An Ontology-Based Interactive System for Understanding User Queries

Giorgos Stoilos(✉), Szymon Wartak, Damir Juric, Jonathan Moore, and Mohammad Khodadadi

Babylon Health, London SW3 3DD, UK
{giorgos.stoilos,szymon.wartak,damir.juric,jonathan.moore, mohammad.khodadadi}@babylonhealth.com

Abstract. The use of ontologies in applications like dialogue systems, question-answering or decision-support is gradually gaining attention. In such applications, keyword-based user queries are mapped to ontology entities and then the respective application logic is activated. This task is not trivial as user queries may often be vague and imprecise or simply don't match the entities recognised by the application. This is for example the case in symptom-checking dialogue systems where users can enter text like "I am not feeling well", "I sleep terribly", and more, which cannot be directly matched to entities found in formal medical ontologies. In the current paper we present a framework for automatically building a small dialogue for the purposes of bridging the gap between user queries and a set of pre-defined (target) ontology concepts. We show how we can use the ontology and statistical techniques to select an initial small set of candidate concepts from the target ones and how these can then be grouped into categories using their properties in the ontology. Using these groups we can ask the user questions in order to try and reduce the set of candidates to a single concept that captures the initial user intention. The effectiveness of this approach is hindered by well-known underspecification of ontologies which we address by a concept enrichment pre-processing step based on information extraction techniques. We have instantiated our framework and performed a preliminary evaluation largely motivated by a real-world symptom-checking application obtaining encouraging results.

1 Introduction

Knowledge Bases (KBs) have started to play a key role in many academic and industrial-strength applications like recommendation systems [8], dialogue-systems [19], and more. In such applications users form their requests using short queries, e.g., "I want to book a flight", "I am looking for Italian Restaurants", "I have a fever", and so forth, and these should be used to activate the proper KB entities which are used to encode or control the background application logic. In particular, symptom-checking dialogue-systems (SCDSs) have attracted considerable attention due to their promise of low-cost and continuous availability and many academic [5,6,12,18] and industrial systems are also starting to emerge (see also [13] for an overview).

© Springer Nature Switzerland AG 2019

All previous approaches assume that users express their requests in a precise and clear way like "I have a stomach ache", "I want to book a flight", "I want a loan", "I am looking for a thriller movie", from which the relevant terms ("stomach ache", "book flight", etc.) can be extracted using Machine Learning or rule-based slot-filling techniques. However, this assumption fails in applications where users need to convey complex meaning to the service. For example, in a symptom-checking scenario [13] user input may often be highly vague like "I have a pain" in which case several entities in the KB may be relevant like "Abdominal Pain", "Low Back Pain", and more, or be highly colloquial like "I feel my head will explode". In all these cases it is impossible to match user input to the right entities in the inference model. In addition, there is usually a gap between the formal medical language encountered in medical KBs and the terms used by users. For example, a symptom-checker may contain nodes that represent the medical entities "Periumbilical pain" and "sputum in throat", however, users will never use such formal language to report their symptoms.

The above issue is partially solved by using information retrieval techniques like full-text search or sentence embeddings which can retrieve the top-k most "similar" KB entities and then ask the user to select from them, an approach that is actually followed by many free as well as commercial SCDSs. Unfortunately, this approach suffers from several drawbacks. First, it is clearly not user friendly and, second, especially in the symptom-checking scenario users may still find it difficult to browse through the list and understand the differences of the (possibly similar in many cases) symptoms. Third, there is clearly a limit to the number of candidate entities that users can browse, which drops even more if we consider speech-based dialogue systems where the list needs to be read to them.

To address the above issues we designed a framework and algorithm that can be used to "guide" users into associating to their initial query an entity from a pre-defined set of target entities, that most closely matches their intention. First, an initial "small" subset of the target entities is extracted using the hierarchy of the KB together with statistical techniques like embeddings. Second, the properties of these candidates in the KB are used to group them into categories. These categories are then used to ask the user specific questions. For instance, in our running example the system could ask the user "Where is your problem located?" with potential answers "In eye", "On Leg", etc. The effectiveness of the grouping algorithm crucially depends on the number of properties that the target entities share and given the well-known underspecification of KBs [9] this step is not expected to behave well in practice. For this reason we propose an entity enrichment step that uses information extraction techniques and a custom scoring model to prioritise the verification of the newly extracted properties.

Our approach combines in a non-trivial way techniques from guided navigation [10,17] and dialogue systems and also extends previous works to mapping keywords to KBs [16] in an attempt to set a framework for understanding vague and imprecise user queries in a user-friendly and interactive way. Moreover, at a technical level it combines various technologies like ontologies (their hierarchical structure and domain/range constraints), statistical techniques (sentence embeddings and triple scoring), and information extraction (text annotation and some simple triple extraction).

Differently to typical approaches to dialogue-systems our approach does not assume a fixed set of frames with a pre-defined set of slots that need to be filled [19]. In contrast, the target set of entities may contain highly diverse elements and the user query may match any subset of them. Hence, the algorithm is highly dynamic and modular and is able to handle highly diverse and broad domains like symptom-checking, the complexity of which has been noted in the past [15]. Moreover, our approach is largely unsupervised as does not depend on any pre-existing corpora of sample dialogues [15,20] or user queries from logs where a mapping from user text to KB entities can be learned.

Compared to guided navigation and faceted search, our approach is implemented as a short dialogue that presents one question at a time and has to prioritise which question to ask first. In contrast, faceted navigation is prevalently click-based and all facets with result counts and current candidates are presented to the user. Moreover, in our case the set of property-values over which the grouping is applied is not completely available a priori but has to be obtained by our enrichment process.

Our work was motivated by a real-world industrial strength use case in Babylon Health[1]. Babylon Health is a digital healthcare provider and one of its services is an SCDS that is implemented via a Probabilistic Graph Model (PGM) [12]. Users of chatbot enter text like "My stomach hurts" and subsequently the relevant nodes (symptoms) in the PGM need to be activated in order to proceed with symptom-checking. These nodes are encoded using concepts from a medical KB [14] and can either be simple like Fever or also complex like $\mathsf{AnteriorKneeTenderness}$. Babylon's SCDS is experiencing a drop-off when users enter text that is vague or simply does not match complex symptoms. We used our framework to build a prototype system with the aim of reducing Babylon's drop-off. We applied our entity enrichment step on the PGM entities, associated actual questions to each of our concept categories and conducted a preliminary evaluation obtaining encouraging results.

2 Preliminaries

For a set of tuples $\mathsf{tup} = \{\langle k_1, v_1\rangle, \langle k_2, v_2\rangle, ..., \langle k_n, v_n\rangle\}$, the *projection* of tup on the first argument is defined as $\pi_1\mathsf{tup} = \{k_1, k_2, ..., k_n\}$; dually we have $\pi_2\mathsf{tup} = \{v_1, v_2, ..., v_n\}$. A map is a collection of *key/value* pairs. We allow for *semi-structured* maps, that is, maps where the values of different keys may be of a different type. For M a map and k some key we use the notation $\mathsf{M}.k$ to denote the value associated with k. If no key k exists, then $\mathsf{M}.k := v$ means that a new key k is added to the map and its value is set to v.

[1] https://www.babylonhealth.com/.

2.1 Knowledge Bases

Let **C** and **R** be countably disjoint sets of *concepts* and *properties*. A Knowledge Base (KB) is a tuple $\langle \mathcal{K}, \mathbb{T}, \mu, \rho, \delta \rangle$, where $\mathcal{K}$ is a set of *subject*, *property*, *object* triples of the form $\langle s\ p\ o \rangle$ like in the RDF standard[2], $\mathbb{T}$ is a subset of concepts from $\mathcal{K}$ called *semantic types* (stys), μ is a mapping from every concept in $\mathcal{K}$ to a non-empty subset of $\mathbb{T}$ and both ρ and δ are mappings from each $R \in \mathbf{R}$ to a possibly empty subset of **C**. In addition, we assume that every concept C is associated with a preferred label and a user-friendly label both of which are unique. We use the notation $C.\ell$ and $C.\mathsf{lay}$ to refer to these two labels and we also use the notation $C.p$ to refer to the set $\{C' \mid \langle C\ p\ C' \rangle \in \mathcal{K}\}$. Finally, for a concept C we also use the function $\mu^+(C)$ to denote the set $\mu(C) \cup \bigcup_{C' \in C.p} \mu(C')$.

Intuitively, stys denote general/abstract categories of interest in the KB and are used to group other concepts while ρ and δ define the range and domains of properties. For example, in a medical KB we can have stys like Disease, Drug, BodyPart and the like, and then we can have that Malaria has sty Disease. This information can also be encoded within $\mathcal{K}$ using triples of the form $\langle \mathsf{Disease}\ \mathsf{is\_sty}\ \mathit{true} \rangle$ and $\langle \mathsf{Malaria}\ \mathsf{has\_sty}\ \mathsf{Disease} \rangle$. Moreover, for concept MyocardialInfarction we can have $\mathsf{MyocardialInfarction}.\ell =$"Myocardial infarction" and MyocardialInfarction.lay = "Heart attack". A key property that we use from the RDF standard is subClassOf ($\sqsubseteq$ for short) that can be used to specify that the subject of a triple implies the object, e.g., $\langle \mathsf{VivaxMalaria}\ \mathsf{subClassOf}\ \mathsf{Malaria} \rangle$.

We say that C is *subsumed by* D w.r.t. a KB $\mathcal{K}$ if $\mathcal{K} \models \langle C\ \mathsf{subClassOf}\ D \rangle$ where $\models$ is entailment under the standard FO-semantics. For simplicity and without loss of generality we assume that μ is closed under subClassOf, that is, if $C \in \mu(A)$ and C' is some sty s.t. $\mathcal{K} \models \langle C\ \mathsf{subClassOf}\ C' \rangle$, then $C' \in \mu(A)$.

For efficient storage and querying we assume that the KB is loaded to a triple-store employed with (at least) RDFS forward-chaining reasoning. Forward-chaining implies that inferences under the RDFS-semantics are materialised during loading. Hence, if $\mathcal{K} \models \langle C\ \mathsf{subClassOf}\ D \rangle$ under the RDFS-semantics then $\langle C\ \mathsf{subClassOf}\ D \rangle \in \mathcal{K}$.

2.2 Problem Statement

The problem we study is the following: given a subset of concepts TargetCons defined in a KB $\mathcal{K}$ (target concepts or concepts of interest) and a text query Q, provide mechanisms to "guide" the user to a *single* $C \in$ TargetCons that expresses best the intention of Q. This problem is important especially in cases that TargetCons contains concepts representing complex real-world entities like AmenorrheaAfterMenopause, AcuteDyspnoea, AnteriorKneeTenderness, which are unlikely that a user can express with Q. Restricting the problem to single concepts is motivated by the fact that systems like virtual assistants and task-oriented dialogue systems usually require a single entity (frame) to be activated in order to proceed with their application logic.

[2] https://www.w3.org/RDF/.

3 An Interactive Search Approach

Most previous works to task-oriented dialogue systems and virtual assistants are focused on fairly precise tasks and domains like flight booking, bank transfer, home automation, and more. However, as we move towards more complex problems and domains the developed systems would need to be able to deal with vague or imprecise user text.

Example 1. Consider a user that intends to use a symptom-checking dialogue system (SCDS). In such a scenario a user can enter text like $Q =$"i have a rash". Such a statement is quite vague and an SCDS is likely to contain more specific symptoms like $\mathsf{CircularRash}$, $\mathsf{RashInAbdomen}$, $\mathsf{RashInArm}$, and $\mathsf{BumpyRash}$, all of which are relevant to the user text. Even when people visit clinics, doctors usually need to ask a series of questions about the nature of the reported symptom, like its location, its onset, severity, and more, in order to understand the patient conditions better. ♢

A first challenge in the above scenario is to determine an initial and highly relevant set of concepts from the set of concepts that the dialogue system "understands" ($\mathsf{TargetCons}$). Several different alternatives can be considered for this step. A popular approach which is actually used in some SCDS is by using sentence embeddings [1]. Sentence embeddings can be used to map text to a vector space with the property that (semantically) similar text is clustered closely in the space. In our scenario, all labels of the entities in $\mathsf{TargetCons}$ as well as a given user query can be embedded in the vector space and then the top-k closest (w.r.t. cosine similarity) vectors of entities in $\mathsf{TargetCons}$ can be returned. For the above two operations we assume two functions $\mathsf{vectorize}$ and sim. The former takes as input some text ℓ and returns a vector in some vector space while the latter is the similarity between two vectors. Alternatively, one could use text annotation [4] in combination with some form of KB-based reasoning.

Example 2. Consider Example 1, the input sentence Q and the concepts of the SCDS mentioned. Assume also some text annotator that annotates text with entities from some medical KB $\mathcal{K}$ like SNOMED CT. When applied to Q the annotator will return concept $C := \mathsf{Rash}$. It is expected that in SNOMED CT, concept C is somehow semantically related to the symptoms and conditions of SCDS that are potentially relevant to the patient condition, e.g., that $\mathsf{RashInAbdomen}$ is a sub-concept of C.

Relevance between two concepts can be defined in a strong way as "all those $D \in \mathsf{TargetCons}$ such that $\mathcal{K} \models D \sqsubseteq C$" or maybe also in a more loose way as, "all $D \in \mathsf{TargetCons}$ s.t. some path of triples from D to C exists in $\mathcal{K}$". ♢

The above let us to the development of method GenerateCandidates that is depicted in Algorithm 1. The method takes as input some text, a set of concepts of interest, a positive integer k that controls the number of candidates to be considered by the embedding approach, and a set of stys that can be used for additional filtering. The algorithm internally uses a text annotator and a

Algorithm 1. $\mathsf{GenerateCandidates}_{\mathcal{K}}(\mathsf{txt}, \mathsf{TargetCons}, k, \mathsf{styList})$

```
1: $\mathsf{txtAnn} := \mathsf{AnnotateText}_{\mathcal{K}}(\mathsf{txt})$
2: $\mathsf{CandCons} := \{C \mid \langle C \sqsubseteq A\rangle \in \mathcal{K}, A \in \mathsf{txtAnn}, C \in \mathsf{TargetCons}\}$
3: if $\mathsf{CandCons} == \emptyset$ then
4:     $\mathsf{ConsWithWeight} := \{\langle C, \mathsf{sim}(\mathsf{vectorize}(C.\ell), \mathsf{vectorize}(\mathsf{txt}))\rangle \mid C \in \mathsf{TargetCons}\}$
5:     $S_1 := \bigcup_{A \in \mathsf{txtAnn}} \mu(A)$
6:     $\mathsf{ConsWithWeight} := \{\langle C, n\rangle \in \mathsf{ConsWithWeight} \mid S_1 \cap \mu^+(C) \neq \emptyset\}$
7:     $\mathsf{CandCons} := \pi_1 \mathsf{top}(k, \mathsf{ConsWithWeight})$
8: end if
9: $\mathsf{CandCons} := \{C \in \mathsf{CandCons} \mid \mu^+(C) \cap \mathsf{styList} \neq \emptyset\}$
10: return CandCons
```

Knowledge Base ($\mathcal{K}$) which the annotator uses to link text phrases to concepts. In order to abstract from implementation details of different annotators we define the following function.

Definition 1. *Function* $\mathsf{AnnotateText}_{\mathcal{K}}$ *takes as input a text* txt *and returns a set of concepts* $\{C_1, ..., C_n\}$ *from* $\mathcal{K}$ *such that for every* C_i *some substring* str *of* txt *exists such that* $\mathsf{str\text{-}sim}(str, C_i.\ell) \geq thr$ *where* $\mathsf{str\text{-}sim}$ *is some similarity function and* thr *some threshold.*

The algorithm first uses the semantic approach described in Example 2 since this is expected to be more selective and with higher precision (fewer false positives). If no candidates can be computed, then the more "relaxed" embedding approach is employed. Finally, candidates computed by either method can be further filtered according to a set of stys of interest (line 9). In the evaluation section we analyse the effectiveness of various combinations of these methods.

After computing an initial list of candidates the most relevant one of those needs to be selected and passed to the dialogue-system. In a naive approach the user can be presented with the full list of candidates and asked to choose, an approach that is actually followed in many free on-line as well as commercial SCDSs. Unfortunately, this approach is not user-friendly and still users may find it hard to pick the correct entities if their differences are not clear to them or more than one seem relevant. Even worse, this approach cannot be implemented in spoken dialogue-systems where a potentially long list of candidates needs to be read to the user. Ideally we need a way to group the candidates according to some set of properties and ask the user which value of that properties are most closely related to the condition they report.

Example 3. Consider again Example 1. As can be seen many of the potentially relevant concepts are about some kind of "Rash" which is further specialised with either the body location where it manifests ("Abdomen", "Arm", etc.) or its appearance ("Circular", "Bumpy"). We can assume that these differences are also explicated in a medical KB using appropriate triples like the following ones:

$\langle\mathsf{RashInAbdomen\ location\ Abdomen}\rangle$	$\langle\mathsf{RashInArm\ location\ Arm}\rangle$
$\langle\mathsf{CircularRash\ shape\ Circular}\rangle$	$\langle\mathsf{BumpyRash\ shape\ Bumpy}\rangle$

Algorithm 2. $\mathsf{CandidateSearch}_{\mathcal{K}}(\mathsf{CandCons}, \mathsf{styList}, n)$

```
    Input: A set of concepts and stys styList from the KB and a positive integer.
 1: while $|\mathsf{CandCons}| \geq n$ do
 2:     Create a map styToCands such that for every $sty \in \mathsf{styList}$ we have
            $\mathsf{styToCands}.sty := \{\langle C, C'\rangle \mid C \in \mathsf{CandCons}, C' \in C.p, sty \in \mu(C')\}$
 3:     Let $sty_m$ be some key in styToCands with the most values.
 4:     if $|\mathsf{styToCands}.sty_m| < n$ then break
 5:     $\mathsf{ansCons} := \mathsf{askUser}(\mathsf{styToCands}.sty_m, sty_m)$
 6:     if $\mathsf{ansCons} == \emptyset$ then
 7:         then $\mathsf{CandCons} := \mathsf{CandCons} \setminus \pi_1 \mathsf{styToCands}.sty_m$
 8:     else
 9:         $\mathsf{CandCons} := \mathsf{ansCons}$
10:     end if
11: end while
12: if $|\mathsf{CandCons}| > 1$ then
13:     $\mathsf{CandCons} := \mathsf{askUser}(\{\langle C, \bot\rangle \mid C \in \mathsf{CandCons}\}, \mathsf{null})$
14: end if
15: return CandCons
```

The semantic types of the objects in these triples (e.g., BodyPart for Abdomen and Arm and Shape for Circular and Bumby) provide the categories over which grouping can be performed and questions asked. For example, for the above candidates the questions that arise are "Where is your Rash?" and "How does its shape look like?". The potential answers for the first question are "Abdomen", "Arm", or "None of the above", and the last answer includes candidates CircularRash and BumpyRash that are not connected in the KB with any body part. ♢

Based on the above intuition we designed Algorithm 2. The algorithm takes as input a set of candidate concepts (possibly computed using Algorithm 1), a set of stys styList over which grouping is done and a positive integer which is used to control the grouping process. The algorithm enters a loop where it groups the current set of candidates according to the semantic types of concepts to which candidates are connected in the KB and builds pairs of the form $\langle C, C'\rangle$ where $C \in \mathsf{CandCons}$ and C' is a concept that C points to. A pair of this form is built because the label of C' will be used as an answer value for the question that would be generated. Subsequently, the algorithm selects the group that contains the most candidates and asks a question related to the type of that group. Our strategy of selecting the group with the most members is similar to selecting facets with the largest specificity [17], however, different strategies or more complex models can also be used [11].

Generating the question to be asked for the selected group as well as the potential answer values is done using function askUser which is discussed in Sect. 4. The possible values of the answers also include a "None of the above" answer in which case this function returns the empty set. If this is the answer of the user then all candidates in the group are removed (line 7). The algorithm stops the grouping process and exits the while loop when the set of candidates has

Algorithm 3. askUser(ConceptPairs, sty)

```
    Input: A set of pairs of concepts and a semantic type ($sty$)
 1: if $sty$ == null then
 2:     println "Which of the following?"
 3:     for $\langle C, \_ \rangle \in$ ConceptPairs do
 4:         println "\t " + $C$.lay
 5:     end for
 6:     print "\t None"
 7:     $ans$ := read answer from console
 8:     return $\{C \mid C.\mathsf{lay} = ans\}$
 9: else
10:     println fetchQuery($sty$)
11:     for $\langle C, C' \rangle \in$ ConceptPairs such that $C'$.lay hasn't been printed before do
12:         println "\t " + $C'$.lay
13:     end for
14:     println "\t None"
15:     $ans$ := read answer from console
16:     return $\{C \mid \langle C, C' \rangle \in \mathsf{ConceptPairs}$ with $C'.\mathsf{lay} = ans\}$
17: end if
```

dropped below a threshold n. In this case the set is considered to be sufficiently small that the remaining candidates can be presented to the user to select the most relevant one.

For the concepts of Example 3 the algorithm would create the following two groups:

$$\begin{aligned} &\mathsf{styToCands.BodyPart} &&:= &&\{\langle \mathsf{RashInAbdomen}, \mathsf{Abdomen}\rangle, \langle \mathsf{RashInArm}, \mathsf{Arm}\rangle\} \\ &\mathsf{styToCands.Shape} &&:= &&\{\langle \mathsf{CircularRash}, \mathsf{Circular}\rangle, \langle \mathsf{BumpyRash}, \mathsf{Bumpy}\rangle\} \end{aligned}$$

4 Question Generation

Algorithm 2 generates two types of questions, one that asks users to clarify the value of a specific property of the candidates (line 5) and one that simply prints all candidates and asks users to choose one of them (line 13). The generation of fluent and natural questions is a non-trivial problem and simple but effective template-based shallow generation approaches are usually preferred in real-world applications [6], which we also follow as an initial approach.

Our two types of questions are generated by Algorithm 3 which takes as input a pair of concepts and an sty. If the sty is null then the algorithm proceeds with printing a question of the form "Which of the following?" and then the user-friendly label of each candidate (line 4). Note that since the set of candidates does not contain duplicates and by the assumption of uniqueness of user-friendly labels in the KB these labels are unique.

In case the semantic type is not null a specific query that depends on the semantic type needs to be printed. A question has been assigned to each semantic

Table 1. Questions associated to Semantic Types and potential answer values

Semantic type	Question	Answer values
BodyPart	"Is your problem located in:"	abdomen, leg, finger, ...
NeurologicalFinding	"Do you feel/experience:"	pain, irritation, ...
Severity	"How severe is your problem?"	severe, moderate, mild, ...
BiologicalSubstance	"Do you see:"	urine, mucus, blood, ...
Appearance	"Does it look:"	bruised, scared, clear, ...
Colour		red, black, yellow, ...
Shape		circular, bent, curved, ...
SpatialQualifier	"Is it:"	left, right, top, bottom, ...

type at design time. An excerpt of questions for a symptom-checking scenario as well as some potential answer values are depicted in Table 1. Regarding answer values the user-friendly label of the possible property value concepts are used. In this case duplicates may exist. Consider for example, that the candidate selection step has returned concepts C_1 = HeadInjury and C_2 = HeadPain which we have grouped according to body structure generating pairs $\langle C_1, \mathsf{Head}\rangle$ and $\langle C_2, \mathsf{Head}\rangle$. In that case the answer value for the question "Where is your problem located?" is the same for both concepts (i.e., "Head"). The algorithm takes care to print "Head" only once and if the user selects this as an answer then both concepts C_1 and C_2 would need to be returned by the algorithm.

5 Knowledge Base Enrichment

Algorithm 2 is using the properties of concepts in the KB in order to group the candidate concepts. It is clear that the more properties these concepts have and the more they share them with each other, the more effective these groupings would be. In a different case, groups will mostly contain a single concept and all the other would be in the "None of the above" answer. Unfortunately, several authors have noted the "incompleteness" and "underspecification" of medical KBs [9]. For example, in SNOMED CT concept RecentInjury is not associated with concepts Recent and Injury and SevereAbdominalPain is not linked with concept Severe. The same issue can easily be observed in other KBs like DBpedia [2] where the category ItalianRenaissancePainters is not connected to concepts Painter or ItalianRenaissance.

In order to improve the effectiveness of the grouping strategy all concepts in TargetCons need to be enriched with as many triples as possible. This task can be manual but it would be beneficial if automatic ways could also be employed to assist it. It has been noted that labels of concepts in (biomedical) ontologies are a good source of additional information [3]. For instance, in Example 1 we can immediately see that the label of concept RashInHead implies a link between Rash and Head.

Algorithm 4. $\mathsf{conceptEnrichment}_{\mathcal{K}}(\mathsf{TargetCons}, \mathsf{styList}, thr)$

```
Input: A set of concepts and stys from some KB $\mathcal{K}$ and a real number $thr$
 1: $\mathsf{Inspect} := \emptyset$
 2: for all $C \in \mathsf{TargetCons}$ do
 3:     for all $C' \in \mathsf{AnnotateText}_{\mathcal{K}}(C.\ell)$ such that $\mu(C') \cap \mathsf{styList} \neq \emptyset$ do
 4:         if $\mathsf{score}_{\mathsf{model}}(\langle C\ C' \rangle) \geq thr$ then
 5:             Add $\langle C\ p\ C' \rangle$ to $\mathcal{K}$ for some $p$ with domain in $\mu(C)$ and range in $\mu(C')$
 6:         else
 7:             $\mathsf{Inspect} := \mathsf{Inspect} \cup \{\langle C\ p\ C' \rangle\}$
 8:         end if
 9:     end for
10: end for
```

Building on this idea we designed a semi-automatic pipeline depicted in Algorithm 4 that can be used to extract such information from concept labels. The algorithm takes as input a set of concepts and uses their label to extract triples of the form $\langle C\ p\ C' \rangle$ which is again achieved using text annotation. To control the number of new triples extracted some list of stys of interest ($\mathsf{styList}$) is also passed as a parameter.

Clearly, triples extracted from such an automated pipeline can be erroneous. To assist in the manual verification, techniques to score them and focus validation on the low-scored pairs would be beneficial. Several different methods can be used like KB embedding models [7], training a custom deep NN classifier using label embeddings, or training a traditional classifier using features like n-grams or the dependency-parse tree of concept labels. We have experimented with a custom deep neural classifier which we detail in the evaluation section.

6 Evaluation

We have used our methodology and framework in order to build an interactive system that can be used to help understand vague user text in Babylon Health's SCDS. The current version of Babylon Health's SCDS contains about 2261 symptoms which correspond to a small subset of a much larger medical Knowledge Base [14]. This KB is built by aligning and integrating ontologies like SNOMED CT, NCI, and more [14] and currently contains 1,5m concepts, 173 properties, 1,8 million subsumption axioms, 2,2m labels, 93 semantic types and 34 domain/range axioms. It is stored and queried using RDF4J[3] and saturated using RDFS inference rules.

We collected 265 user text queries from the Babylon SCDS and asked medical doctors to map each of them to the most relevant concept in PGM; we call these concepts the *user intended concepts*. We also prepared another variation on the input text queries where we tried to 'degenerate' them by removing some of the parts of the text in an attempt to make them more vague. For example, if the user text is "I feel pain around my heart" we try to construct text "I feel pain".

[3] http://rdf4j.org/.

Table 2. Counts of PGM concepts with the given semantic types (stys) shown before and after concept enrichment.

sty	Before	After	sty	Before	After
BodyPart	1585	2566	ClinicalFinding	1049	1567
ObservableEntity	728	1102	AbnormalBodyPart	452	712
QualifierValue	31	652	AnatomyQualifier	42	218
Substance	17	139	SpatialQualifier	3	119
Organism	3	16	ClinicalQualifier	2	11

To do so we used sentence embeddings between original input text and labels of PGM concepts that appear in the object position of triples. For example, we can use the triple ⟨Pain findingSite Heart⟩ and the label $\mathsf{Heart}.\ell =$"Heart" to remove the respective text from the above sentence.

6.1 Concept Enrichment

First, we run the enrichment pipeline presented in Sect. 5 in order to extract additional triples for all PGM concepts. To control the process we fixed a list of 10 stys of interest (parameter styList in Algorithm 4). All extracted triples were scored using two different models and evaluated using 240 labelled data. The first method used the RESCAL [7] approach for KB embeddings and this model yielded an AUC of 0.52 on the testing data. The second approach was a three layer Neural Network developed using Keras[4] with two hidden layers of sizes 3 and 5. The input was the concatenation of the sentence embeddings of the PGM node text and the extracted triple. Both embeddings were of size 512, yielding a combined input layer of size 1024. The training and test sets were of sizes 192 and 48, respectively, with the network trained using a binary cross-entropy loss function. Resulting accuracy was 0.84 while AUC was 0.73. Our result is interesting in the sense that even simple custom approaches work better than off-the-shelf involved KB embedding approaches.

The enrichment process increased the triples of the 2261 symptoms from 3920 to 7102. The breakdown of these triples per sty before and after enrichment is depicted in Table 2. From those numbers we can conclude that, in spite of the enrichment, some stys will most likely not be very effective in grouping candidates as they do not appear often in the PGM nodes (e.g., all below a count of 100). In Sect. 6.4, we will investigate further which of those stys were actually the most frequently used ones when we run the main algorithm on input user text.

[4] https://keras.io/.

6.2 Evaluating the Effect of *k* in Selecting the Top-*k* Candidates

A crucial property of the candidate selection algorithm is the number of top-k candidates that we need to compute in order for this set to always include the user intended concept. To evaluate the sensitivity of the embedding approach on the selection of k we varied the value of k starting with $k = 5$ and using increment of 5 we established in how many cases (out of the 265) the intended user concept was in the top-k candidates. The results are presented in Table 3. As can be seen for $k = 30$ the embedder was able to include the user intended concept in 257 (97%) and 169 (64%) cases in the two different test query sets in contrast to 245 (92%) and 117 (44%) for $k = 10$ which is the usual one. As can be seen for vague (degenerated) queries going above the fairly standard top-10 is highly beneficial.

Table 3. Number of correct concepts distributed in various ranks according to k; e.g., (5,10] denotes how many intended concepts appear between the top-5 and top-10.

k value	[1, 5]	(5, 10]	(10, 15]	(15, 20]	(20, 25]	(25, 30]	[1, 30]	(30, 256]
Degenerated	91	26	20	14	10	8	169	96 (36%)
Full Queries	229	16	5	4	1	2	257	8 (3%)

6.3 Evaluating Candidate Selection and Depth of Correct Answer

We evaluated two variations of the candidate selection algorithm (Algorithm 1). The first implements Algorithm 1 as presented, that is, first uses subClassOf traversal on the KB (line 2 of Algorithm 1) and if this step returns an empty set then it falls back to embeddings, while the second only uses the embedding approach. In both cases k was set to 30 for the embedder.

Table 4 shows the number of times that the correct answer was included in the candidate set computed by the two algorithms in the two different sets of queries. We further broke down the first approach and measured in how many cases the KB descendant approach returned a non-empty set of candidates and in how many of them the intended concept was within the candidates. As can be seen there are quite a few cases where the KB approach returns an empty set (68 in the first and 18 in the second set of queries) and the fall-back needs to be employed. Moreover, even if a non-empty set is returned, in quite a few cases this set does not contain the intended user query. This is because the current implementation of checking descendants in the KB is semantically "very strict" compared to the flexible way in which a query can be formed in text. For example, the behaviour of this approach is considerably different if the input text is "Blood in stool" or "Bleeding". In this case the annotator associates KB concepts of even different sty. In contrast the embedder works considerably better due to the fact of loosely capturing the semantics and similarities between concepts.

Table 4. Frequency where correct answer was included in the candidate set by each approach (correct cases/cases that approach was employed).

Input text	KB+Embedder		Embedder (only)
	KB	Embedder	
Degenerated	90/197	38/68	169/265
Full queries	181/247	15/18	257/265

Table 5. Number of cases and questions required to reach the intended concept.

Input text	KB+Embedder		Embedder (only)
	KB	Embedder	
Degenerate	48(1), 31(2), 8(3), 3(4)	10(1), 25(2), 3(3)	68(1), 67(2), 32(3), 2(4)
Full queries	115(1), 53(2), 12(3), 1(4)	3(1), 10(2), 2(3)	50(1), 127(2), 71(3), 9(4)

After candidate selection we want to evaluate how many questions would be required before Algorithm 2 returned the user intended concept. This is similar to the number of *clicks* (scan effort) in faceted search evaluation. We only consider the cases that the candidate selection approaches did manage to return the user intended concept in the set of candidates. The results are depicted in Table 4; the format X (Y) means that in X number of cases the intended user concept is reachable after Y questions. As can be seen most concepts are reachable after one or two questions and all concepts are reachable at most after four (Table 5).

6.4 Further Evaluation of Grouping Algorithm

Table 6 (left) shows which stys were used to group the set of candidates in any of the user queries of our test sets. Results were very similar in all variations we ran. In 51% of tests, the final question of the algorithm did not have an sty to distinguish the correct answer from the others (although previous question would have), so the answers were collapsed to a generic question.

Ideally the groups created by the algorithm should neither be very small (which leads to too many questions) nor too large (which leads to few question with too many answers). We provide summary statistics for these results in Table 6 (right). As expected degenerated text queries result in larger answer sets since they are more vague. Another aspect to note is the smaller answer sets produced with the annotation filter compared to the embedding filter. This comes as a trade off against the lower recall of the annotation filter, which we can interpret as a more homogeneous set of candidates being returned by the stricter filter.

Table 6. Left: stys that appeared in evaluation (% of total tests); Right: Number of answers per question (median; mean)

Anatomical structure	75%
Qualifier value	36%
Observable entity	11%
Morphologically altered Str.	6.3%

	KB+Embedder		**Embedder**
Input Text	KB	Embedder	
Degenerated	4; 26.2	15; 12.3	13; 11.8
Full Queries	4; 4.3	13; 11.0	6; 9.8

7 Conclusions

In the current paper we studied the problem of interpreting and understanding (possibly vague and imprecise) user queries over KBs. This problem is highly relevant in applications like ontology-based dialogue systems and virtual assistants where user requests need to be mapped to KB entities that activate some background service. First, users may not have prior knowledge of the content and entities that the service supports, second, they may not know how to accurately express their request and, third, in applications like symptom-checking the language they use may be substantially different compared to the formal medical language used to construct the service by doctors. Hence, we may either be unable to match a user query to any entity in the KB (technically involved domains) or more than one entities may be relevant (vague input queries). To be able to associate the "intended" entity to the user query we developed a framework which can construct on the fly a small dialogue that asks the user few "clarification" questions in an attempt to "activate" the proper entity. Semantic and ML-based techniques were proposed for selecting an initial (small) set of candidates and then an algorithm for grouping those candidates according to their properties in the KB was designed. The "most relevant" group is then selected according to some cost model and the question associated to the group is printed. To improve the effectiveness of the approach and overcome underspecification issues of KBs an information extraction-based enrichment pipeline is proposed and various scoring models were used to assess the soundness of the extracted information. This process should be complemented with manual curation in order to build questions that are more user friendly and fluent.

We implemented our framework and evaluated it using real world data obtained from the symptom-checking service provided by Babylon Health. We evaluated the two different approaches for obtaining candidates, the sensitivity of the approach with respect to top-k retrieval, and the number of questions and size of answer values constructed in the small dialogues. Although largely motivated by symptom-checking, we envision that our work is relevant and useful in any domain and can greatly contribute towards building more user-friendly ontology-based intelligent systems. It can be used to form the basis for providing a bridge between initial vague, imprecise, or noisy user text and the services implemented by an ontology-based dialogue system or for accessing large KBs.

References

1. Arora, S., Liang, Y., Ma, T.: A simple but tough-to-beat baseline for sentence embeddings. In: Proceedings of the 5th International Conference on Learning Representations (ICLR), pp. 1–14 (2017)
2. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: a nucleus for a web of open data. In: Aberer, K., et al. (eds.) ASWC/ISWC -2007. LNCS, vol. 4825, pp. 722–735. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76298-0_52
3. Fernandez-Breis, J.T., Iannone, L., Palmisano, I., Rector, A.L., Stevens, R.: Enriching the gene ontology via the dissection of labels using the ontology pre-processor language. In: Cimiano, P., Pinto, H.S. (eds.) EKAW 2010. LNCS (LNAI), vol. 6317, pp. 59–73. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16438-5_5
4. Hachey, B., Radford, W., Nothman, J., Honnibal, M., Curran, J.R.: Evaluating entity linking with wikipedia. Artif. Intell. **194**, 130–150 (2013)
5. Kao, H., Tang, K., Chang, E.Y.: Context-aware symptom checking for disease diagnosis using hierarchical reinforcement learning. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI) (2018)
6. Milward, D., Beveridge, M.: Ontology-based dialogue systems. In: 3rd Workshop on Knowledge and Reasoning in Practical Dialogue Systems (2003)
7. Nickel, M., Murphy, K., Tresp, V., Gabrilovich, E.: A review of relational machine learning for knowledge graphs. Proc. IEEE **104**(1), 11–33 (2016)
8. Oramas, S., Ostuni, V.C., Noia, T.D., Serra, X., Sciascio, E.D.: Sound and music recommendation with knowledge graphs. ACM Trans. Intell. Syst. Technol. **8**(2), 21:1–21:21 (2016)
9. Pacheco, E.J., Stenzhorn, H., Nohama, P., Paetzold, J., Schulz, S.: Detecting underspecification in SNOMED CT concept definitions through natural language processing. In: Proceedings of the American Medical Informatics Association Annual Symposium (2009)
10. Papadakos, P., Armenatzoglou, N., Kopidaki, S., Tzitzikas, Y.: On exploiting static and dynamically mined metadata for exploratory web searching. Knowl. Inf. Syst. **30**(3), 493–525 (2012)
11. Qarabaqi, B., Riedewald, M.: User-driven refinement of imprecise queries. In: IEEE 30th International Conference on Data Engineering ICDE, pp. 916–927 (2014)
12. Razzaki, S., et al.: A comparative study of artificial intelligence and human doctors for the purpose of triage and diagnosis. CoRR abs/1806.10698 (2018)
13. Semigran, H.L., Linder, J.A., Gidengil, C., Mehrotra, A.: Evaluation of symptom checkers for self diagnosis and triage: audit study. BMJ **351**, h3480 (2015)
14. Stoilos, G., Geleta, D., Shamdasani, J., Khodadadi, M.: A novel approach and practical algorithms for ontology integration. In: Vrandečić, D., et al. (eds.) ISWC 2018. LNCS, vol. 11136, pp. 458–476. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00671-6_27
15. Tang, K., Kao, H., Chou, J., Chang, E.: Inquire and diagnose: neural symptom checking ensemble using deep reinforcement learning. In: 3rd Deep Reinforcement Learning Workshop (2016)
16. Tran, T., Cimiano, P., Rudolph, S., Studer, R.: Ontology-based interpretation of keywords for semantic search. In: Aberer, k, et al. (eds.) ASWC/ISWC -2007. LNCS, vol. 4825, pp. 523–536. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76298-0_38

17. Vandic, D., Aanen, S., Frasincar, F., Kaymak, U.: Dynamic facet ordering for faceted product search engines. IEEE Trans. Knowl. Data Eng. **29**(5), 1004–1016 (2017)
18. Wei, Z., et al.: Task-oriented dialogue system for automatic diagnosis. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, pp. 201–207. ACL (2018)
19. Wessel, M., Acharya, G., Carpenter, J., Yin, M.: OntoVPA - an ontology-based dialogue management system for virtual personal assistants. In: 8th International Workshop on Spoken Dialog Systems, IWSDS, pp. 219–233 (2017)
20. Yan, Z., Duan, N., Chen, P., Zhou, M., Zhou, J., Li, Z.: Building task-oriented dialogue systems for online shopping. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI), pp. 4618–4626 (2017)