A framework for merging and ranking of answers in DeepQA

D. C. Gondek
A. Lally
A. Kalyanpur
J. W. Murdock
P. A. Duboue
L. Zhang
Y. Pan
Z. M. Qiu
C. Welty

The final stage in the IBM DeepQA pipeline involves ranking all candidate answers according to their evidence scores and judging the likelihood that each candidate answer is correct. In DeepQA, this is done using a machine learning framework that is phase-based, providing capabilities for manipulating the data and applying machine learning in successive applications. We show how this design can be used to implement solutions to particular challenges that arise in applying machine learning for evidence-based hypothesis evaluation. Our approach facilitates an agile development environment for DeepQA; evidence scoring strategies can be easily introduced, revised, and reconfigured without the need for error-prone manual effort to determine how to combine the various evidence scores. We describe the framework, explain the challenges, and evaluate the gain over a baseline machine learning approach.

Introduction

IBM Watson* answers questions by first analyzing the question, generating candidate answers, and then attempting to collect evidence over its resources supporting or refuting those answers. For each answer, the individual pieces of evidence are scored by *answer scorers*, yielding a numeric representation of the degree to which evidence justifies or refutes an answer. The role of final merging is to use the answer scores in order to rank the candidate answers and estimate a confidence indicating the likelihood that the answer is correct. Crafting successful strategies for resolving thousands of answer scores into a final ranking would be difficult, if not impossible, to optimize by hand; hence, DeepQA instead uses machine learning to train over existing questions and their correct answers. DeepQA provides a confidence estimation framework that uses a common data model for registration of answer scores and performs machine learning over large sets of training data in order to produce *models* for answer ranking and confidence estimation. When answering a question, the models are then applied to produce the ranked list and confidences. This approach allows developers to focus on optimizing the performance of their component and entrust to the framework the assessment and proper combination of the component's scores for the end-to-end question-answering task.

We first identify some of the challenges we faced in applying machine learning in the question-answering domain. We then describe how our approach fits in an agile development setting with many contributors. We discuss the framework designed in the DeepQA system and the capabilities it provides for applying machine learning techniques. Then, we discuss how capabilities of that framework were used to address the challenges in this domain. We finish with an experimental evaluation showing some of the major contributors to improvements in system accuracy.

Challenges arising in question answering

There are a number of characteristics of the question-answering domain and the design of DeepQA that impose special challenges in using machine learning to rank answers and estimate confidence. The following are most notable:

- Some sets of candidate answers may be equivalent, whereas others may be closely related; in the latter case, evidence for one of those answers may be relevant to the other.

Digital Object Identifier: 10.1147/JRD.2012.2188760

0018-8646/12/$5.00 © 2012 IBM

- The significance of the various features may be radically different for different questions and question classes (e.g., puzzles and language translation), and there may be little training data available for some question classes (e.g., language translation clues are very rare in Jeopardy!**).
- Some features are more or less valuable at different stages of ranking. For example, after a preliminary ranking is performed, some features that had minimal impact on the preliminary ranking may be disproportionately important for making finer distinctions among the high-ranked candidates.
- Features are extremely heterogeneous. They are derived from a variety of distinct algorithms that were independently developed. Consequently, many features are not normalized in any way, and their distribution may vary considerably conditioned on characteristics of the question. Feature values are frequently missing, and some occur sparsely in the training set.
- There is a large class imbalance. The system may find an enormous number of candidate answers using text search [1]. However, few of these will be correct answers.

Challenges in deploying machine learning in an agile development environment

Machine learning for confidence estimation has been vital for the development of Watson since its early beginnings when it had just two features to combine. As part of the Watson methodology [2], experiments are conducted to measure the effect of the addition or modification of any component. To properly assess the impact, models must be retrained with the changes incorporated. The confidence estimation framework has been used in more than 7,000 experiments. This is a dramatically different environment in which to deploy machine learning compared with traditional settings where models are trained over static feature sets and allow extensive and expert tuning by hand. In our setting, the framework must be as turnkey and automatic as possible, driving the framework's facilities for missing value imputation, filtering sparse features, and use of robust machine learning techniques without sensitive parameter selection.

Framework

The machine learning framework takes a set of candidate answers and their evidence scores. It trains a model for ranking them and for estimating a confidence that each is correct. Here, an instance is a question–answer pair and the features are the candidate answer scores. Most of these scores provide some measure of how justified the answer is for the question; there are also scores capturing question-level features, which indicate information about the question, for instance, the lexical answer type (LAT) [3]. Although question-level features will be shared across candidate answers for a given question, they can help better assess confidence (which substantially varies depending on question type) and may also be used to better weigh the other features to help in ranking.

We assume that there is a training set $X = \{x_0, x_1, \ldots x_N\}$ with ground truth labels $Y = \{y_0, y_1, \ldots y_N\}$ with $y_i = \{+1, -1\}$ representing whether an answer is correct or incorrect. The goal for a test question $X = \{x_0, \ldots, x_M\}$ is to rank x_i in order to optimize a metric, typically question accuracy (whether the top-ranked answer is correct). This and other metrics are discussed in more detail in the "Experiments" section. The Watson system was trained with a set of approximately 25,000 Jeopardy! questions comprising 5.7 million question-answer pairs (instances) where each instance had 550 features.

The DeepQA machine learning framework is phase-based. In each phase, candidate answers and their features may be manipulated, after which a model is trained and applied to produce a confidence score for each answer. An answer's confidence score from one phase may be used by later phases. This general framework of successive application of models allows for implementation of machine learning techniques such as transfer learning, stacking [4], and successive refinement. The number and function of phases are configurable and, in Watson's instantiation for Jeopardy!, consisted of the following seven phases:

1. *Hitlist Normalization*—Rank and retain top 100 answers (see the section "Successive refinement").
2. *Base*—Partition into question classes (see the section "Specializing to question classes").
3. *Transfer Learning*—Transfer learning for uncommon question classes (see "Transfer learning").
4. *Answer Merging*—Merge evidence between equivalent answers and select a canonical form (see "Answer Merging").
5. *Elite*—Successive refinement to retain top five answers (see "Successive refinement").
6. *Evidence Diffusion*—Diffuse evidence between related answers (see "Evidence diffusion").
7. *Multi-Answers*—Join answer candidates for questions requiring multiple answers.

Within each phase, there are three main steps:

1. *Evidence Merging* combines evidence for a given instance across different occurrences (e.g., different passages containing a given answer).
2. *Postprocessing* transforms the matrix of instances and their feature values (e.g., removing answers and/or features, deriving new features from existing features).
3. *Classifier Training/Application* runs in either *training mode*, in which a model is produced over training data,

or *application mode*, where the previously trained models are used to rank and estimate confidence in answers for a run-time question.

- *Training*—This step treats the combined results of evidence merging and postprocessing as a set of instances for training a model. It uses a manually created answer key as ground truth for supervised learning. This step occurs only on training data.
- *Application*—This step applies the model developed during training to the combined results of evidence merging and postprocessing. It produces a confidence score. Answers are ranked in accordance with their confidence scores. This step occurs on test data. In addition, this step occurs on the training data for each phase *after* training is complete for all instances on that phase but *before* training begins on the next phase; this ordering is essential because later phases can use the classifier rank and score of earlier phases as additional features for each instance.

The ease of experimentation is achieved by wrapping these high-level constructs in a domain-specific language (DSL) [5] targeted toward a declarative exploration of what-if scenarios in machine learning. In the following subsection, we drill-down into different issues that occur in one or more steps of one or more of these phases. Then, in the "Integrated design" section, we provide a broader perspective on how these pieces fit together.

Techniques

We now detail some of the algorithmic techniques implemented within the DeepQA confidence estimation framework. First, the facility for final merging is discussed, in which answers and their evidence may be merged. Then, we discuss the application of machine learning and particular machine learning technologies that are used.

Answer Merging

There is often more than one way to textually refer to the same entity, e.g., "John F. Kennedy," "J.F.K.," and "Kennedy." Multiple candidate answers for a question may in fact be potentially equivalent despite their different surface forms. This is particularly confusing to ranking techniques that make use of relative differences between candidates. Without merging, ranking algorithms would be comparing multiple surface forms that represent the same answer and trying to discriminate among them. Although one line of research has been proposed based on boosting confidence in equivalent candidates after confidence scores are estimated [6], our approach is designed to first merge evidence in equivalent candidates and then estimate confidence of the combination. Different surface forms are often disparately supported in the evidence (for instance, "John Fitzgerald Kennedy" may be encoded in a knowledge-base, whereas "Kennedy" or "JFK" may be more commonly used in a textual content) and therefore result in complementary scores. This motivates an approach that merges answer scores before ranking and confidence estimation are performed. Using an ensemble of matching, normalization, and co-reference resolution algorithms, Watson identifies equivalent and related hypotheses and then enables custom merging per feature to combine scores.

The task of Answer Merging is to examine each pair of answers, determine whether they are equivalent or strongly related, and if so, merge the answers, select a canonical form, and trigger the evidence merging of their feature vectors into one. How exactly the features are combined is determined by the Feature Merging component (described below).

Our Answer Merging implementation consists of a number of independent components that use different techniques. A *Morphological* merger uses knowledge of English morphology to merge different inflections of the same base word. *Pattern-based* mergers are applied for certain semantic types (e.g., for person names, `FirstName LastName` is more specific than `LastName`; for locations, `City`, `State/Country` is more specific than `City`). *Table Lookup* mergers use prebuilt tables of related entities. We have automatically extracted these tables from semi-structured text, for example, Wikipedia** redirects and disambiguation pages and boldface terms in the first paragraph of a Wikipedia article.

One important consideration is that whenever two answers are combined, we must choose a single string to represent the combined answer (called the "canonical form" of the combined answer). Answer strings are often ambiguous, and detecting equivalence is noisy; thus, the system will, at times, merge nonequivalent answers. Robust techniques to select canonical form are needed, as selecting poorly can be a disaster, causing a correct answer (which, in many cases, may have been ranked in first place) to be supplanted by the incorrect answer that it was merged with. Consider a question where the correct answer is John F. Kennedy. If we spuriously merge our answer "J.F.K." with "John F. Kennedy International Airport," selecting the better canonical form can mean the difference between being judged correct or incorrect. Our strategy in selecting a canonical form follows the intuition that the correct form (in this case, "J.F.K.") will likely have better evidence for the question than the incorrect form.

We used our multiphase learning architecture to greatly improve the selection of the canonical form. In an initial phase, we do no answer merging. The machine learning model trained in this phase is used to generate an initial ranking of the individual answers. Later, we introduce an Answer Merging phase that examines each pair of answers

and decides which pairs should be merged. Because we have done the initial ranking, we can then choose the canonical form by selecting whichever answer had the better initial ranking (unless there is significant evidence that suggests overriding this and choosing a different canonical form). By using this strategy, we can avoid the serious problem where we lose a top-ranked answer by merging it with some wrong answer and choosing the wrong answer as the canonical form.

Answer Merging can merge answers that are connected by a relation other than equivalence. For example, our implementation merges answers when it detects a `more_specific` relation between them. In addition to combining their evidence, this was used during a Jeopardy! game to address the case where Watson selected a canonical form for the answer that was not wrong but was not specific enough. When the host asked "more specific?", if the answer had a variant that was `more_specific` than the canonical form, it would be given. In one sparring game question:

> MYTHING IN ACTION: One legend says this was given by the Lady of the Lake & thrown back in the lake on King Arthur's death.

Watson merged the two answers "sword" and "Excalibur" and selected "sword" as the canonical form (because it happened to have a better initial ranking). The host asked "more specific?" and Watson was able to correctly answer "Excalibur" due to having detected a `more_specific` relation between the two.

Answer Merging takes place in the Evidence Merging step of the Answer Merging phase.

Feature Merging

In the input to the final merger, an answer is often associated with multiple pieces of evidence, and a given feature can have a different value in each (e.g., if an answer occurs in several different passages, each passage-scoring algorithm will provide a distinct score for that answer in each passage [7]). In order to produce a feature vector that is appropriate for our machine learning model, we need to transform this input such that each feature has exactly one value. In addition, when Answer Merging occurs, we must combine the feature vectors of the individual answers and produce a single feature vector for the merged answer.

Feature Merging is the task of reducing the multiple values associated with each feature label down to one value per feature label. Our implementation supports plugging in a different feature-merging policy for each feature. For many features, it works well to select the "best" of all of the values (maximum or minimum, depending on the feature). However, there are situations where one can do better.

For example, in Watson's massively parallel architecture, a passage-scoring feature [7] computes a score for each candidate on the basis of a single supporting passage for the candidate. Selecting the best value for the feature results in considering the evidence only from the one passage that best supports that answer. This can be an effective policy for scorers that use deep semantic analysis and only give high scores when a passage provides clear direct evidence of the answer being correct. In contrast, other passage-scoring algorithms only consider whether the answer appears in proximity to terms from the question; for those features, we are particularly interested in combining results across multiple passages with the expectation that correct answers are more likely to have higher combined scores. One technique that we use to combine results across passages is to sum the scores for each passage. Another strategy that we use for some passage-scoring features is "decaying sum." Decaying sum is defined as

$$decay(p_0, \ldots, p_K) = \sum_{i=0}^{K} \frac{p_i}{2^i},$$

where $p_0, \ldots, p_K$ are the scores of the passages that contain the answers, sorted in descending order. Decaying sum is designed to perform better than sum in cases in which having a few passages that seem to match the question well is better than having many questions that match less well. Decaying sum performs better empirically than the other merging strategies for many passage-scoring features; hence, we configure the feature merger to use it for those features.

Feature Merging takes place in the Evidence Merging step of various phases. Specifically, Feature Merging is first employed in the Hitlist Normalization phase to combine merge features across different instantiations of an answer (e.g., different passages containing the same answer). Feature Merging is also employed after Answer Merging in the Answer Merging phase. It is then employed one last time after Evidence Diffusion in the Evidence Diffusion phase.

Ranking with classifiers

The DeepQA framework is designed to allow for experimentation with a range of machine learning approaches, and it is straightforward to integrate new ranking approaches. Many existing approaches to ranking take advantage of the fact that ranking may be reduced to classification [8]. The DeepQA framework has native support for classification techniques to be applied using the simple heuristic of sorting by classification score; this is the approach used in the Watson Jeopardy! application.

Over the course of the project, we have experimented with logistic regression, support vector machines (SVMs) with linear and nonlinear kernels, ranking SVM [9, 10],

boosting, single and multilayer neural nets, decision trees, and locally weighted learning [11]; however, we have consistently found better performance using regularized logistic regression, which is the technique used in the final Watson system. Logistic regression produces a score between 0 and 1 according to the formula

$$f(x) = \frac{1}{1 + e^{-\beta_0 - \sum_{m=1}^{M} \beta_m x_m}},$$

where m ranges over the M features for instance x and β_0 is the "intercept" or "bias" term. (For details, the interested reader is referred to [19].)

An instance x is a vector of numerical feature values, corresponding to one single occurrence of whatever the logistic regression is intended to classify. Output $f(x)$ may be used like a probability, and learned parameters β_m may be interpreted as "weights" gauging the contribution of each feature. For example, a logistic regression to classify carrots as edible or inedible would have one instance per carrot, and each instance would list numerical features such as the thickness and age of that carrot. The training data consists of many such instances along with labels indicating the correct $f(x)$ value for each (e.g., 1 for edible carrots and 0 for inedible carrots). The learning system computes the model (the β vector) that provides the best fit between $f(x)$ and the labels in the training data. That model is then used on test data to classify instances. In Watson's case, instances correspond to individual question/candidate-answer pairs, and the numerical values for the instance vector are features computed by Watson's candidate generation and answer-scoring components. The labels on the training data encode whether the answer is correct or incorrect. Thus, Watson learns the values for the β vector that best distinguish correct answers from incorrect answers in the training data. Those β values are then used on test data (such as a live Jeopardy! game) to compute $f(x)$, our confidence that each candidate answer is correct. Candidate answers with higher confidence scores are ranked ahead of candidate answers with lower confidence scores.

Logistic regression benefits from having no tuning parameters beyond the regularization coefficient and optimization parameters to set and therefore was robust without requiring parameter fitting in every experimental run. Answers must be both ranked and given confidence scores; thus, the framework does allow for separate models to be trained to perform ranking and confidence estimation. In practice, however, we found competitive performance using a single logistic regression model where ranking was done by sorting on the confidence scores.

Ranking with classifiers is performed in the Classifier Application step of each phase. It uses a model produced in the Classifier Training step for that phase.

Standardization

One aspect of ranking that can be lost when converting a ranking task to a classification task is the *relative* values of features within a set of instances to be ranked. It can be useful to look at how the value of the feature for one instance compares with the value of the same feature for other instances in the same class. For example, consider the features that indicate whether the candidate answer is an instance of the LAT that the question is asking for [12]. In cases where a large proportion of the candidate answers that the system has generated appear to have the desired type, the fact that any one of them does so should give the system relatively little confidence that the answer is correct. In contrast, if our search and candidate generation strategies were only able to find one answer of that appears to have the desired type, that fact is very strong evidence that the answer is correct.

To capture this signal in a system that ranks answers using classification, we have augmented the existing features with a set of *standardized* features. Standardization is *per query*, where for each feature j and candidate answer i in the set Q of all candidate answers for a single question, the standardized feature x_{ij}^{std} is calculated from feature x_{ij} by centering (subtracting the mean μ) and scaling to unit variance (dividing by the standard deviation σ) over all candidate answers for that question

$$x_{ij}^{\text{std}} = \frac{x_{ij} - \mu_j}{\sigma_j}, \quad \mu_j = \frac{1}{|Q|}\sum_{k=1}^{|Q|} x_{kj},$$

$$\sigma_j = \sqrt{\frac{1}{|Q|}\sum_{k=1}^{|Q|}(x_{kj} - \mu_j)^2}.$$

For each feature provided as input to the ranking component, we generate an additional new feature that is the standardized version of that feature. Both mean and standard deviation are computed within a single question since the purpose of standardization is to provide a contrast with the other candidates that the answer is being ranked against. We chose to retain the base features and augment with the standardized features as that approach showed better empirical performance than using standardized features or base features alone.

Standardization is performed during the postprocessing step of several different phases. It is performed in the Hitlist Normalization phase because that is the first phase so no standardized features exist. It is also performed in the Base and Elite phases because some answers are removed at the start of those phases; hence, the revised set of answers can have different mean and standard deviation. Furthermore, it is performed in the Answer Merging and Evidence Diffusion phases because the Feature Merging step for these phases makes substantial changes to the feature values of the instances. It is not performed in other phases because

those phases are able to inherit the standardized values from their predecessors.

Imputing missing values

Many answer scorers within the system are designed to use forms of analysis or resources that do not apply to every question (e.g., temporal scorers require that a temporal expression occur in the clue) or answer (e.g., knowledge-base-driven type coercion requires an answer to exist in its knowledge-base [12]). As a result, the feature vector may have many missing values. The fact that a feature is missing may provide important additional information in estimating correctness for an answer, providing a different signal from a zero feature score. The DeepQA confidence estimation framework provides an extensible set of configurable imputation policies that may incorporate feature scores of any candidate answers for that question as well as averages taken over the training set. One important capability is the ability to add a *response indicator*, here called a "missing flag," for each feature indicating whether it is missing. The response indicator is then treated as another feature in the training. The experimental results below show a sizable gain from employing this imputation strategy over the conventional baseline of imputing to train set mean.

Imputing missing values occurs as a postprocessing step during the first phase and in every other additional phase that introduces new features that might be missing.

Specializing to question classes

Different classes of questions such as multiple choice, useless LAT (e.g., "it" or "this"), date questions, and so forth may require different weighing of evidence. The DeepQA confidence estimation framework supports this through the concept of *routes*. Each phase specifies its set of routes, where for each route, a specialized model is trained. This is an ensemble approach where the space of questions is partitioned on question classes. In the Jeopardy! system, profitable question classes for specialized routing were manually identified. Then, components using the LAT or other features of the question were added to automatically identify the class of a question at train and test time. Although routes in Watson were manually identified, deriving routes from automatic techniques to learn mixtures of experts (as introduced in [13]) could also be done.

Specializing to question classes is done in the classifier training and classifier application steps of each phase.

Feature selection

Our machine learning framework allows each pair of phase and route to learn over some or all of the features in the complete system. Making all features available for every phase and route would be problematic because we have a very large number of features and some routes are specific to relatively uncommon types of questions; many features but few training instances tend to lead to overfitting. Thus, for those phase/route pairs for which we have a limited amount of training data, it is important to use only a subset of all features for training.

One useful metric for selecting features is the correlation between the feature value and whether the answer is correct. However, naively selecting features according to this criterion can lead to selecting many features, each of which is good individually but collectively do poorly because they overlap too heavily in the signal they provide. When we first started experimenting with routes for infrequently occurring question types, we ranked all features according to correlation and then manually selected features that were well correlated with correctness, making an effort to avoid too many features that we expected to have very similar behavior; for example, if many features relating to typing answers [12] were all highly correlated with answer correctness, we might select a few that were particularly high and then skip a few to get a more diverse set of features. This approach seemed effective but was also labor intensive and arbitrary.

To address this limitation, we experimented with a variety of off-the-shelf and custom techniques for automatically selecting features. One approach that we found to be effective employs a combination of correlation with correctness and best first search driven by the consistency subset attribute evaluator [14] in the Weka (Waikato Environment for Knowledge Analysis) machine learning toolkit [15]. Specifically, we used the best first consistency subset technique to select an initial set of features. However, in some cases, this approach appeared to be more conservative than we intended in terms of the number of features used (some features that appeared to be very useful for identifying right answers were omitted). Then, if the number of features selected was less than our heuristic estimate of the number of features desired (one for every ten training questions), we added in the features that had the highest correlation with correctness that were not included in the initial set. This process was separately performed for each phase/route pair with limited training data. We used this approach to select the features that were employed in the Jeopardy! configuration that performed on television.

Even in routes for which there is sufficient training data available, there may be some scorers that produce scores for only a very small portion of that training data. As discussed in the previous section, this sparse feature phenomenon is common for natural-language processing (NLP)-based scorers. Features of this sort can lead to overfitting since the learning algorithm has very little experience with those features. Although this effect may be

somewhat ameliorated by use of regularization techniques, the DeepQA framework provides an additional capability for *sparse feature filtering* in which features firing less than a predefined threshold are filtered out of the set used in training (during the Classifier Training step of each phase).

Feature selection is performed before training. We perform feature selection in each phase for each route that has a relatively small amount of training data. The list of selected features for each route of each phase is used during Classifier Training (to determine what features to use as training data) and during Classifier Application (to determine what features to apply to the model).

Transfer learning for rare question classes

Certain question classes of questions, for example, definition or translation questions, may benefit from specialized models. Because of their rarity, however, even over our entire set of questions, there may not be sufficient training instances. The phase-based framework supports a straightforward parameter-transfer approach to transfer learning by passing one phase's output of a general model into the next phase as a feature into a specialized model. In the case of logistic regression, which uses a linear combination of weights, the weights that are learned in the transfer phase can be roughly interpreted as an update to the parameters learned from the general task. Transfer learning is performed in the Transfer Learning phase.

Successive refinement

Although a single question in the Watson system typically results in hundreds of candidate answers, many of these will score quite poorly and a model trained over these may not optimally differentiate among high-scoring candidates. The Watson system implements successive refinement in which the first phase (i.e., the Hitlist Normalization phase) is used to weed out extremely bad candidates; the top 100 candidates after hitlist normalization are passed to later phases. A later "elite" phase near the end of the learning pipeline trains and applies to only the top five answers as ranked by the previous phase.

Instance weighting

As noted earlier, question answering tends to lead to severe class imbalance because it is easy to find many candidate answers and most questions have few correct answers. In the case of Jeopardy!, the system produces far more incorrect than correct answers with a class imbalance of approximately 94 to 1 in the main phases. We have experimented with instance weighting, cost-sensitive learning, and resampling techniques (based on sampling uniformly and sampling with bias toward those instances close to the decision boundary of a previously trained model). Logistic regression allows for instance weights to be associated with sets of instances. Ultimately, we settled on using an instance weighting of 0.5 for the incorrect instances, which showed the best empirical performance. Instance weighting is employed during the Classifier Training step of every phase.

Evidence diffusion

Evidence diffusion is inspired by [6]: gathering additional answer evidence on the basis of background relationship information *between* candidate answers. We attempt to infer from the context of the question and relationships between the candidates whether evidence found for one candidate is also meaningful evidence for another. In contrast with [6], which merges final answer scores, our approach merges evidence directly.

Consider the following Jeopardy! question:

> WORLD TRAVEL: If you want to visit this country, you can fly into Sunan International Airport or ... or not visit this country. (Correct answer: "North Korea")

The Sunan International Airport mentioned in the question is located in the city of Pyongyang. Most text sources tend to mention this fact, as opposed to the country, North Korea, where the airport is located. As a result, there is strong textual evidence for the candidate answer "Pyongyang" [7], which can overwhelm the type-based evidence [12] for the correct answer "North Korea", causing the question-answering system to prefer Pyongyang although it is of the wrong type.

In DeepQA, evidence may be *diffused* from *source* (Pyongyang) to *target* (North Korea) if a *source-target* pair meets the following evidence diffusion criteria.

1. The target meets the expected answer type (in this example, "country").
2. There is a semantic relation between the two candidates (in this case, *located-in*).
3. The transitivity of the relation allows for meaningful diffusion given the question.

Criterion 1 is evaluated using the LAT of the question and candidate type coercion scores [12], which indicate whether a candidate answer meets the expected answer type. This identifies suitable target candidates.

Criterion 2 requires identification of relationships between two given candidate answers using background knowledge sources. This is done in two ways.

- Consulting structured knowledge-bases such as DBpedia [16] by mapping the source and target to corresponding resources in a structured knowledge-base. As described in [12] and [17], this is a nontrivial entity disambiguation task.
- Using shallow lexical knowledge extracted from unstructured text [18], we mine over a large text corpus for frequent textual relations that link source and target.

Criterion 3 is particularly challenging because it requires a deep understanding of the semantics of the question and the relations found between answers and entities expressed in the question. For example, although *located-in* may be meaningful for the example question, a *formerly-ruled-by* relation (between Pyongyang and Japan) would not be meaningful for this question. For now, we use a restricted simple strategy that ignores the semantics of the relation and question and instead uses a preidentified set of relations that was filtered according to popularity and requires that the source answer is of the wrong type (which is judged using type-matching scores [12]).

If all criteria are met, we augment the target answer with a new set of features that are the scores of the source answer. The diffusion phase can then learn how to tradeoff between diffused and original scores. Suppose that the incoming phase uses a linear classifier with weights β_m. The input features to the evidence diffusion phase are the previous score, $f(x) = \sum_m \beta_m x_m$ as well as transferred features x'_m. Suppose weights γ are learned for a linear model $f'(x)$ in the Evidence Diffusion phase. The output of Evidence Diffusion is then

$$\begin{aligned} f'(x) &= \gamma_0 + \sum_m \gamma_m x'_m + \gamma_{M+1} f(x) \\ &= \gamma_0 + \sum_m \gamma_m x'_m + \gamma_{M+1}\left(\beta_0 + \sum_m \beta_m x_m\right), \\ &= \gamma_0 + \gamma_{M+1}\beta_0 + \sum_m \gamma_m x'_m + \gamma_{M+1}\beta_m x_m, \end{aligned}$$

where $\gamma_{M+1}\beta_m$ versus γ_m can be interpreted as a weighted linear combination of original features x_m and transferred features x'_m. Thus, the Evidence Diffusion phase learns how to tradeoff between an answer's original features and its transferred features diffused from related answers. It should be noted that in the actual implementation, we found good empirical performance using logistic regression for all phases, which further introduces the logistic function to the calculation of $f(x)$.

Evidence Diffusion is performed in the Evidence Merging step of the Evidence Diffusion phase.

Integrated design

Figure 1 provides a partial illustration of how the elements described above are integrated in Watson. A different DeepQA application may combine and configure these capabilities differently. The figure shows the first four phases of processing. Answers begin at the Hitlist Normalization phase. At the start of this phase, answers whose string values are identical are merged (even if they are derived from different sources). After merging, the standardized feature values are computed. The answers are then sent to one of a set of routes based on how the question was classified. There are distinct routes for Puzzle questions, Multiple-Choice questions, questions asking for a date, and questions asking for a number. There is also a route for questions with no focus, for which some passage-scoring algorithms may be less effective [7], and a route for questions with no LAT, for which the type coercion algorithms are not applicable [12]. Question characteristics that have an impact on the effectiveness of a broad class of scoring algorithms are generally good candidates for a distinct route because there is a reason to expect that different weights for the features are ideal for those classes. The last route is the *default* route, which handles answers for questions that do not fit into any of the other categories.

At the start of the base phase, answers are ranked according to the scores assigned by the Hitlist Normalization phase, and only the 100 highest scoring answers from that phase are retained. With at most 100 answers per question, the standardized features are recomputed. The recomputation of the standardized features at the start of the base phase is the primary reason that the Hitlist Normalization phase exists: By eliminating a large volume of "junk" answers (i.e., ones that were not remotely close to being considered), the remaining answers provide a more useful distribution of feature values to compare each answer to. The base phase provides the same set of routes as the previous phase.

The score assigned to each answer by the model (for the selected route) in the base phase is provided as an input feature to the Transfer Learning phase. The Transfer Learning phase provides additional routes for which we expect the results of the base model to provide a useful starting point but not an ideal complete model. For example, Final Jeopardy! questions are very similar to ordinary Jeopardy! questions in many respects, but there are some systematic differences in terms of topic areas and level of complexity. If we put a Final Jeopardy! model into the earlier phases, then our ability to rank Final Jeopardy! answers would be limited by the size of the available training data, which is much smaller than the available data for all Jeopardy! questions. On the other hand, if we did not provide a Final Jeopardy! model in any phase, then our learning system would have no way to build a model that is tuned to that class of questions. Introducing the Final Jeopardy! route in the Transfer Learning phase provides a compromise between these extremes; the model in the Final Jeopardy! route of the Transfer Learning phase is able to learn how much to trust the base model and how much to adjust those results on the basis of how various features perform on Final Jeopardy! questions.

The Answer Merging phase combines equivalent answers. As noted earlier in the "Answer Merging" section, it uses a variety of resources to determine equivalence. The reason there is a distinct Answer Merging phase is that the merging process uses the results of the previous phase to determine which of the various forms being merged is "canonical." The selected canonical form is the one that is used when

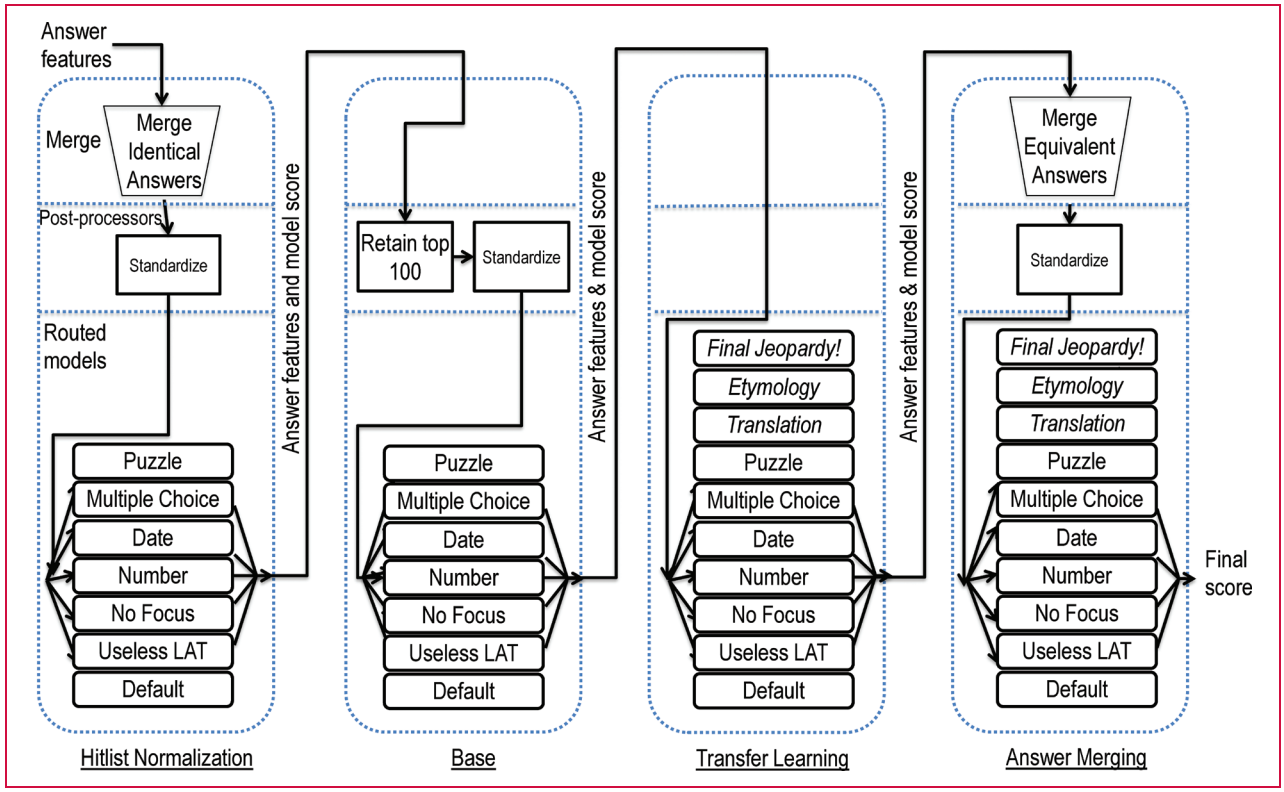

Figure 1

First four phases of merging and ranking in DeepQA.

presenting final answers. After merging, the answers are again divided up among the various routes to combine their features into a score for each answer. It is important to recombine the features after answer merging because different variants of an answer can have different kinds of evidence. For example, given a question asking for a president from Massachusetts, we might have evidence that "John F. Kennedy" is a president and "Jack Kennedy" is from Massachusetts. If we combine these answers (using whichever of those two labels scored higher in the previous phase), we want to then score that answer on the basis of the combination of those two kinds of evidence.

The remaining phases are not shown in Figure 1 because of space limitations. The elite phase works much like the base phase but employs only the top five answers from the previous phase. The Evidence Diffusion phase is similar to the Answer Merging phase but combines evidence from related answers, not equivalent ones (as described earlier). The Multi-Answer phase provides one additional type of merging that is only invoked for questions that ask for multiple distinct answers (e.g., a question asking for two countries that share a common characteristic); it combines multiple answers from earlier phases into a final answer. For those questions (only), it employs one additional model (with one route) that scores combined answers.

Experiments

We evaluate correctness and confidence using *precision* and *percentage answered*. *Precision* measures the percentage of questions that the system gets right out of those it chooses to answer. *Percentage answered* is the percentage of questions it chooses to answer (correctly or incorrectly). The system chooses which questions to answer on the basis of an estimated confidence score: For a given threshold, the system will answer all questions with confidence scores above that threshold. The threshold controls the tradeoff between precision and percentage answered. Assuming reasonable confidence estimation, for higher thresholds, the system will be more conservative, answering fewer questions with higher precision. For lower thresholds, it will be more aggressive, answering more questions with lower precision. *Accuracy* refers to the precision if all questions are answered; it corresponds to the vertical axis value on the far right of **Figure 2**. We often consider Precision@70 as a numerical

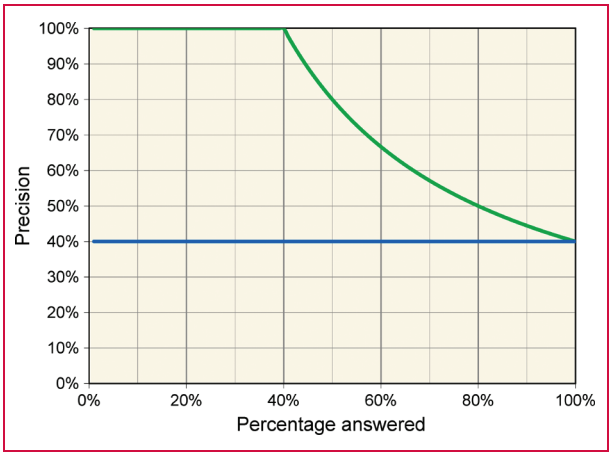

Figure 2

Precision versus percentage attempted: (green) perfect confidence estimation and (blue) no confidence estimation. (Modified and used, with permission, from D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, N. Schlaefer, and C. Welty, "Building Watson: an overview of the DeepQA project," *AI Magazine*, vol. 31, no. 3, pp. 59–79, 2010; ©2010 Association for the Advancement of Artificial Intelligence.)

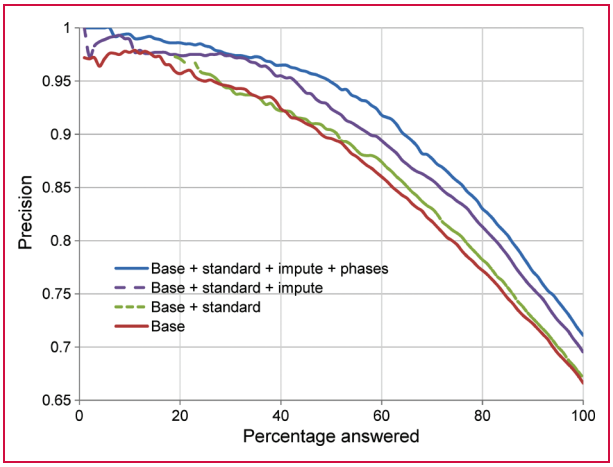

Figure 3

Improvements to answer precision from Final Merger enhancements. The precision at 100% answered represents the system accuracy.

measure that combines the ability to correctly answer questions and the ability to measure confidence; this metric corresponds to the precision when the system answers 70% of the questions for which it is most confident; it corresponds to the vertical axis value for the 70% horizontal axis value. Figure 2 shows the impact of confidence estimation for a plot of precision versus percentage attempted curves for two theoretical systems, obtained by evaluating the two systems over a range of confidence thresholds. Both systems have 40% accuracy. They differ only in their confidence estimation. The green line represents an ideal system with perfect confidence estimation. Such a system would identify exactly which questions it gets right and wrong and give higher confidence to those it got right. The blue line represents a system without meaningful confidence estimation. Because it cannot distinguish between which questions it is more or less likely to get correct, it has constant precision for all percent attempted.

Figure 3 shows the effect of our enhancements on confidence estimation. The red baseline curve is for a system using a single phase of logistic regression. The next line shows the effect of adding standardized scores to this phase, which results in a small gain in precision values. The next line adds in our custom per-query imputation strategy, which results in a jump of +2.4% accuracy. The gain from imputation may be surprising in comparison but shows the importance of imputation strategies within a system where many of the answer scorers may not fire. Finally, the top line is the system with all phases of confidence estimation integrated, which shows a further +1.6% incremental gain in accuracy. This version includes multiple routes for different question types as well as phases that perform transfer learning, successive refinement, and canonical form answer merging. Overall, the impact on system performance from the enhancements to the confidence estimation and machine learning is +4.5% accuracy and +5.9% improvement to Precision@70. The accuracy differences among these four configurations are all statistically significant ($p < 0.1\%$ from McNemar's test with Yates' correction for continuity) except for the difference between the first two ("base" and "base + standard" in Figure 1).

Conclusion

The DeepQA final merger framework is critically responsible for ranking and estimating confidence over candidate answers for a question. Throughout the development of DeepQA, this framework served as the test bed for a range of experiments on machine learning techniques to optimize ranking and confidence estimation for candidate answers. It also performs as the daily workhorse for integration of new scores as people experiment with adding answer scorers to the DeepQA system. Correspondingly, it must be robust and stable while also affording the flexibility to allow experimentation with advanced machine learning and answer-merging approaches. We have explained the framework and how it may be used to implement strategies to address the challenges that we have encountered in evidence-based hypothesis evaluation. Finally, we evaluated the enhancements to confidence estimation over a standard baseline and showed that they contribute a substantial improvement to system accuracy and precision.

Acknowledgments

The authors would like to acknowledge the contributions of other researchers who have done important work involving merging and ranking answers in DeepQA, including Apoorv Agarwal, Eric Brown, Ma Li, Manas Pathak, John Prager, and Chang Wang.

References

1. J. Chu-Carroll, J. Fan, B. K. Boguraev, D. Carmel, D. Sheinwald, and C. Welty, "Finding needles in the haystack: Search and candidate generation," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 6, pp. 6:1–6:12, May/Jul. 2012.
2. D. A. Ferrucci, "Introduction to 'This is Watson'," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 1, pp. 1:1–1:15, May/Jul. 2012.
3. A. Lally, J. M. Prager, M. C. McCord, B. K. Boguraev, S. Patwardhan, J. Fan, P. Fodor, and J. Chu-Carroll, "Question analysis: How Watson reads a clue," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 2, pp. 2:1–2:14, May/Jul. 2012.
4. D. H. Wolpert, "Stacked generalization," *Neural Netw.*, vol. 5, no. 2, pp. 241–259, 1992.
5. D. Ghosh, *DSLs in Action*. Greenwich, CT: Manning Publ., 2010.
6. J. Ko, E. Nyberg, and L. Si, "A probabilistic graphical model for joint answer ranking in question answering," in *Proc. 30th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2007, pp. 343–350.
7. J. W. Murdock, J. Fan, A. Lally, H. Shima, and B. K. Boguraev, "Textual evidence gathering and analysis," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 8, pp. 8:1–8:14, May/Jul. 2012.
8. M. Balcan, N. Bansal, A. Beygelzimer, D. Coppersmith, J. Langford, and G. B. Sorkin, "Robust reductions from ranking to classification," in *Proc. Mach. Learn.*, 2008, pp. 139–153.
9. R. Herbrich, T. Graepel, and K. Obermayer, "Large margin rank boundaries for ordinal regression," *Adv. Large Margin Classifiers*, vol. 88, no. 2, pp. 115–132, 2000.
10. Y. Cao, J. Xu, T. Y. Liu, H. Li, Y. Huang, and H. W. Hon, "Adapting ranking SVM to document retrieval," in *Proc. 29th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2006, pp. 186–193.
11. C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artif. Intell. Rev.*, vol. 11, no. 1–5, pp. 11–73, Feb. 1997.
12. J. W. Murdock, A. Kalyanpur, C. Welty, J. Fan, D. A. Ferrucci, D. C. Gondek, L. Zhang, and H. Kanayama, "Typing candidate answers using type coercion," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 7, pp. 7:1–7:13, May/Jul. 2012.
13. R. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Comput.*, vol. 3, no. 1, pp. 79–87, Spring 1991.
14. H. Liu and R. Setiono, "A probabilistic approach to feature selection—A filter solution," in *Proc. 13th ICML*, Bari, Italy, 1996, pp. 319–327.
15. Mach. Learn. Group, Univ. Waikato, *Weka 3: Data Mining Software in Java*. [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/
16. C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann, "DBpedia—A crystallization point for the web of data," *J. Web Semantics, Sci., Serv. Agents World Wide Web*, vol. 7, no. 3, pp. 154–165, Sep. 2009.
17. A. Kalyanpur, B. K. Boguraev, S. Patwardhan, J. W. Murdock, A. Lally, C. Welty, J. M. Prager, B. Coppola, A. Fokoue-Nkoutche, L. Zhang, Y. Pan, and Z. M. Qiu, "Structured data and inference in DeepQA," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 10, pp. 10:1–10:14, May/Jul. 2012.
18. J. Fan, A. Kalyanpur, D. C. Gondek, and D. A. Ferrucci, "Automatic knowledge extraction from documents," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 5, pp. 5:1–5:10, May/Jul. 2012.
19. E. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning,* 2nd ed. New York: Springer-Verlag, 2008. [Online]. Available: http://www-stat.stanford.edu/~tibs/ElemStatLearn/

Received November 15, 2011; accepted for publication December 19, 2011

David C. Gondek *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (dgondek@us.ibm.com)*. Dr. Gondek is a Research Staff Member and Manager at the T. J. Watson Research Center. He received a B.A. degree in mathematics and computer science from Dartmouth College in 1998 and a Ph.D. degree in computer science from Brown University in 2005. He subsequently joined IBM, where he worked on the IBM Watson Jeopardy! challenge and now leads the Knowledge Capture and Learning Group in the Semantic Analysis and Integration Department.

Adam Lally *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (alally@us.ibm.com)*. Mr. Lally is a Senior Technical Staff Member at the IBM T. J. Watson Research Center. He received a B.S. degree in computer science from Rensselaer Polytechnic Institute in 1998 and an M.S. degree in computer science from Columbia University in 2006. As a member of the IBM DeepQA Algorithms Team, he helped develop the Watson system architecture that gave the machine its speed. He also worked on the natural-language processing algorithms that enable Watson to understand questions and categories and gather and assess evidence in natural language. Before working on Watson, he was the lead software engineer for the Unstructured Information Management Architecture project, an open-source platform for creating, integrating, and deploying unstructured information management solutions.

Aditya Kalyanpur *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (adityakal@us.ibm.com)*. Dr. Kalyanpur is a Research Staff Member at the IBM T. J. Watson Research Center. He received his Ph.D. degree in computer science from the University of Maryland in 2006. His research interests include knowledge representation and reasoning, natural-language processing, statistical data mining, and machine learning. He joined IBM in 2006 and worked on the Scalable Highly Expressive Reasoner (SHER) project that scales ontology reasoning to very large and expressive knowledge bases. Subsequently, he joined the algorithms team on the DeepQA project and helped design the Watson question-answering system. Dr. Kalyanpur has over 25 publications in leading artificial intelligence journals and conferences and several patents related to SHER and DeepQA. He has also chaired international workshops and served on W3C Working Groups.

J. William Murdock *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (murdockj@us.ibm.com)*. Dr. Murdock is a member of the IBM DeepQA Research Team in the T. J. Watson Research Center. In 2001, he received a Ph.D. degree in computer science from Georgia Tech, where he was a member of Ashok Goel's Design and Intelligence Laboratory. He worked as a post-doctorate with David Aha at the U.S. Naval Research Laboratory. His research interests include natural-language semantics, analogical reasoning, knowledge-based planning, machine learning, and self-aware artificial intelligence.

Pablo A. Duboue *Les Laboratoires Foulab, Montreal, Quebec, Canada, H4C 2S3 (pablo@duboue.ca)*. Dr. Duboue is an independent language technologist. His work focuses on applied language technology and natural-language generation. He received a Licenciatura en Computacion degree from Cordoba University (Argentina) in 1998 and M.S., M.Phil., and Ph.D. degrees in computer science from Columbia University, New York, in 2001, 2003, and 2005, respectively. He is passionate about improving society through language technology and spends his time teaching, researching, and contributing to free software projects. He has taught at Cordoba University, Columbia University, and Siglo21 University and has worked at the IBM Thomas J. Watson Research Center as a Research Staff Member.

Lei Zhang *IBM Research Division, China Research Lab, Haidian District, Beijing 100193, China (tozhanglei@qq.com)*. Dr. Zhang received his Ph.D. degree in computer science and engineering from Shanghai Jiao Tong University in China in 2005. His research interests include knowledge representation, Semantic Web, information retrieval, and statistical machine learning. After his Ph.D. study, he worked as a Research Staff Member in the Information and Knowledge Department of IBM Research–China. Since 2005, he and his team have worked on Semantic Web technologies and semantic search and their applications in the healthcare domain. Since 2008, he and his team have worked on using structured knowledge (including Semantic Web data) to help question-answering in the DeepQA project. He is active in several academic communities and is one of the major initiators of the China Semantic Web Symposium series, which started in 2007. He has been a program committee member of conferences such as WWW, IJCAI (International Joint Conferences on Artificial Intelligence), ISWC (International Semantic Web Conference), etc. More recently, he was one of the local organizers of the ISWC 2010 conference.

Yue Pan *IBM Research Division, China Research Lab, Haidian District, Beijing 100193, China (panyue@cn.ibm.com)*. Mr. Pan is the Chief Scientist for Information Analytics and Executive for Healthcare of IBM Research–China. He is a major advocator of ontology technology and Semantic Web research in IBM. He and his team's research work focuses on knowledge representation and reasoning, Semantic Web, information integration, search and question-answering systems, and medical informatics. In addition to research at IBM, he also serves as doctoral advisor in Shanghai Jiao Tong University. He has several patents and has published tens of papers in international conferences and journals. He is a member of the Association for Computing Machinery (ACM), IEEE (Institute of Electrical and Electronics Engineers), CCF (China Computer Federation), and a committee member of YOCSEF (CCF Young Computer Scientists and Engineers Forum). He was the program co-chair of the International Semantic Web Conference 2010.

Zhao Ming Qiu *IBM Research Division, China Research Lab, Haidian District, Beijing 100193, China (qiuzhaom@cn.ibm.com)*. Dr. Qiu was a Research Staff Member in the IBM China Research Laboratory (CRL). He received a Ph.D degree in physics from Columbia University in 1985, after which he joined the Institute of Theoretical Physics, Chinese Academy of Science. In 1996, he joined the IBM CRL. He has received an IBM Outstanding Technical Achievement award for his work on homepage translator in the internet solution set. Dr. Qiu retired in 2006 and is now working part time in the IBM CRL.

Chris Welty *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (cawelty@gmail.com)*. Dr. Welty is a Research Staff Member in the Semantic Analysis and Integration Department at the T. J. Watson Research Center. He received a Ph.D. degree in computer science from Rensselaer Polytechnic Institute in 1995. He joined IBM in 2002, after spending 6 years as a professor at Vassar College, and has worked and published extensively in the areas of ontology, natural-language processing, and the Semantic Web. In 2011, he served as program chair for the International Semantic Web Conference, and he is on the editorial boards of the *Journal of Web Semantics*, the *Journal of Applied Ontology*, and *AI Magazine*.