Leveraging Community-built Knowledge for Type Coercion in Question Answering

Aditya Kalyanpur, J William Murdock, James Fan and Chris Welty

IBM Research, 19 Skyline Drive, Hawthorne NY 10532
{adityakal, murdockj, fanj, welty}@us.ibm.com

Abstract. Watson, the winner of the Jeopardy! challenge, is a state-of-the-art open-domain Question Answering system that tackles the fundamental issue of answer typing by using a novel type coercion (TyCor) framework, where candidate answers are initially produced without considering type information, and subsequent stages check whether the candidate can be *coerced* into the expected answer type. In this paper, we provide a high-level overview of the TyCor framework and discuss how it is integrated in Watson, focusing on and evaluating three TyCor components that leverage the community built semi-structured and structured knowledge resources -- DBpedia (in conjunction with the YAGO ontology), Wikipedia Categories and Lists. These resources complement each other well in terms of precision and granularity of type information, and through links to Wikipedia, provide coverage for a large set of instances.

Keywords: Question Answering, Type Checking, Ontologies, Linked Data

1 Introduction

Typing, the task of recognizing whether a given entity is a member of a given class, is a fundamental problem in many AI areas. We focus on the typing problem where both the entity and type are expressed lexically (as strings), which is typically the case in open-domain Question Answering (QA), and present a solution that leverages community-built web knowledge resources.

Many traditional QA systems have relied on a notion of *Predictive Annotation* 10 in which a fixed set of expected answer types are identified through manual analysis of a domain, and a background corpus is automatically annotated with possible mentions of these types before answering questions. These systems then analyze incoming questions for the expected *answer type*, mapping it into the fixed set used to annotate the corpus, and restrict candidate answers retrieved from the corpus to those that match this answer type using semantic search (IR search augmented with search for words tagged with some type).

This approach suffers from several problems. First, restricting the answer types to a fixed and typically small set of concepts makes the QA system brittle and narrow in its applicability and scope. Such a closed-typing approach does not work well when answer types in questions span a broad range of topics, are expressed using a variety of lexical expressions. Second, the QA system performance is highly dependent on the precision and recall of the predictive annotation software used, which acts as a candidate selection filter.

In contrast to this type-and-generate approach, we consider a *generate and type* framework, in which candidate answers are initially produced without use of answer type information, and subsequent stages check whether the candidate answer's type can be *coerced* into the Lexical Answer Type (LAT) of the question. The framework is based loosely on the notion of *Type Coercion* 11 (TyCor). The most notable aspects of the approach are: it does not rely on a fixed type system, however it does not discard one when available and useful; it is a multi-strategy and multi-source approach, gathering and evaluating evidence in a generative way rather than a predictive one; it is not part of candidate generation, rather it is simply another way of analyzing and scoring candidate answers; it is not a hard filter, producing for each candidate answer a probability that it is (or is not) of the right type.

In this paper, we provide a high-level architecture of the TyCor framework and discuss how it is integrated into Watson. We present three TyCor components that leverage community built semi-structured and structured knowledge resources, including DBpedia 2 (in conjunction with YAGO 13), Wikipedia Categories and Lists.

2 Background: Open Domain Question Answering

Watson is a QA system capable of rivaling expert human performance on answering open-domain questions on the challenging TV quiz show Jeopardy!, whose questions cover a wide range of topics and are expressed using rich, complex natural language expressions. The typing problem for historical Jeopardy! questions is not trivial, as our analysis reveals that nearly any word in the English language can be used as an answer type in Jeopardy! questions, as shown in Table 1.

Given this variability, one of the intuitive problems with predictive annotation is that we cannot reliably predict what types there are going to be and what their instances are. We need to be open and flexible

about types, treating them as a property of a question and answer combined. In other words, instead of finding candidates of the right type, we want to find candidates (in some way) and judge whether each one is of the right type by examining it in context with the answer type from the question. Furthermore, we need to accommodate as many sources as possible that reflect the same descriptive diversity as these questions. Furthermore, by not relying on a fixed type system, it is imperative to develop a system that has wide coverage for *rare* types.

Jeopardy! Question	Answer
Invented in the 1500s to speed up the game, this *maneuver* involves 2 pieces of the same color	Castling
The first known airmail service took place in Paris in 1870 by this *conveyance*	Hot-air balloon
When hit by electrons, a phosphor gives off electromagnetic energy in this *form*	Light
A 1968 *scarefest*: The title character made it a family of 3 for the Woodhouses	Rosemary's Baby

Table 1: Sample Jeopardy! Questions showing variability in answer types

2.1 DeepQA

Underlying the Watson system is DeepQA, a massively parallel probabilistic evidence-based architecture designed to answer open domain natural language questions. It consists of the following major stages (more details can be found in 4):

Question Analysis: The first stage of processing performs a detailed analysis to identify key characteristics of the question (such as focus, lexical answer type, question class, etc.) used by later stages. The focus is the part of the question that refers to the answer, and typically encompasses the string representing the lexical answer type (LAT). The system employs various lexico-syntactic rules for focus and LAT detection, and also uses a statistical machine-learning model to refine the LAT(s). Like all parts of our system, LAT detection includes a confidence, and all type scores are combined with LAT confidence

Hypothesis (Candidate) Generation: For the candidate generation step, the system issues queries derived from question analysis to search its background information (corpora, data- and knowledge-bases) for

relevant content, and uses a variety of candidate generators to produce a list of potential answers.

Hypothesis and Evidence Scoring: Answer scoring is the step in which all candidates, regardless of how they were generated, are evaluated. During this phase, many different algorithms and sources are use to collect and score evidence for each candidate answer. Type information is just one kind of evidence that is used for scoring, other dimensions of evidence include temporal/spatial constraints, n-grams, popularity, source-reliability, skip-bigrams, substitutability, etc.

Candidate Ranking: Finally, machine-learning models are used to weigh the analyzed evidence and rank the answer candidates and produce a confidence. The models generate a confidence that each answer candidate is the correct answer to the given question, and the system answers with the top-ranked candidate. The system can also choose to refrain from answering if it has a low confidence in all of its candidates.

2.2 Type Coercion (TyCor)

The TyCor framework is part of Hypothesis and Evidence scoring, and consists of a suite of answer scoring components that each take a Lexical Answer Type (LAT) and a candidate answer, and return a probability that the candidate's type is the LAT. Each TyCor component uses a source of typing information and performs four steps, each of which is capable of error that impacts its confidence:

Entity Disambiguation and Matching (EDM): The most obvious, and most error-prone, step in using an existing source of typing information is to find the entity in that source that corresponds to the candidate answer. Since the candidate is just a string, this step must account for both polysemy (the same name may refer to many entities) and synonymy (the same entity may have multiple names). Each source may require its own special EDM implementations that exploit properties of the source, for example DBpedia encodes useful naming information in the entity URI. EDM implementations typically try to use some context for the answer, but in purely structured sources this context may be difficult to exploit.

Predicate Disambiguation and Matching (PDM): Similar to EDM, the type in the source that corresponds to the LAT must be found. In some sources this is the same algorithm as EDM, in others, type look-

ing requires special treatment. In a few, especially those using unstructured information as a source, the PDM step just returns the LAT itself. In type-and-generate, this step corresponds to producing a semantic answer type (SAT) from the question. PDM corresponds strongly to notions of word sense disambiguation with respect to a specific source.

Type Retrieval (TR): After EDM, the types of the retrieved entity must be themselves be retrieved. For some TyCors, like those using structured sources, this step exercises the primary function of the source and is simple. In others, like unstructured sources, this may require parsing or other semantic processing of some small snippet of natural language.

Type Alignment: The results of the PDM and TR steps must then be compared to determine the degree of match. In sources containing e.g. a type taxonomy, this includes checking the taxonomy for subsumption, disjointness, etc. For other sources, alignments utilize resources like WordNet for finding synonyms, hypernyms, etc. between the types.

Each of the steps above generates a score reflecting the accuracy of its operation, taking into account the uncertainty of the entity mapping or information retrieval process. The final score produced by each TyCor component is a combination of the four step scores and the confidence in the LAT.

3 Acquiring Community-built Knowledge for TyCor

We wanted to determine if community-built knowledge resources could be effectively (and cheaply) used to bootstrap a dynamic open-domain typing system, as well as deal with the very long tail of answer types. For this reason, we acquired a broad domain structured knowledge base (DBpedia) and ontology (YAGO), and semi-structured folksonomies with wide topical coverage (Wikipedia Categories and Lists).

3.1 DBpedia and YAGO

The DBpedia knowledge base contains relational information found in the info-boxes of Wikipedia pages. A one-to-one correspondence between all Wikipedia pages and DBpedia entries maps the two resource names (or URIs): e.g., the DBpedia page with URI

`http://dbpedia.org/resource/IBM` corresponds to the Wikipedia page titled "IBM" with relational facts (triples) captured from the infobox.

Additionally, DBpedia has type assertions for many instance objects. The types are assigned from a collection of ontologies, including YAGO, a large taxonomy of more than 100K types. Crucially, the YAGO ontology has mappings to WordNet 7: every YAGO type corresponding to a WordNet concept has the associated 9-digit WordNet sense id appended to its name/id. Thus the YAGO type "*Plant100017222*" links to the WordNet concept plant (living organism), while the type "*Plant103956922*" corresponds to the concept of an industrial plant or factory.

YAGO types are arranged in a hierarchy, and DBpedia instances are often assigned several low-level types corresponding to Wikipedia categories (e.g. "*CompaniesEstablishedIn1896*"). For these, navigation up the YAGO type tree leads to more general and normalized (via sense-encoding) YAGO WN concepts.

These design points of DBpedia and YAGO enable us to obtain precise type information for many instances, given Wikipedia domain coverage and YAGO-WordNet type/sense coverage.

One downside is that the YAGO ontology does not handle mutually exclusive (disjoint) types – ones that do not share instances. For example, Country and Person types are logically disjoint: no given instance can be both a Country and a Person; on the other hand, Painter and Musician are not disjoint. Type disjointness is useful for QA, to rule out candidate answers with types incompatible with the question LAT. For this reason, we decided to add disjointness relations between YAGO types. Given the size of YAGO ontology (> 100K types), manually asserting such relations between all applicable type pairs is infeasible. Instead, we only specify disjointness between prominent top-level types of the YAGO hierarchy, and use a logical reasoning mechanism to propagate the disjointness to lower subclasses. For example, it follows logically that if the *Person* and *GeoPoliticalEntity* are disjoint, then every subclass of *Person* is disjoint with every subclass of *GeoPoliticalEntity* (e.g. *Musician* is disjoint with *Country*). Our additions to the YAGO Type system comprise approximately 200 explicit disjoint relations, which translate through inference to more than 100K disjoint relations.

3.2 Wikipedia-based Folksonomies

For the purposes of this paper, we consider Wikipedia Categories and Lists to be folksonomies. Wikipedia categories are true tags that are applied to articles by Wikipedia users without very much centralized control, and new categories can be invented as desired. The categories have some explicit structure, in that category pages can themselves be put into categories.

One may reasonably argue that Wikipedia lists are not true tags, as they are assigned in a more middle-out and sometimes top-down method than bottom up. Lists are generally created and then populated with articles, and frequently one cannot access the lists an article is in from the article. We ignore this difference and treat Wikipedia lists as the same kind of resource as categories.

4 TyCor Algorithms

In this section, we describe the algorithms underlying the three TyCor components that use YAGO (through DBpedia), Wikipedia Categories and Lists respectively as a source of type information.

4.1 Shared EDM Algorithm

The three TyCors described here share one EDM algorithm, that takes as input the candidate answer string and a corresponding context – the question text and (optionally) a text passage containing the candidate – and returns a ranked list of Wikipedia page URIs that match the candidate, with associated match scores. The match scores are computed based on five heuristics:

- **Direct Contextual Match.** In some cases the Wikipedia URI of the candidate is known, and EDM is not performed. For example, if the candidate answer was generated from the title of a Wikipedia page, or if the candidate mention itself is hyper-linked to another Wikipedia page, we store that information in our candidate answer object and use it as the result of our EDM step with a score of 1.0.
- **Title Match.** When there is an exact string match between the candidate string and the title of a Wikipedia page, the URI is returned with a score of 1.0.
- **Redirect Match.** When the candidate string matches the name of a redirect page, the redirect destination URI is returned with a score of 1.0. There are some noisy

redirects in Wikipedia, e.g. Eliza Doolittle (character) redirects to Pygmalion (play), but in general we have observed the redirects to be reliable.

- **Disambiguation Match.** When the candidate string matches the title of a Wikipedia disambiguation page all the disambiguation URIs are returned with a score of 1/(the number of disambiguations).
- **Anchor-Link Match.** When a candidate string matches one or more anchor text strings in Wikipedia, all the URIs pointed to by those anchors are returned with a score for each based on the conditional probability of the link pointer given the anchor text (ie how often does the anchor text point to the URI).
- **DBpedia name properties.** DBpedia includes over 100 *name properties*, properties whose objects are some form of name string (firstName, lastName, etc). When a candidate string matches one of these, the triple subject is returned with a score of 1/(number of URIs returned)

The EDM algorithm also contains an optional parameter to rank the results based on the *popularity* of the corresponding Wikipedia page, overriding the confidence set by the heuristics. Popularity is computed using information such as page-rank and IDF of the title string.

4.2 YAGO TyCor

The YAGO TyCor uses the EDM step described above, and transforms the Wikipedia page URLs returned at the end of the step to corresponding DBpedia URIs.

Type Retrieval using DBpedia The TR algorithm produces a set of URIs for the Yago types of the candidate entity (the result of the EDM step). DBpedia contains type information for a vast number of Wikipedia entities, represented by the *rdf:type* relation, that come from several ontologies; the largest is YAGO. In many cases, the explicitly assigned type for an entity is a low-level (and highly-specific) type in the YAGO ontology, such as *yago:CompaniesEstablishedIn1898* (typically derived from Wikipedia Categories). We generalize this type by navigating up the hierarchy till we reach a Yago type that has a WordNet sense id associated with it (e.g. *yago:Company108058098*). This generalization helps type alignment (when aligning with types derived from the LAT) and improves the coverage of the TyCor.

PDM in YAGO The PDM algorithm produces a set of URIs for the Yago types that match the LAT, by matching the LAT to the labels or IDs of Yago types. We then score the matches based on a weighted combination of its WordNet sense rank, and the number of instances of

the concept in DBpedia. The latter is an approximation of type popularity, and has performed well in our experiments.

There are two additional features of our PDM algorithm that help improve its precision and recall respectively. The first is the notion of a domain-specific type-mapping file that is optionally input to the algorithm. For example, based on analysis of Jeopardy! question data, we found the LAT "*star*" refers to the sense of star as a *movie star* roughly 75% of the time, with the remaining cases referring to the *astronomical object*.

The second heuristic we use in PDM helps improve its recall. We estimate a statistical relatedness between two types by computing the conditional probability that an instance with type A also has type B, using the metric: NI (A and B) / NI (A), where NI is the number of instances of the concept in DBpedia (including instances of its subtypes). In PDM, if the lexical type matches some YAGO type, we expand it to include related types based on their conditional probabilities exceeding an empirically determined threshold (0.5).

YAGO Type Alignment The type alignment algorithm produces a single score based on the alignment of the *instance types* from the TR step, and the *LAT types* from the PDM step. The algorithm uses these conditions:

- **Equivalent/Subclass match.** When the instance type and the LAT type are equivalent (synonyms) in the YAGO ontology, or the instance type is a subclass (hyponym) of the LAT type, a score of 1.0 is returned.
- **Disjoint match.** When the instance type and LAT type are found to be disjoint (based on axioms added to YAGO, see Section 3.1) a score of -1.0 is returned.
- **Sibling match.** When the instance type and LAT type share the same parent concept in the YAGO ontology, a score of 0.5 is returned. In this case, we exclude cases in which parent classes' depth < 6, since these high level types (like "Physical Entity") tend to be less useful.
- **Superclass match.** When the instance type is a superclass (hypernym) of the LAT type a score of 0.3 is returned. This may seem counter-intuitive since the candidate answer is supposed to be an instance of the LAT and not vice versa, however, we have seen cases where checking the type alignment in the opposite direction helps, either due to inaccuracies in the EDM or PDM step, or due to source errors, or the question itself asks for the type of a particular named entity.
- **Statistical Relatedness.** When the statistical type relatedness between the instance type and LAT type, computed as described above, exceeds an empirically determined threshold of 0.5, a score of 0.25 is returned.
- **Lowest Common Ancestor (LCA).** When the LCA of the instance type and LAT type is deep in the taxonomy (we use a depth threshold of 6 in the Yago taxonomy), a score of 0.25 is returned. This is based on the intuition that the

types are strongly related, even though they may not be a direct subclass or sibling relationship among them, if their LCA is not a very high level class.

The thresholds used in the type matching conditions above and the weights of the respective rules are manually assigned based on an empirical evaluation, conducted by running the algorithm on a large number of test cases. Since the TR phase may produce multiple types per candidate, the maximum type alignment score is returned.

4.3 Wiki-Category and Wiki-List TyCors

The Wiki-Category and Wiki-List TyCors are fundamentally different from YAGO TyCor because the types that they use are natural language strings and not types in a formal ontology. The Wikipedia list pages do not have any explicit structure among them. Wikipedia categories do have some hierarchical structure, but the Wiki-Category TyCor does not use that structure as it is too unreliable.

Both of these TyCors use the same **Entity Disambiguation and Matching** component as YAGO TyCor. They also both use a simple lookup in an RDF store for **Type Retrieval** (we augmented DBpedia with list associations), that returns the category (resp. list) names for the entity.

Wiki-Category and Wiki-List both have a trivial **Predicate Disambiguation and Matching** step that simply returns the LAT itself.

Type Alignment is the most complex portion of these TyCors. It receives as input a natural language LAT and a natural language type (from Type Retrieval) and tries to determine if they are consistent. In both cases, Type Alignment divides this task in to two distinct subtasks:

(1) Is the head word of the LAT consistent with the head word of the category or list names,

(2) Are the modifiers of head word of the LAT consistent with the modifiers of the category or list names.

In both cases, terms are matched using a variety of resources such as WordNet. For example, given a list named “cities in Canada” and a question asking for a “Canadian metropolis” (i.e., with a LAT “metropolis” that has a modifier “Canadian”), Type Alignment will separately attempt to match “city” with “metropolis” and “Canada” with

"Canadian." Type Alignment uses a variety of resources to do this matching; for example, in WordNet the primary sense of "city" is synonymous with the primary sense of "metropolis." Wiki-Category and Wiki-List provide separate match scores for the (head word) type match and the modifier match. Those separate scores are used as features by the DeepQA Candidate Ranking mechanism.

5 Experiments

All experiments were done on the March, 2010 version of Wikipedia, and used DBpedia release 3.5. Wikipedia categories were obtained from DBpedia, Wiki lists were scraped from Wikipedia.

5.1 Evaluating EDM on Wikipedia Link Anchors

We evaluated the performance of the EDM component on a Wikipedia link anchor data set. This data set is comprised of 20,000 random pairs of Wikipedia anchor texts and their destination links. Note that the destination of an anchor text is a Wikipedia article whose title (string) may or may not explicitly match the anchor text. For example, the article *Gulf of Thailand* contains the following passage: "*The boundary of the gulf is defined by the line from Cape Bai Bung in southern Vietnam (just south of the mouth of the Mekong river) to the city Kota Baru on the Malayian coast*". While anchors *Mekong* and *Kota Baru* point to articles whose titles are exactly the same as the anchor text, *Cape Bai Bung*'s link points to the article titled "*Ca Mau Province"*.

We use the anchor texts as inputs to EDM, and the destinations as ground truth for evaluation, similar to 4. The performance of the EDM is shown in Table 4 with precision, recall (over all candidates returned) and the average number of candidates returned. We tested four versions of EDM, with and without popularity ranking and DBpedia name properties. Note that DBpedia names increase the number of candidates without impacting precision or recall. This is partially a side-effect of the link-anchor based evaluation, which skews the results to prefer alternate names that have been used as link anchor texts, and thus does not really reflect a test of the data the alternate names provide. However, from inspection we have seen the name properties to be extremely

noisy, for example finding DBpedia entities named "China" using the name properties returns 1000 results. The TyCor experiments below, therefore, do not use DBpedia name properties in the EDM step.

Ranking	Names	Precision	Recall	Avg. # of candidates
No	No	74.6%	94.3%	16.97
No	Yes	74.7%	94.3%	13.02
Yes	Yes	75.2%	94.3%	16.97
Yes	No	75.7%	94.3%	13.02

Table 4. EDM performance on 20,000 Wikipedia anchor texts

5.2 Evaluating Typing on Ground Truth

Although the TyCor components' impact on end-to-end QA performance can be measured through ablation tests, they do not reflect how well the components do at the task of type checking because wrong answers may also be instances of the LAT. To measure the TyCor components performance on the task of entity-typing alone, we manually annotated the top 10 candidate answer strings produced by Watson from 1,615 Jeopardy! questions, to see whether the candidate answer is of the same type as the lexical answer type. Because some questions contain multiple LATs, the resulting data set contains a total of 25,991 instances. Note that 17,384 (67%) of these are negative, i.e. the candidate answer does not match the LAT.

Tycor Component	Accuracy	Precision	Recall
Yago Tycor	76.9%	64.5%	67.0%
Wikipedia Category Tycor	76.1%	64.1%	62.9%
List Tycor	73.3%	71.9%	31.6%
All Three (Union)	73.5%	58.4%	69.5%

Table 5. Evaluating TyCor Components on Entity-Type Ground Truth

Table 5 shows the performance of the three TyCor components that use community–built knowledge, by counting any candidate with a TyCor score > 0.0 to be a positive judgment; this is not the way the score is used in the end to end Watson system, but gives a sense for how the different components perform. The "All Three" experiment counts any candidate with at least one score from the three TyCors that is > 0.0 to be a positive judgment. The bump in recall shows that they

can complement each other, and in the end to end system experiments below, this is validated.

5.3 Impact on end-to-end question answering

Table 6 shows the accuracy of the end-to-end DeepQA question answering system with different TyCor configurations.

	No TyCor	**YAGO TyCor**	**Wiki-Category TyCor**	**Wiki-List TyCor**	**All 3 TyCors**
Baseline Accuracy	50.1%	54.4% (+4.3%)	54.7% (+4.6%)	53.8% (+3.7%)	56.5% (+6.5%)
Watson Accuracy	65.6%	68.6% (+3.0%)	67.1% (+1.5%)	67.4% (+1.8%)	69.0% (+3.4%)

Table 6: Accuracy on end-to-end question answering with only the TyCors specified in each column

The "Baseline Accuracy" shows the performance of a simple baseline DeepQA configuration on a set of 3,508 previously unseen Jeopardy! questions. The baseline configuration includes all of the standard DeepQA candidate generation components, but no answer scoring components other than the TyCors listed in the column headings. The baseline system with no TyCor components relies only on the candidate generation features (e.g., rank and score from a search engine) to rank answers. The subsequent columns show the performance of the system with only that component added (the difference from No Tycor is show in parens), and the final column shows the impact of combining these three TyCors.

The "Watson Accuracy" shows the accuracy of the complete Watson question answering system *except* for the TyCor components. Again, the first column shows the accuracy of the system with no TyCor components, and the next four columns show distinct (*not* cumulative) additions, and the last column shows the accuracy with all three combined. Again, each TyCor alone is better than no TyCor, and effectively combining all three components is better than any one of them.

All the column-wise gains (from Baseline to Watson), and all the differences from "No Tycor" shown in the table are statistically significant

(significance assessed for $p < .05$ using McNemar's test with Yates' correction for continuity). The "All-3 Tycors" improvement is significant over the individuals in all cases except the Watson Yago TyCor.

7. Related Work

QUARTZ 12 is a QA System that uses a statistical mapping from LATs to WordNet for PDM, and collocation counts for the candidate answer with synonyms of the mapped type for Type Retrieval. In 6 the approach has been taken a step further by combining correlation-based typing scores with type information from resources such as Wikipedia, using a machine-learning based scheme to compute type validity. Both 6 and 12 are similar to our TyCor approach in that they defer type-checking decisions to later in the QA pipeline and use a collection of techniques and resources (instead of relying on classical NERs) to check for a type match between the candidate and the expected answer type in the question. However, the fundamental difference with our approach is that the type match information is not used as a filter to throw out candidate answers, instead, the individual TyCor scores are combined with other answer scores using a weighted vector model. Also, our type-coercion is done within a much more elaborate framework that separates out the various steps of EDM, PDM, Type Retrieval and Alignment etc.

A similar approach to our combination of NED and WikiCat is presented in 3. The traditional type-and-generate approach is used when question analysis can recognize a semantic answer type in the question, and falls back to Wikipedia categories for candidate generation, using it as a hard filter instead of predictive annotation. In our approach we assume any component can fail, and we allow other evidence, from other TyCor components or other answer scoring components, to override the failure of one particular component when there is sufficient evidence.

An approach to overcoming problems of *a-priori* answer type systems in proposed in 9, based on discriminative preference ranking of answers given the question focus and other words from the question. This approach is actually quite similar in spirit to other components of our TyCor suite that we did not discuss here. In other work we have shown that techniques like this provide coverage for infrequent types that may not have been accounted for in some ontology or type system,

such as “scarefest” to describe a horror movie, but do not perform nearly as well on the types known by the ontology. Thus we found combining techniques like 9 with those that use structured information provides the best overall performance.

A TyCor component that uses Linked Open Data (LoD), including DBpedia, geoNames, imdb, and MusicBrainz was presented in 8. This TyCor component includes a special-purpose framework for scaling LoD type triple datasets combined with *latent semantic indexing* to improve the matching steps (EDM and PDM), and an intermediate ontology designed specifically for the Jeopardy! task. The approach is evaluated on the classification task alone, as our question answering performance was still confidential. The classification performance is considerably lower than shown here, roughly 62% accuracy, though we expect it to improve as more LoD sources are added. In internal experiments, it had no impact on Watson performance, and was not used in the final fielded Watson system.

The idea of using Wikipedia link anchor text as a gold standard was presented in 4, along with a word-sense disambiguation algorithm for EDM using context vectors and Wikipedia categories. Our EDM results (see Table 4) are roughly the same, as the 86-88% accuracy numbers reported in 4 do not include cases where recall fails completely. In our experiments, we found popularity ranking of the results from our heuristics performs just as well as the method in 4, and is significantly faster at run-time.

8. Conclusion

We have presented a novel open-domain type coercion framework for QA that overcomes the brittleness and coverage issues associated with Predictive Annotation techniques. The TyCor framework consists of four key steps involving entity disambiguation and matching (EDM), predicate disambiguation and matching (PDM), type retrieval and type alignment. We have shown how community-built knowledge resources can be effectively integrated into this TyCor framework and provided corresponding algorithms for the four TyCor steps. The algorithms exploit the structure and semantics of the data, and in some cases, benefit from extensions made to existing knowledge to add value (e.g. addition of disjoints to YAGO). Our results show that the TyCors built using

Web knowledge resources perform well on the EDM and entity typing tasks (both fundamental issues in NLP and Knowledge Acquisition), as well significantly improving the end-to-end QA performance of the Watson system (which uses machine learning to integrate TyCor) on rich and complex natural language questions taken from Jeopardy!

References

1. Elif Aktolga, James Allan, and David A. Smith. Passage Reranking for Question Answering Using Syntactic Structures and Answer Types. In P. Clough et al. (Eds.): ECIR, LNCS 6611, pp. 617–628 (2011).
2. Sören Auer , Christian Bizer , Georgi Kobilarov , Jens Lehmann , Zachary Ives. DBpedia: A Nucleus for a Web of Open Data. In 6th Int'l Semantic Web Conference, Busan, Korea (2007)
3. D. Buscaldi and P. Rosso. Mining Knowledge from Wikipedia for the Question Answering task. In Proceedings of the International Conference on Language Resources and Evaluation. (2006)
4. Silviu Cucerzan. Large-Scale Named Entity Disambiguation Based on Wikipedia Data. In Proceedings of EMNLP-2007. pp 708-716. Prague, (2007).
5. David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A. Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John Prager, Nico Schlaefer, and Chris Welty. Building Watson: An Overview of the DeepQA Project. In, AI Magazine. (2010).
6. A Grappy, B Grau. Answer type validation in question answering systems. Proceeding RIAO '10 Adaptivity, Personalization and Fusion of Heterogeneous Information (2010).
7. George A. Miller. WordNet: A Lexical Database for English. Communications of the ACM Vol. 38, No. 11: 39-41 (1995).
8. Yuan Ni, Lei Zhang, Zhaoming Qiu, and Chen Wang Enhancing the Open-Domain Classification of Named Entity Using Linked Open Data. In Proceedings of ISWC (2010).
9. Christopher Pinchak and Dekang Lin and Davood Rafiei. Flexible Answer Typing with Discriminative Preference Ranking. In, Proceedings of EACL 2009. pp 666—674. (2009)
10. Prager, J.M., Brown, E.W., Coden, A. and Radev, R. Question-Answering by Predictive Annotation. Proceedings of SIGIR 2000, pp. 184-191, Athens, Greece. (2000)
11. J. Pustejovsky. Type Coercion and Lexical Selection. In Semantics and the Lexicon, J. Pustejovsky (ed.), Kluwer Academic Publishers, Dordrecht, The Netherlands. (1993)
12. Stefan Schlobach, David Ahn, Maarten de Rijke, Valentin Jijkoun. Data-driven type checking in open domain question answering. J. Applied Logic **5**(1): 121-143 (2007)
13. Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A Core of Semantic Knowledge-Unifying WordNet and Wikipedia. In Proceedings WWW (2007).