-director, team-player, etc. The relations between the entities also get created by aggregating factual knowledge from several structured data sources and are further augmented with unstructured relationships using data-mining techniques such as analysis of hyperlink structures within Wikipedia. The facts of the RKG represented via entity-identifiers are hence separated out from the language-specific lexical ontology such as WordNet. The lexical ontology is mainly used in understanding the intent of the query through natural language parsing and pattern matching techniques; whereas the named entity extraction is based on the RKG. The intent and entities are then combined together to retrieve the final answer to the user query.

Though intent and entity recognition are very closely dependent on each other, the conceptual separation of these two components give more modular organization and also permits flexible extensions of the conversation system. For example, consider the queries "How long is iron man?" and "How long is San Mateo?" In the former query, intent is "running time of movie" whereas in the latter query, the intent is "distance". This intent-ambiguity here can be resolved with the help of named entities. On the other hand, consider the query "play lights?" While "lights" can refer to several concepts, the entity-ambiguity is resolved by using the intent "play" and prefers either songs or movies.

Another unique aspect of the proposed voice system is the seamless handling of session continuity, where entities as well as intent spread across multiple queries. For example, the user may start with a broad query such as "how about some good action movies" and then narrow the search in the next query as "any of these by Tom Cruise?" and so on. The system is able to intelligently tie in the context of the first query in the interpretation of the second query.

ARCHITECTURE OVERVIEW

The central portion of the conversation system is the understanding of the natural-language query in terms of intent and entity. Based on the entities and detailed intent, the fetching of the final results is then a more straightforward problem of doing a lookup in a structured database or some similarly organized data storage. This query-understanding phase consists of two main parts; Named Entity Recognition and Intent Recognition, both closely interleaved with each other. The named entity recognizer determines the “entities” involved in the query using several classification algorithms, wherein each entity corresponds to a particular node in the RKG. The intent recognizer determines the “intent” of the query that encapsulates the necessary information for determining the final results from the entities in the query. This is done using several NLP and machine-learning algorithms. Although the entity recognition and the intent recognition are two separate sets of algorithms, they are intricately interleaved since the entity recognition algorithms yield more accurate results using the final predicted intent; and the final intent is in turn aided from the recognized entities in several ambiguous situations.

An overall flow of the conversation pipeline is shown in Figure 1 below. The initial query is usually the output of an automatic speech recognition (ASR) system, though in certain cases it can be from a non-voice interface that permits natural language text input. This query is fed into the first stage of the entity recognizer. This stage mainly involves tokenizing the query into the possible phrase candidates using n-grams, and determining all the possible entities associated with each phrase. For efficiency reasons, in this stage the determination of the possible entities for each phrase is done independent of other phrases in the query. Due to the wide coverage of the underlying knowledge graph, it is possible to get a lot more possible phrase candidates at this early stage and several hundred entities associated with any particular phrase.

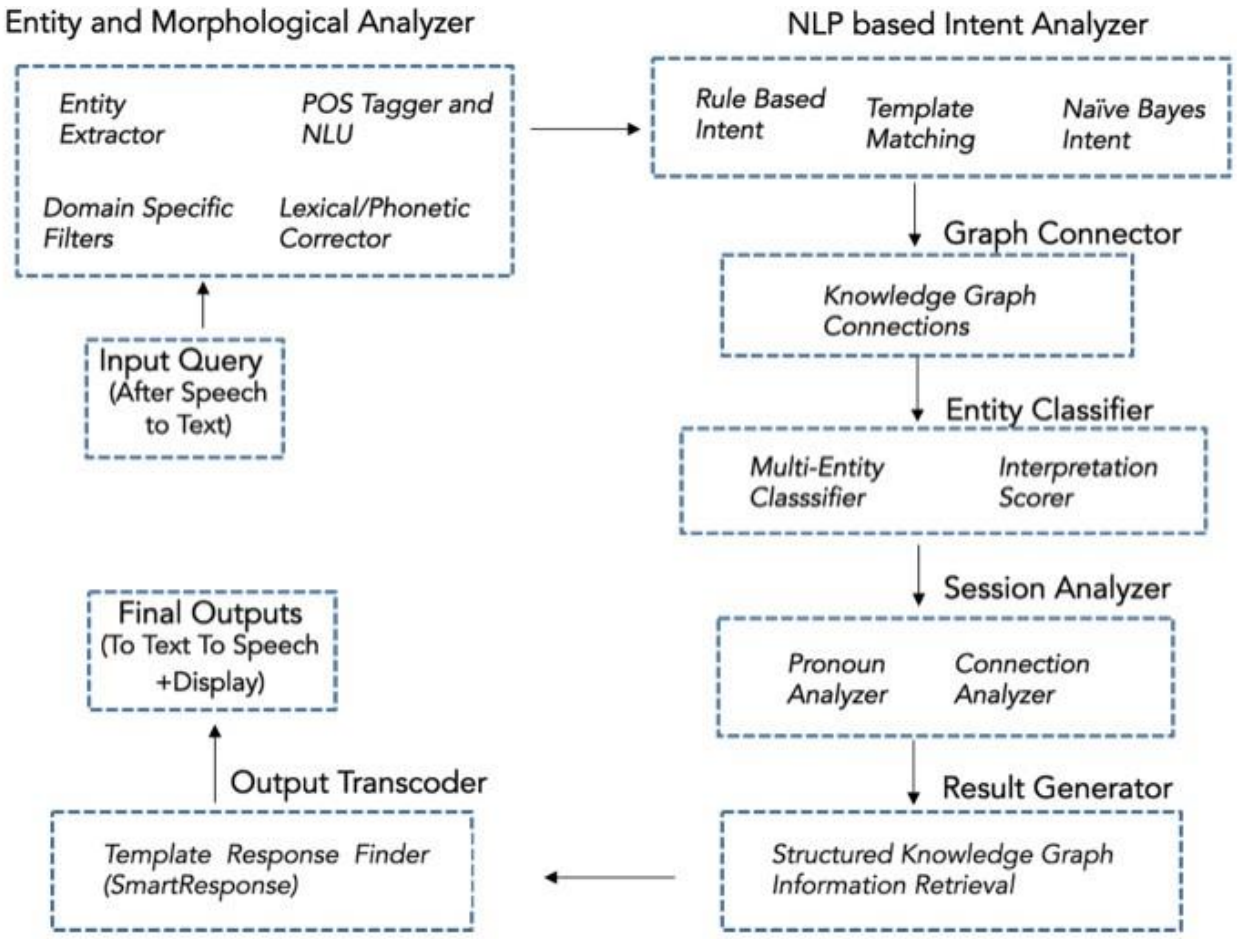

Figure 1 - Stages involved in resolving a natural language query

This stage is then followed by the intent recognizer that determines the most likely intent in the query using a combination of several machine learning algorithms such as template-based, bag-of-word and Naïve Bayes classification techniques. The next two steps involve the core of entity recognition that uses the predicted intent and several classification algorithms to determine precisely the entities corresponding to the phrases in the query. One important input to these algorithms is provided by the graph connector, which determines the various pairs of entities that are connected within the same query. For example, in the query “play Lights by Goulding”, the term “lights” could mean several entities like the movie “lights” or different songs with “lights” in the title. The term “Goulding” could also refer to people in other industries. However, there is a unique pair that has a graph connection, namely the song “Lights” and the artist “Ellie Goulding”. This classifier can then use this connection to determine precisely these entities. The next step in the pipeline is the handling of conversational sessions. One of the unique aspects of the natural language interface is to allow conversational continuity across several queries. To enable this, a context is maintained that consists of the entities recognized in earlier queries. Pronouns in the current query are automatically tied to those context entities. Furthermore, the system is able to switch context automatically whenever the user’s new query involves entities that are not graph connected to the context entities. The goal of all the above steps is to understand the natural language unstructured query in terms of a

more structured query that can be fed to a database or an information retrieval (IR) system for retrieving the set of final results. This fetching of final results is done in the last step and then presented back to the user. One key difference from standard search results is the ability to narrow down a limited set of results (which could be a single result in many cases) instead of always showing multiple results. Furthermore, a smart-response engine is used to reply back to the user in a more natural language in the same spirit as the input query. In the following sections, we will explain each of these stages in greater detail.

NAMED ENTITY RECOGNITION

The Entity Recognition system is primarily based on the RKG, which is a database of countless “known” or “named” things. The RKGis a dynamic system that has been created by crawling multiple metadata sources that evolve and are refreshed continuously over time. The video knowledge graph has around 900K movies, 250K TV shows, 4M episodes, 4M personalities, 20K tournaments. Apart from carrying rich meta-data associated with each entity, the knowledge graph also provides other useful information that aid in named entity recognition. For example, many entities have AKA or alternative names and these provide different ways to query the same entity (such as MI 2 for Mission Impossible 2). The named entity classifier uses this information to associate the right entities with the aka phrases along with the titles.

The knowledge graph also provides association strengths between phrases and the entities that is used in the classification of a phrase to the right entity. For example, the term “newman” can refer to multiple entities in the RKG: a role “Newman” (in Seinfeld), a team “Club
Newman”, actors Paul Newman and Laraine Newman, a movie “New Man”, and several other possibilities. Each of these possibilities is given a particular weight in the scale between 0 and 100 depending on the frequency of reference of that entity with the phrase “newman”. So in this example, the highest weight (60) is for the role entity “newman” due to matching the complete title followed by Paul Newman that has weight 43. In a plain query containing just “newman”, the default entity would be the role-entity since it has the highest strength. But the situation would be quite different if there are more entities that are extracted from other phrases in the query and have specific relations to the entities corresponding to “newman”. This is discussed in detail in the following sections.

Graph Connections

Consider the query “newman as rooney”. In this query, there are two phrases extracted, “newman” and “rooney” extracted, each having multiple entities. Just like before “newman” has entities, such as actors, role, and movie. “Rooney” matches the famous football player, several roles with that title, some musicians, and also some people with title “Rodney” (matching via lexical correction). After this stage of entity extraction, we enter a graph connection phase wherein entities across different phrases get connected to each other based on a connection in the RKG. In that stage, an important connection is found between the actor Paul Newman and the role of John Rooney, since he played this role in the movie Road to Perdition. Due to this connection, these entities get preferred over all other entities, and are the final ones retained in the entity extractor. However, if the query had been “newman as frog-eyed woman” that is input to the conversation system, a graph

connection is found between Laraine Newman the actress and the role “frog-eyed woman” because Laraine Newman played the role of frog-eyed woman in the movie “Fear and Loathing in Las Vegas”. Due to this connection, these entities are preserved by the entity extractor.

Graph connections are very useful to resolve the right entity when there are multiple conflicting entities that are potential candidates in the query. In the above examples, we saw that the actorrole-movie entity forms an important mutual graph connection between them. Similarly, two actors acting in the same movie can have one kind of graph connection. Another important graphconnection exists between sports teams that play a game or are part of the same tournament. These connections are very temporal in nature and are used in that temporal context to identify the right team. For example, “giants” can either mean San Francisco Giants in baseball or New York Giants in football. But if the query is “Giants Miami”, the context is identified as baseball due to the graph connection between the San Francisco Giants and Miami Marlins. But the query “Giants Bengals” identifies the context to football with New York Giants and Cincinnati Bengals.

Lexical Corrections

Lexical errors are quite common in speech to text conversion, especially ones which are phonetic variations of each other. This is done by using a lexical module that determines the top-k phrases, with k usually between 5 and 10, of the entire vocabulary corresponding to each phrase in the query even if spelled wrong. The top-k phrases are determined based on lexical closeness that uses a combination of Levenstein-distance and edit-distance between strings. One example is the query “neuman,” which gives entities corresponding to this phrase as well as entities corresponding to the phrase “newman” via lexical correction. In general, entities that result via lexical correction are given lesser priority than direct matches. However, the lexical corrected entities may be preferred when they are more popular or when there are other phrases in the query, which result in entities that have a graph-connection with those entities. Apart from lexical variations, the system also supports stemvariations such as plural forms in the phrases of the entities. This module also handles addition/deletion of white-spaces (and stop-words) such as “new man” to “newman” or vice-versa.

INTENT RECOGNITION

Along with entity extractor, the other main piece of the conversation system is the intent recognizer whose primary function is to recognize the structure of the query and determine the intent of the user. For example in the query “who directed Titanic”, the entity extractor recognizes the entity “Titanic” as a movie; while the intent recognizer deduces “director-intent”. Using the entities extracted and the deduced intent, the final results can be found easily. The system currently supports more than 50 intents along with a rich suite of common TV control commands. Some examples of intent supported are listed below:

- Movies and TV shows (e.g. “show me movies of Tom Cruise” or “funny shows tonight”)
- Cast like director, producer, actor (e.g. “who starred in Dark Knight?” or “who played Tony Montana in Scarface?”)
- Song related (e.g. “show me the latest songs of Ellie Goulding”)

- TV or on-demand availabilities (e.g. “is Breaking Bad available on TV?” or “is it available on demand?”
- Sports teams or game time (e.g. “who do the Giants play tonight?” or “when do the Pats play next?”)
- TV control commands such as “channel up”, “volume down”, “switch off”, etc.
- Award-related (e.g. “who won the best actor Academy Award in 2014?”)

The intent recognition module is mainly based on templates that get automatically learnt when queries get manually tagged with the corresponding intent. Templates are sample queries where entities in the query are replaced by placeholders representing the entity-types and a bag-of-words to denote the influential words. Some examples of query templates are given below:

- in which <film> did CREW play ROLE <who> directed MOVIE what is MOVIE <rated>?
- did TEAM <win> in TOURNAMENT

All these templates are generated automatically from a set of user queries with annotated intents and the extracted entities are replaced using the entity-type. For example, in the first template above, the intent is a movie whereas the last template would be a game and a binary sub-intent whether the team won the tournament. Using sample user queries with minimal manual tagging, we were able to create a few thousand templates spanning the range of queries and intents. The bag-ofwords representing each influential word like <film> or <won> are also learnt using a combination of manual tagging and machine learning algorithms by analyzing multiple tagged user queries. For example, the word-bag of <film> also consists of “movie” and “flick”. So even if a query uses the word “flick” instead of “film”, it can still match the first template above as long it is followed by a crew and a role. This permits generalization to newer user-queries without the need to define a specific template for each kind of query. The other words like “in” and “did” are classified as un-influential words during the machine learning process since they occur in several other templates or predefined as stop-words.

Template matching begins by first checking if the words in the input query match any of the influential bag-of-words in the template. This creates a candidate set of templates that matches the structure of the input query. The query and candidate templates are then passed to the entity extractor that tries to prefer the entity-types corresponding to the templates in the specific locations. If the right entity types get recognized for a given candidate template, then that template gets selected, and the corresponding intent is returned. For example, in the query “who directed titanic”, even if there are multiple possible entities for the phrase “titanic” including songs and albums, the entity extractor would prefer the movie for that phrase due to matching the second template and returns a directorintent for the query.

CONVERSATIONAL SESSION

One unique aspect of our conversation system is the seamless handling of a conversational session that spans across multiple queries. A typical conversation session could go as follows:

User: Who directed Titanic?
System: James Cameron directed Titanic.
User: Is it available on Netflix?
System: No. But it is available in Vudu for $2.99.
User: Can you show any of his other movies?
System: Here you go. Here are some more movies of James Cameron. User: Cool. Play the second one.

As shown in the above example the user starts with a specific query, but then may continue using pronouns like “his”, “her”, “it” or “the ones”. In such cases, the pronoun in the subsequent query is replaced by the most suitable entity from the previous context. For example, in the above example, the pronoun “his” gets replaced by “James Cameron”, whereas pronoun “it” gets replaced by the movie “Titanic” and the system responds accordingly. The above example also shows how pronouns can either refer to an entity in the earlier query, such as “Titanic,” or can refer to results to an earlier query, such as “James Cameron.” Another example of using pronouns to refer to a result is in selections. In the above example, when the user finally says “play the second one”, the phrase “second movie” is automatically tied to the second movie in the list of movies shown for the earlier query. Apart from pronouns, the next class of hints used for conversation continuity is the presence of graph connections between entities in the next query with the entities in the previous query. For example, if the earlier query is “Tom Cruise movies” and the next query “with Nicole Kidman”. In this case, the system establishes a graph-connection between “Tom Cruise” and “Nicole Kidman" since there are several movies where both of them have acted together. In such cases, the system gives preference to conversation continuity and shows movies of both “Tom Cruise” and “Nicole Kidman.” However, if the user says a different actor “Tom Hanks” or some sport-team in the next query which does not have any graph-connection with “Tom Cruise”, then the system changes context automatically and treats it as a separate query.

Another kind of hint in establishing conversational continuity is when the user narrows the earlier query with certain “semantic filters” such as “now”, “latest ones”, “tomorrow”, or “on TV”. For example the user starts with “movies of Tom Cruise” and then narrows it to “how about some new ones”. The system then applies the “new” filter to sort all the movies of tom cruise in descending order of release year. Another interesting case of conversation continuity is when the user rectifies an earlier query using some phrase like “I meant…” For example, the user can start a query as “Beethoven”. The system picks the most likely entity and shows all songs or movies of Ludwig van Beethoven. However, if the user really meant movies with Beethoven the dog, the user can rectify the earlier query say “No, I meant the dog”. In such cases, the system performs entity recognition again with phrases “beethoven” and “dog”. This time the entity recognizer chooses the entity Beethoven dog instead of Ludwig van Beethoven.

Conversational sessions are implemented by having a context maintained as part of each query. The context is stored in a runtime storage server such as Redis and contains all entities associated with each phrase in the previous query as well as all the results shown in the previous query. The context reference is passed back to the client and is expected to be sent back from the client in the next query if it intends to support session continuity. When the server receives the context reference, it retrieves back the context entities from

the Redis object and then uses these to resolve future queries if they involve pronouns or graph connections.

RESULT GENERATOR AND SMART RESPONSE

The final stage of the conversation system is the fetching of the results based on the query intent and sending it back to the client with a suitable response. Once the intent and entities in the query are recognized, finding the final result(s) is a straightforward lookup in a data store system such as a database or an alternative in-memory storage system exposed via a HTTP REST API endpoint. Each specific intent is translated to a specific kind of lookup in a specific data-store. Certain kinds of dynamic information for example sport scores may need a specialized data-store (such as redis) to answer corresponding queries. In our implementation, the data-store servers consists of a combination of search/recommendation servers for movie or TV show information, another set of search servers for music information, and dynamic redis-servers for sporting event information. One unique aspect of the result generator phase is the smart-response that gets determined for each query based on the retrieved results. Smart-response is a human-like response that accompanies the set of results that can be spoken back to the user by the client. For example, if the user query is
"who directed Titanic?", the result generator sends "James Cameron" back to the client. Along with the entity, a smart-response text such as "The director of Titanic is James Cameron" is also generated, which can be spoken back to the user. The system has been designed such that multiple smart-responses can be associated with a particular query-response template so that the system looks more varied like natural human conversation and avoids monotonic responses. So for the similar query, the system may at other times respond, "I am quite definite that James Cameron directed Titanic". Currently, the system supports over several hundred templates wherein each template is a response to a specific kind of query. Each template has in turn multiple response templates associated with it and can be extended easily by the administrators in a user-interface below.

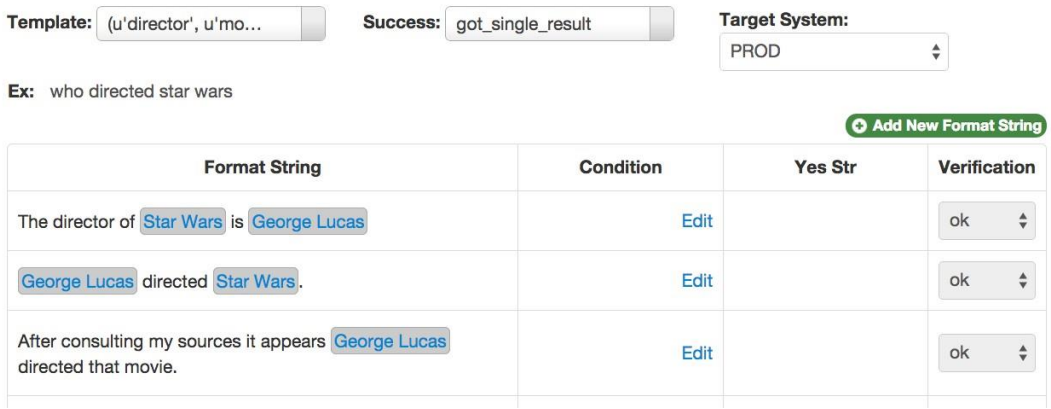

Figure 2 – Smart Response Interface to edit responses for specific query templates

Figure 2 above shows a snapshot of the interface used to add smart-responses dynamically to any chosen query template. The format-string column shows the various

responses that have been added and can be extended by clicking on the “add new format string”. For certain format-strings, one can add a condition (from a predefined set) that gets checked before the corresponding string can get invoked. For example, one may have specific responses that should be used only when the movie that is searched for is also available on a TV channel that day.

PERFORMANCE AND ACCURACY TESTS

We regularly tag the User queries with the expected Intent, Entities and Results expected into the Test System. Overall we collected a diverse [random?] sample of around 21K queries from various user queries and tagged with the right intent and final results. This included 11032 movies, 3257 TV shows, 2721 genres, 10392 personalities, 803 sport-related, 512 TV commands, 3002 awards, 2835 roles and 557 related to on-demand with many queries containing multiple entities. Table 1 explains the accuracy rate of the system on this test set with accuracy distributions across various phases of a QA system. In this test, intent recognition worked over 90% of the time and we obtained over 80% in entity recognition. The intent failures include unsupported intents such as “shows/movies above a specific duration” which are not currently supported; while some of the entity recognition failures were due to misspellings caused by speech-to-text problems. Each sub-system is hosted in separate servers and supports a HTTP rest API. The individual server clusters are horizontally scalable and is able to independently handle at least 10 queries per second. This rate can be increased in specific deployments by scaling individual systems with more processes and/or servers accordingly.

Category	Success Count	Success Percentage
Intent Recognition	19665	90.26%
Entity Recognition	17260	81.22%

Table 1 – Accuracy statistics over 21K queries

CONCLUSION

In this paper, we described a conversation platform that integrated the back-end search and recommendation server with a query-processing engine that can process natural-language queries. The system makes use of a comprehensive knowledge graph for named entity recognition. While the intent recognition phase makes use of the possible named entities, the intent and template recognized serves to determine the final named entities. Graph connections from the RKGare further used to resolve ambiguous entities. Another unique aspect of the system is to handle conversational sessions wherein the user can continue a query context in a subsequent query. The system automatically uses heuristics to either continue the session context with the previous queries or change the context to treat it as a new query. The system also has a

unique smart-response interface wherein human-like responses can be flexibly associated with each kind of query and can be evaluated dynamically based on the results. Currently, we are generalizing the conversational system to handle multiple dataspaces beyond

entertainment. Mainly, we want to generalize the intent recognition phase wherein different kinds of question-answer templates can be trained easily into the system such as weather, stock-prices, ticket booking, etc. The goal is to create a system that can learn the intent and evaluation methodology on-the-fly using training examples. Furthermore, we want to allow the possibility of generating final results from other third-party websites beyond our back-end search/recommendation servers.

REFERENCES

1. Surdeanu M, Moldovan D, 2003. “On the role of Information Retrieval and Information Extraction in Question Answering Systems”, Information Extraction in Web Era - Springer.

2. Sucunuta M E, Riofrio G E, 2010. “Architecture of a Question-Answering System for a Specific Repository of Documents, ” In 2nd Intl. Conference on Software Technology and Engineering.

3. Athira P. M., Sreeja M. and P. C. Reghuraj, 2013. Architecture of an Ontology-Based DomainSpecific Natural Language Question Answering System. International journal of Web and Semantic Technology.

4. Moreda, Paloma., Llorens Hector., Saquete, Estela. and Palomar, Manuel, 2010 “Combining semantic information in question answering systems” Journal of Information Processing and Management 47, 2011. pp 870- 885.

5. V. Lopez, M. Pasin, and Enrico Motta, “AquaLog”, 2005. An Ontology-Portable Question Answering System for the Semantic Web,” Lecture Notes in Computer Science, Vol. 3532, Springer, Berlin, pp. 546-562.

6. Abdullah Moussa and Rehab Abdel-Kader, 2011. QASYO: A Question Answering System for YAGO Ontology. Intl. Journal of Database Theory and Application Vol. 4, No. 2, June, 2011.