A Machine Learning Approach for Factoid Question Answering[*]

David Dominguez-Sal
Universitat Politècnica de
Catalunya, DAMA-UPC
C/Jordi Girona 1-3, Barcelona
ddomings@ac.upc.edu

Mihai Surdeanu
Universitat Politècnica de
Catalunya, TALP
C/Jordi Girona 1-3, Barcelona
surdeanu@lsi.upc.edu

Resumen: Este artículo presenta un sistema de Question Answering para respuestas de tipo entidad que está completamente basado en técnicas de aprendizaje automático. Nuestro sistema consigue resultados similares al estado del arte para sistemas de QA con reglas para la extracción de respuestas construidas por un experto humano. Nuestro enfoque evita la intervención humana y simplifica la adaptación del sistema a nuevos entornos o conjuntos de atributos más extensos. Además, su tiempo de respuesta es adecuado para lugares donde se necesita un servicio de Question Answering orientado a tiempo real.
Palabras clave: Question answering, aprendizaje automático

Abstract: This paper presents a factoid Question Answering system that is fully based on machine learning. Our system achieves similar results to a state-of-the-art QA system with answer extraction rules developed by a human expert. Our approach avoids human intervention and simplifies adaptation of the system to new environments or extended feature sets. Moreover, its response time is suitable for places where real-time Question Answering is needed.
Keywords: Question answering, machine learning

1 *Introduction*

Question Answering (QA) is the task of extracting short, relevant textual answers in response to natural language questions. As a subset of QA, factoid QA focuses on questions whose answers are syntactic and/or semantic entities, e.g. organization or person names. QA systems in general have many important real-world applications, such as search engine enhancements or automated customer service.

As of today, the state of the art in QA technology combines machine learning (ML) with linguistic information encoded by human experts in the form of rules or heuristics in most of the QA systems' components (Paşca, 2003). The main drawback of these approaches is that such systems have a extremely high cost of development or customization, especially when high coverage is desired. For example, the system described in (Paşca, 2003) analyzes natural language questions using a manually-built ontology that maps a large number of WordNet synsets to known question types.

In this paper we show that a state-of-the-art QA system can be designed using only ML, known information retrieval (IR) techniques, and a limited amount of world knowledge. The system introduced in this paper performs comparably with the state of the art (we consider only systems without logic prover, as it is ours), even though the amount of training data made available to the system was small. The proposed approach has several other important advantages: (a) it is robust – it relies only on a small number of syntax-analysis tools (part of speech taggers and syntactic chunk detectors); (b) it is easy to customize or extend with new features or knowledge; and (c) it provides answers in real

[*] David Domínguez is granted by the *Generalitat de Catalunya* (2005FI00437). Mihai Surdeanu is a research fellow within the Ramón y Cajal program of the Spanish Ministry of Education and Science. DAMA-UPC want to thank *Generalitat de Catalunya* for its support through grant GRE-00352

time due to its simple architecture based on a small number of multi-class classifiers.

This paper is organized as follows. In Section 2 we describe the Question Answering system. Section 3 describes the empirical evaluation. We conclude with a discussion comparing our ML approach with a state-of-the-art Answer Extraction system based on human developed extraction rules.

2 Architecture

The QA system introduced in this paper uses a typical architecture consisting of three components linked sequentially: *Question Processing* (QP), which identifies the type of the put question, followed by *Passage Retrieval* (PR), which extracts a small number of relevant passages from the underlying speech transcriptions, and finally *Answer Extraction* (AE), which extracts and ranks exact answers from the previously retrieved passages. This section describes all these components.

2.1 Question Processing

The QP component detects the type of the input questions by mapping them into a two-level taxonomy consisting of 6 question types and 53 subtypes:

type	**subtype**
ABBREVIATION	abbreviation, expression abbreviated
ENTITY	animal, body organ, color, creative work, currency, disease, event, food, instrument, language, letter, other, plant, product, project, religion, sport, symbol, system, technique, equivalent term, vehicle, special word
DESCRIPTION	definition, description, manner, reason
HUMAN	group, individual, title, description
LOCATION	city, country, mountain, other, state
NUMERIC	angle, code, count, date, distance, money, order, other, period, percent, speed, temperature, size, weight

In our system there is a one-to-one mapping from question type to the category of the expected answer. For example, the tuple `LOCATION:country` entails that the answer is a named entity of type LOCATION.

Because the NERC used in the experiments reported in this paper extracts only 3 types of entities (Person, Location and Organization), we selected questions whose answers correspond to those types. We map question types to answer types using the following mapping:

Answer Type	**type:subtype**
PERSON	HUMAN:individual
LOCATION	LOCATION:{city, country, mountain, other,stated}
ORGANIZATION	HUMAN:group

Note that the three selected types make Answer Extraction more difficult. There are two main reasons: (a) our answer types are difficult to be identified. For example, (Surdeanu, Turmo, and Comelles, 2005) report F-measure differences of more than 16 points higher for Money entities than Organization ones. Moreover, (Surdeanu, Turmo, and Comelles, 2005) show that Person and Organization types figure among hardest types to be recognized. (b) The considered answer types yield more answers (in most corpora). For instance, (Surdeanu, Turmo, and Comelles, 2005) show that the number of location names in the Switchboard corpus are six times more frequent than the money entities. If the number of entities is high then it gives the system more options to choose and consequently increases the probability of error.

The question taxonomy is largely inspired by (Li and Roth, 2002). Nevertheless our classification mechanism is different: instead of using a hierarchy of 6 + 53 binary classifiers (one for each type and subtype), we opted for a single Maximum Entropy multi-class classifier that extracts the best tuple <`type:subtype`> for every question. We chose the single-classifier design because it significantly improves the classification response time, which is a paramount requirement for any interactive system. We compensate for the possible loss of accuracy with a richer feature set. Formally, our question classifier assigns a question class – i.e. tuple <`type:subtype`> – to each question, using the function:

$$qc(\mathbf{q}) = \underset{c\,\in\,\mathcal{C}}{\arg\max}\ \text{score}(\phi(\mathbf{q}), c) \qquad (1)$$

where $\mathbf{q}$ is the sequence of all the question words, e.g. {“What”, “is”, “the”, “Translanguage”, “English”, “Database”, “also”, “called”}, $\mathcal{C}$ is the set of all possible question classes, score is the classifier confidence, and ϕ is a feature extraction function, defined as a composition of several base feature extraction functions:

$$\phi(\mathbf{q}) = \phi_{sequence}(\mathbf{q}) + \phi_{sequence}(\mathbf{h}) + \phi_{qfw}(\mathbf{q}) \qquad (2)$$

where $\mathbf{h}$ is the sequence of heads of the basic syntactic phrases in the question, e.g. {“What”, “is”, “Database”, “called”} for the above example, $\phi_{sequence}$ extracts n-gram features from a sequence of words, and ϕ_{qfw}

```
$\phi_{\textbf{sequence}}(\mathbf{x})$
foreach($x_i \in \mathbf{x}$) add features:
    w($x_i$), l($x_i$), $\textbf{sem}(x_i)$, $\textbf{prox}(x_i)$,
    w($x_i$) $\cdot$ w($x_{i+1}$), l($x_i$) $\cdot$ l($x_{i+1}$)
    foreach($c \in \textbf{sem}(x_i), c' \in \textbf{sem}(x_{i+1})$): $c \cdot c'$
    foreach($c \in \textbf{prox}(x_i), c' \in \textbf{prox}(x_{i+1})$): $c \cdot c'$

$\phi_{\textbf{qfw}}(\mathbf{x})$
add features:
    w(qfw($\mathbf{x}$)), l(qfw($\mathbf{x}$)), $\textbf{sem}$(qfw($\mathbf{x}$)), $\textbf{prox}$(qfw($\mathbf{x}$)),
    w($x_0$) $\cdot$ w(qfw($\mathbf{x}$)), w($x_0$) $\cdot$ p(qfw($\mathbf{x}$))
    foreach($c \in \textbf{sem}$(qfw($\mathbf{x}$))): w($x_0$) $\cdot c$
    foreach($c \in \textbf{prox}$(qfw($\mathbf{x}$))): w($x_0$) $\cdot c$
```

Table 1: The feature extraction functions for the question classifier. w - token word, l - token lemma, p - token POS tag, **sem** - set of semantic classes (from (Li and Roth, 2002)) that contain this word, **prox** - set of proximity-based word sets (from (Lin, 2006)) that contain this word, qfw - the QFW detection function. · stands for string concatenation.

extracts features related to the question focus word (QFW). The QFW, which indicates the question emphasis, is usually the head of the first noun or verb in the question, skipping stop words, auxiliary and copulative verbs. For example, for the above question, the QFW is "database". We have implemented the detection of the QFW with 7 surface-text patterns. We detail the feature extraction functions in Table 2.1.

2.2 Passage Processing

Our passage retrieval algorithm is inspired by the query relaxation algorithm of (Paşca, 2003), which adds or drops query keywords depending on the quality of the information retrieved. A novelty of our algorithm is that our implementation is capable to adjust not only the set of keywords used, but also the proximity between the keywords.

The retrieval algorithm consists of two main steps: (a) in the first step all non-stop question words are sorted in descending order of their priority, and (b) in the second step, the set of keywords used for retrieval and their proximity is dynamically adjusted until the number of retrieved passages is sufficient.

Keyword priorities are assigned solely based on their POS tags and lexical context. In descending order of the assigned priority, all non-stop keywords are grouped as follows: (a) words that appear within quotes, (b) proper nouns, (c) numbers, (d) contiguous sequences of nouns and adjectives, (d) contiguous sequences of nouns, (e) other adjectives, (f) other nouns, (g) verbs, (h) adverbs, (i) the QFW, (j) other words. For example, for the question "What is a measure of similarity between two images?", the set of sorted keywords extracted by this algorithm is: {"two", "images", "similarity", "measure"}.

In the second step, the actual passage retrieval is implemented using the following algorithm:

```
(1) retrieve passages using keyword
    set K and proximity $p$.
(2) if number of passages < $MinPass$:
        if $p < MaxProx$
            increment $p$; goto step (1)
        else
            reset $p$;
            drop the least-significant keyword from K;
            goto step (1)
(3) else if number of passages > $MaxPass$:
        if $p > MinProx$
            decrement $p$; goto step (1)
        else
            reset $p$; add the next available keyword to K;
            goto step (1)
(4) return the current set of passages.
```

where the set of keywords **K** is initialized with all keywords with priority larger than the priority assigned to verbs, and the current proximity is initialized with some default value (20 words in our experiments). The algorithm is configured with four parameters: $MinPass$ and $MaxPass$ – lower and upper bounds for the acceptable number of passages (currently 5 and 1000), $MinProx$ and $MaxProx$ – lower and upper bounds for keyword proximity (currently 20 and 40 words).

The actual information retrieval (IR) step of the algorithm (step (1)) is implemented using a Boolean IR system that fetches only passages that contain *all* keywords in **K** at a proximity $\leq p$. Passages do not have a fixed size but rather they extend as long as there are keywords whose distance is smaller than proximity p.

2.3 Answer Extraction

The Answer Extraction (AE) component identifies candidate answers from the relevant passage set and extracts the answer(s) most likely to respond to the user question.

We identify candidate answers with a NERC that uses a battery of one-vs-all SVM classifiers, trained on the CONLL corpus (Sang and Meulder, 2003) . Using the feature set described in (Surdeanu, Turmo, and

Comelles, 2005), this NERC obtains the results reported in Table 2.3 (on the CONLL test corpus).

For the actual AE task we trained a single classifier which decides if a candidate answer is an appropriate answer or not. In Section 3 we show the experimental results when we train the AE classifier using three different ML frameworks: Maximum Entropy, AdaBoost and SVM. All of them give as output a raw activation in the interval [-1..+1], which helps us to sort the candidates not just by a binary choice but by plausibility.

The features used by our AE classifier are grouped in two sets, described below:

Features that measure keyword frequency and density (F_1)

(H1) *Same word sequence* - computes the number of words that are recognized in the same order in the answer context;

(H2) *Punctuation flag* - 1 when the candidate answer is followed by a punctuation sign, 0 otherwise;

(H3) *Comma words* - computes the number of question keywords that follow the candidate answer, when the later is succeeded by comma. A span of 3 words is inspected. The last two heuristics are a basic detection mechanism for appositive constructs, a common form to answer a question;

(H4) *Same sentence* - the number of question words in the same sentence as the candidate answer.

(H5) *Matched keywords* - the number of question words found in the answer context.

(H6) *Answer span* - the largest distance (in words) between two question keywords in the given context. The last three heuristics quantify the proximity and density of the question words in the answer context, which are two intuitive measures of answer quality.

The F_1 feature set is inspired by the heuristics proposed by (Paşca, 2003). However, in (Paşca, 2003) the values of these heuristics are combined in a non linear formula generated by a human expert[1]. This hand-made formula gives state-of-the-art results for a factoid QA. But, the addition of new features is very difficult: for every new heuristic we want to add, the formula has to be readjusted by an expert. Moreover, as the number of heuristics to consider increases, the tuning of the formula becomes more complex. On the other hand, our machine learning approach avoids the intervention of an expert. The addition of new features does not require the intervention of a human expert but a new training of the model. It also reduces adaptation costs of a AE module to different collection types.

All the features are normalized in range [0..1] and then discretized by intervals: [1..1], (1..0,95], (0,95..0,85], etc. For those heuristics whose value depends on the number of keywords (as *same sentence*), we normalized the value dividing it by the number of keywords. For example, if we have two keywords: *president* and *Spain*, and a passage *...the King of Spain met french Prime Minister...*, where only Spain is found: *same sentence* would be evaluated as $\frac{1}{2} = 0.50$.

Features that verify the occurrence of certain patterns (F_2) (the *Entity* underlined corresponds to the candidate answer we are evaluating and *Description* to a part of a sentence):

(P1) $\underline{\textit{Entity}}$ **,|(** *Description* **,|)|.** For example: “Paris, the capital of France,...”.

(P2) Entity **(** $\underline{\textit{Entity of type location}}$ **)** For example: “...inaugurated a new research center in Cluj (Romania)”.

(P3) $\underline{\textit{Entity}}$ *verbToBe Description* For example: “Caesar was the first living man to appear on a Roman Republican coin”.

(P4) $\underline{\textit{Entity}}$ *questionVerb Description* : Where questionVerb is the same verb as in question. For example, “Cervantes wrote Don Quixote in 1605”, for the question “Who wrote ’Don Quixote’?”.

Additionally, four more patterns are implemented with *Description* and *Entity* positions reversed.

Each pattern-based feature measures the *matching* of a *Description* (part of a sentence) against the question. *Matching* is calculated as the average of the number of words

[1]Further references in this paper to heuristics refer to the approach showed by (Paşca, 2003).

Entity Type	Precision	Recall	FB1
PER	92.27%	92.62%	92.44%
LOC	92.95%	94.01%	93.48%
ORG	85.95%	84.86%	85.48%
MISC	89.86%	83.62 %	86.63%
TOTAL	90.71%	89.90%	90.31%

Table 2: Summary of results for NERC task. Based on CONLL evaluation set.

from a question that are contained in the *description*. For instance, if we have three keywords *man*, *gold*, *coin* , and a description *the first living man to appear on a Roman Republican coin*, then *matching* is evaluated as $\frac{2}{3} = 0.66$. The discretization process is the same as for F_1.

We evaluated in the experimental section two different combinations of features: **model 1**, which includes the feature set F_1, and **model 2**, which includes the feature sets F_1, F_2 and a feature that indicates the question type (see section 2.1).

3 Experimental Evaluation

3.1 Data

We trained and tested our QA system using questions and documents from the TREC QA evaluations. We trained both the QP and the AE models with TREC 9-13 questions and answers and used TREC 8 for testing. Only questions identified as locations, persons and organizations are used for AE training. Examples are marked as correct according to the answer patterns provided by NIST. Negative examples are extracted from the passages that the system generates in the PR phase.

3.2 Results

We evaluated the different configurations using the Mean Reciprocal Rank (MRR) score, which assigns to a question a score of $\frac{1}{k}$, where k is the position of the correct answer, or 0 if no correct answer is returned.

We report two sets of experiments. First we evaluate the performance of the AE component alone. For this experiment, we removed questions misclassified by the QP module. The results for the different algorithms and feature sets are summarized in Table 3.2. We differentiate two types of answers: entity answers, which return just the snippet containing the answer named entity, and long answers, which return a context of 250 bytes surrounding the answer named entity. We evaluated three machine learning techniques: AdaBoost, Maximum Entropy and SVM, the latter with the polynomial and linear kernels available from SVM_light. We compare our approach to QA with the state-of-the-art system of (Paşca, 2003)[2].

The results of the machine learning method are statistically similar to the results obtained by the state-of-the-art QA system of (Paşca, 2003). Among the several options tested, Maximum Entropy gives the best results with the same features. We blame the relatively bad performance of AdaBoost and SVM on two known problems of these learning algorithms: (a) AdaBoost (at least in our implementation based on decision trees) is known to suffer from overfitting, and (b) SVM needs fine parameter tuning when the data is not linearly separable. Both these issues are emphasized when the training corpus is small, which is the case in our experiments. On the other hand, Maximum Entropy has better generalization properties than AdaBoost, and, unlike SVM, has virtually no parameters to tune.

In the second experiment, we evaluate the complete QA system, which includes our QP module. We tested our QP on the TREC 8 corpus and measured an accuracy of 88%[3]. We summarize the scores for the complete QA system in Table 3.2. The results are similar to the first experiment, but slightly lower because of the introduced QP errors.

Our results are a proof of concept that a successful AE component that combines both keyword frequency/density and answer patterns can be implemented using machine learning. Such a wide combination of features is virtually impossible in a system driven by expert-designed rules, because the combination of rules is not straight-forward.

The training algorithms were not exhaustively tested for tuning their parameters. Our SVM results are lower than we first expected, but SVM is known to be sensitive to parameter modifications. We believe slightly better results may be obtained, just by readjusting the learning algorithms parameters.

[2]We used our QP module in both systems because: (a) the strategy of (Paşca, 2003) is hard to replicate; (b) our QP approach has higher accuracy.

[3]Using the same evaluation framework as (Li and Roth, 2002), our QA system obtains an accuracy of 85.8%.

Algorithm	Model 1		Model 2	
	Entity	Long Answer	Entity	Long Answer
Maximum Entropy	0.33	0.52	0.34	0.53
AdaBoost	0.31	0.47	0.32	0.47
SVM (best kernel)	0.21	0.36	0.25	0.42
(Paşca, 2003)	0.37	0.52	–	–

Table 3: Summary of MRR results for the different ML algorithms. The (Paşca, 2003) row shows results for our system using the expert-designed answer ranking formula. QP classifies correctly all questions.

Algorithm	Model 1		Model 2	
	Entity	Long Answer	Entity	Long Answer
Maximum Entropy	0.29	0.43	0.31	0.47
AdaBoost	0.29	0.43	0.30	0.44
SVM (best kernel)	0.20	0.32	0.22	0.39
(Paşca, 2003)	0.33	0.47	–	–

Table 4: Summary of MRR results for the different ML algorithms. The (Paşca, 2003) row shows results for our system using the expert-designed answer ranking formula. All experiments are evaluated for the complete QA system.

More precision may be obtained too, with an extended feature set and more training examples.

4 Discussion and Conclusions

We have built a robust QA system with very low NLP requirements and a good precision. We just need a POS tagger, a shallow chunker, a NERC and some world knowledge (NERC and QP gazetteers). Fewer resources make the system easier to be modified for different environments. Low resource requirements also mean fewer processing cycles. This is a major advantage when building real-time QA systems.

We have implemented both Question Processing and Answer Extraction using machine learning classifiers with rich feature sets. Even though, in our experiments performance is not increased above the state of the art, results are similar and several benefits arise from this solution. (a) It is now easy to include new Answer Ranking and Question Processing features. (b) Machine learning can be retrained for new document styles/languages/registers easier than reweighting expert-developed heuristics. (c) We can decide if an answer is more probable to be correct. For instance, the heuristic-based system of (Paşca, 2003) builds a complex formula that allows us to compare (and sort) candidate answers from a single question. But an absolute confidence in the answer appropriateness is not available: for example, a candidate answer can score 100 points but depending on the question it can be a correct answer (and be the first of ranking) or a wrong answer (and be in last positions of the ranking). Our machine learning method is not limited to sorting the answers but it can give us a confidence of how good an answer is. This is very useful for an user interface where incorrect results can be filtered out and the confidence in the displayed answers can be shown beside the actual answer text.

References

Li, X. and D. Roth. 2002. Learning Question Classifiers. *Proceedings of the 19th International Conference on Computational Linguistics (COLING)*, pages 556–562.

Lin, D. 2006. *Proximity-based Thesaurus*. http://www.cs.ualberta.ca/ lindek/downloads.htm.

Paşca, M. 2003. *Open-Domain Question Answering from Large Text Collections*. CSLI Publications.

Sang, Erik F. Tjong Kim and Fien De Meulder. 2003. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. *Proceedings of CoNLL-2003*, pages 142–147.

Surdeanu, M., J. Turmo, and E. Comelles. 2005. Named entity recognition from spontaneous open-domain speech. *Interspeech-2005*.