Learning to Rank for Quantity Consensus Queries

Somnath Banerjee
HP Labs India

Soumen Chakrabarti
IIT Bombay

Ganesh Ramakrishnan
IIT Bombay

ABSTRACT

Web search is increasingly exploiting named entities like persons, places, businesses, addresses and dates. Entity ranking is also of current interest at INEX and TREC. Numerical quantities are an important class of entities, especially in queries about prices and features related to products, services and travel. We introduce Quantity Consensus Queries (QCQs), where each answer is a tight quantity interval distilled from evidence of relevance in thousands of snippets. Entity search and factoid question answering have benefited from aggregating evidence from multiple promising snippets, but these do not readily apply to quantities. Here we propose two new algorithms that learn to aggregate information from multiple snippets. We show that typical signals used in entity ranking, like rarity of query words and their lexical proximity to candidate quantities, are very noisy. Our algorithms learn to score and rank quantity intervals directly, combining snippet quantity and snippet text information. We report on experiments using hundreds of QCQs with ground truth taken from TREC QA, Wikipedia Infoboxes, and other sources, leading to tens of thousands of candidate snippets and quantities. Our algorithms yield about 20% better MAP and NDCG compared to the best-known collective rankers, and are 35% better than scoring snippets independent of each other.

Categories and Subject Descriptors: H.3.3 [**Information Search and Retrieval**]: Retrieval models
General Terms: Algorithms, Experimentation
Keywords: Quantity search, Aggregating evidence from snippets, Learning to rank

1. INTRODUCTION

1.1 Entity search and corroboration

Search engines are getting increasingly sophisticated in extracting and exploiting structured data from unstructured and semistructured Web pages. Most major search engines identify mentions of people, places, organizations, street addresses, ZIP codes, dates, prices, disease names, and several other types of named entities mentioned on the Web pages they crawl.

Entity search has become a standard task in the research community as well. INEX (`http://inex.is.informatik.uni-duisburg.de/`) features a track where the aim is to return entities that satisfy a query. The TREC enterprise track (`http://trec.nist.gov/pubs/trec15/`) includes an expert search task, an important special case of entity search.

Approaches to entity and expert ranking include probabilistic generative models that capture relations between the query, documents, latent topics [9, 2], and lexical proximity between query words and candidate entities [16, 6, 2]. Answers to TREC-style factoid questions (TREC-QA, `http://trec.nist.gov/data/qamain.html`) are frequently named entities.

Corroboration of an entity, mentioned redundantly across multiple sites, often increases ranking accuracy and robustness [7]. Syntactic variations ("Washington" vs. "George Washington") may exist in candidate mentions, and each mention may have a score based on query, language, topic and proximity considerations.

Some researchers [15, 17] have devised type theories with rule systems to conflate syntactic and quantitative variations of candidate answers, aggregate evidence across these variations, and perhaps explain them. Substantial handcrafting of type systems and conflation rules are required in this approach. A recent probabilistic graphical model [12] gives a principled means for learning a model to *collectively rank* candidate entities. However, this technique does not readily apply to quantity search.

1.2 Quantity consensus queries (QCQs)

In this paper we focus on *quantity search*, an important special case of entity search. A quantity may be a unitless number or have an associated unit like length, mass, temperature, currency, etc. TREC-QA 2007, 2006, and 2005 have 360, 403 and 362 factoid queries, of which as many as 125, 177, and 116 queries seek quantities. As against "spot queries" seeking unique answers like date of birth, we are specifically interested in what we call *quantity consensus queries* (QCQs), where there is uncertainty about the answer quantity ("driving time from Paris to Nice" or "battery life of Lenovo X300"). TREC-QA 2007, 2006, and 2005 have at least 61, 39 and 28 such queries. To learn a reasonable distribution over the uncertain quantity, the user may need to browse thousands of pages returned by a regular search engine. A QCQ system reduces this cognitive burden by zooming down from document to snippet to quantity level. QCQ engines can also support sites that offer comparison of prices and features related to products, services and travel.

In the information extraction, integration and warehousing literature, a *curate-and-query* approach is popular; it assumes the existence of entity and relationship extractors [1, 3] for limited domains, which populate (possibly probabilistic) relational databases [4, 20]. We argue that open-domain ad-hoc QCQs cannot leverage the curate-and-query strategy, because the queries are too diverse and the sources are too unstructured for a priori schema design or information extraction. Our hypothesis is that some combination of string-oriented IR and structured aggregation is essential at query time.

1.3 Our contributions

We introduce QCQs (Section 2) and give novel algorithms that aggregate evidence in favor of candidate quantities and quantity intervals from *snippets* in a collective and corroborative fashion, without attempting deep NLP on snippets.

SIGIR'09, July 19–23, 2009, Boston, Massachusetts, USA.

`+giraffe, +height; foot`	
	La Giraffe was small (approx. **11 feet** tall) because she was still young, a full grown giraffe can reach a height of **18 feet**.
	Giraffe Photography uses a telescopic mast to elevate an 8 megapixel digital camera to a height of approximately **50 feet**.
	The record height for a Giraffe unicycle is about **100 ft** (30.5m).
`+weight, weigh, airbus, +A380; pound`	
	Since the Airbus A380 weighs approximately **1,300,000 pounds** when fully loaded with passengers ...
	The new mega-liner A380 needs the enormous thrust of four times **70.000 pounds** in order to take off.
	According to Teal, the **319-ton** A380 would weigh in at **1,153 pounds** per passenger
`far +raccoon relocate; mile`	
	It also says – unnervingly – that relocated raccoons have been known to return from as far away as **75 miles**.
	Sixteen deer, 2 foxes, one skunk, and 2 raccoons are sighted during one **35 mile** drive.
	One study found that raccoons could move over **20 miles** from the drop-off point in a short period of time.

Figure 1: QCQs with snippets (matches underlined; quantities in boldface, good, maybe, bad).

For a given query, the ith snippet is a segment of text tokens, centered around the mention of a quantity x_i. A *quantity* is a number or a range accompanied by an (optional) unit of measurement. To the left and right of the central quantity mention are other *context* tokens of the snippet.

As baseline, we first consider (Section 4) algorithms that learn to rank items (documents or snippets) represented as feature vectors [10, 11, 22] (for a comprehensive list visit `http://research.microsoft.com/users/LETOR/`). An item (here, a snippet) is usually represented as a feature vector $z_i \in \mathbb{R}^d$ in response to a query. In our case, z_i will encode the presence of query words in the snippet context, lexical proximity between query words and quantity x_i, and rarity of matched query words in the corpus (IDF). Using manually-provided snippet relevance labels $y_i \in \pm 1$, these algorithms learn a *model vector* $w \in \mathbb{R}^d$ such that the score of the ith test snippet is $w^\top z_i$, and snippets are then sorted by decreasing score. We show that scoring using z_i performs poorly, because z_i by itself is a very noisy relevance signal.

We then evaluate a recent technique [21] that aggregates evidence across snippets i, j only if x_i, x_j match exactly. This fails in the face of close but not identical quantities in dominant clusters. Next we adapt a graph Laplacian smoothing technique [12, 18] that balances between individual snippet score evidence $w^\top z_i$ and quantity proximity, say, $|x_i - x_j|$. This formulation cannot ignore quantity proximity among irrelevant snippets, and gives only modest gains.

These trials and observations prompt us to propose (Section 5) new scoring mechanisms for entire *intervals* of x values, instead of individual snippets, as was done in prior work. We show how to aggregate snippet scores into candidate interval scores, and then pick the best intervals. This dramatically boosts accuracy.

In Section 6, we give another algorithm: it represents an interval I with novel feature vectors $\hat{z}_I$, where some features are aggregated from snippet-level scores $w^\top z_i$. Note that i indexes individual snippets and I represents an interval. We use max-margin methods [10] to learn a "stacked" model $\hat{w}$. During testing, $\hat{w}^\top z_I$ is used to sort candidate intervals.

Our stacked ranker further enhances accuracy compared to interval scoring using w alone. It achieves over 20% relative improvements in snippet-level MAP and NDCG compared to Laplacian smoothing, which in turn is 10–15% better than independent snippet ranking. We compare favorably with the best TREC-QA participants wrt precision-at-1. We also present a new way to evaluate sequences of quantity intervals, as against snippet lists.

Providing snippet labels y_i is more tedious than providing ground truth x_i values per query. In Section 7, we propose a very simple alternative to training w and $\hat{w}$ using only ground truth x_i, with a very small drop in quality.

Given the extreme diversity and noise in snippets, it is astonishing that clear and often correct consensus can be mined without the help of deep NLP, even for completely ad-hoc queries.

2. TERMINOLOGY

2.1 Query

A QCQ has two main parts: a set of words or phrases, and a quantity type specifier. Some words or phrases may be marked compulsory with a prefixed '+'. The latter may be unitless, if a count is desired, or have an unit. Some example QCQs are shown in Figure 1. As with ordinary Web queries, the onus of getting better snippets, through the use of '+' and phrases, lies with the user.

A third optional component of QCQs that gives additional control is a user-defined *relative width* parameter r, where $0 \leq r \ll 1$, meaning that the user is looking for a quantity interval $[x, x']$, such that $x' \leq (1+r)x$, which has strong collective evidence from snippets. r is necessarily user-defined: a QCQ about Olympic record times has a fundamentally different expectation of precision compared to a QCQ about the distance between the Sun and Pluto. Only the user can provide that domain knowledge. In practice, a large number of QCQs run well with a default setting like "$r = 0.05$". In any case, r is an *upper bound* on the relative width, and our system will tighten the interval if it can.

2.2 Snippet (x_i, z_i)

A snippet is a suitably large window of tokens around a candidate quantity which matches the unit specified in the QCQ. A quantity scanner (Section 3.1) identifies token segments that express quantities. The quantity, including unit, is called x_i for the ith snippet for a given query. The surrounding text is turned into a suitable feature vector representation $z_i \in \mathbb{R}^d$. (z_i depends also on the query.)

The design of z_i must consider the proximity between the central quantity mention to snippet tokens that match query tokens, and is described in detail in Section 3.2. Any snippet that has one or more token matches with the query is potentially a relevant snippet, and its quantity a *candidate quantity*. Some sample relevant and irrelevant snippets for the above QCQs are shown in Figure 1. The snippets make clear the great variety of contexts in which plausible quantities appear close to significant query words.

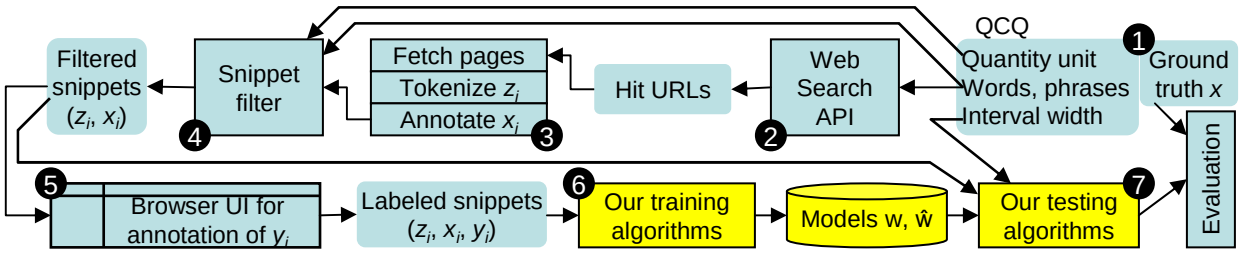

Figure 2: Sketch of our QCQ system prototype. Processing stages are numbered from 1 onward.

2.3 Consensus

As is clear from the examples, QCQs are characterized by an absence of an absolute or single truth. Our first impulse was to model the quantity of interest as a random variable, and build a system to return a distribution over it. But the event space is too complex: it involves natural language usage and extraction accuracy, among other uncertainties. We therefore avoid generative models for quantities, and explore discriminative, collective ranking techniques for snippets. Informally, a *consensus interval* is a tight range $[x, x']$ of quantities that enjoys strong collective support from high-scoring snippets. We will give more precise proposals in Sections 5 and 6. There, we will see that this simple notion of consensus performs very well.

To be fair, consensus is not the only form of useful aggregation; in some cases, it may be limiting or misleading. E.g., plutonium has multiple isotopes with diverse half lives, and a name may refer to many people with diverse birth years. Our QCQ system performs reasonably despite such ambiguity, because it reports (snippets from) not one but a number of top-scoring x-intervals. Time-variant quantities offer another challenge. E.g., the QCQ `+"bill gates", assets, worth; USD` may give an outdated answer, depending on Web coverage. A complete solution would require "carbon dating" each snippet, which appears even more challenging than reliable timestamping of whole Web pages. The causes of multi-valued answers have been analyzed in some detail [15, 17].

3. QCQ SYSTEM AND TESTBED

In this section, we give an overview of our QCQ system, sketched in Figure 2[1]. We will first describe the modules for annotating x_i (Section 3.1) and turning snippet text into z_i (Section 3.2). Then we will describe how we collected queries and ground truth y_i for candidate snippets..

3.1 Quantity scanner for annotating x_i

A *quantity scanner* annotates character spans that are likely to be quantity mentions, which come in diverse forms. Some have unit prefixes, like currency symbols. Some have unit suffixes, like scientific measures. Some have exponent modifiers, like "10 million liters" or "€50 million". Units are expressed diversely, e.g., '$' vs. USD, 'm' vs. meter vs. metre. Even the numerals are written in diverse styles. Scientific quantities may be written without commas, commas after every third digit, or at irregular spacing, as in "Rs 1,20,000". There may be spurious spaces before or after commas. Periods may end sentences or be decimal points. Very large or small quantities may be written in mantissa-exponent form. Small numerals like 1, 2, 30 may be written as words. 1889 might be a unitless count or a year. '$' may indicate different currencies. x_i may also be a range, e.g., **10–20 feet**.

We used the rule-based JAPE engine, which is part of the well-known GATE NLP package (`http://gate.ac.uk/`). We compiled about 150 rules covering mass, mileage, power, speed, density, volume, area, money, time duration, time epoch, temperature length and so on. Augmenting our rule base to capture more types of quantities should be straightforward. Manual spot checks on our annotator led to estimates of precision, recall and F1 as 0.92, 0.97, 0.95. Luckily, ranking intervals using consensus is robust to this small rate of scanner glitches.

Unit normalization: In the example QCQs above, each query has an associated specific unit (unless the answer is a count). In a deployed system, more generic units should be allowed, such as *length* in place of *mile* or *km*, or *time interval* in place of *hour* or *year*. This would also assist collecting consensus across candidate quantities expressed in different units. Our prototype does not handle this issue, except identifying different standard forms of a unit (e.g. foot, feet, ft), but it can be added on easily.

3.2 Feature vector design for z_i

We defined two families of features on (the query and) snippet text: first, standard vector-space ranking features [14, 13], and second, features that encode lexical proximity between query word matches and quantity tokens [16, 6, 2].

3.2.1 Standard ranking features

Each snippet was characterized by the tokens in five *fields* F: snippet, a window of 10 sentences above and below the snippet, the text of the page from where the snippet is originated, the HTML title of the page, and the URL of the page. For each of the five fields F, three features were added to feature vector z_i:

TFSum: $\sum_{t \in q \cap F} \mathrm{TF}(t, F)$
IDFSum: $\sum_{t \in q \cap F} \mathrm{IDF}(t)$
TFIDFSum: $\sum_{t \in q \cap F} \mathrm{TF}(t, F)\mathrm{IDF}(t)$

$\mathrm{TF}(t, F)$ is the term frequency of t in F and $\mathrm{IDF}(t)$ is the standard IDF of t with respect to a reference corpus (union of all documents over all queries). In addition, we used:

- Jaccard similarity between query and snippet tokens.
- Number of tokens in the snippet.

3.2.2 Lexical proximity features

Guided by earlier work on locality or proximity based ranking [8, 6, 16, 2], we defined the *proximity* between the mention of quantity x_i and a query token match t in its vicinity as the *reciprocal of the number of tokens between the mention of x_i and t* (zero if no t exists).

Queries have a variable number of tokens. Therefore we define four proximity features aggregated over query tokens:

[1]Details at `http://www.cse.iitb.ac.in/~soumen/doc/QCQ`

- Maximum proximity of x_i to any query token.
- Proximity of x_i to the rarest (largest IDF) query token.
- Proximity of x_i to the smallest IDF query token.
- IDF-weighted average of proximity to all query tokens.

The weights in w corresponding to these proximity features were among the highest when w was learnt using RankSVM [10]. To keep our system robust and scalable, we avoided deeper NLP techniques like learning to spot relations from dependency parse trees.

Altogether, we used 21 features: 4 proximity, 5×3 similarity features and 2 other features.

3.3 QCQs with ground truth

We collected 162 QCQs from diverse sources. Each QCQ q was collected along with ground truth quantity set X^q. Most X^qs contained multiple values or ranges. Unless noted otherwise, we report performance on the union of these QCQs.

Infobox: We created 40 QCQs by sampling Wikipedia Infoboxes for numeric attributes of Wikipedia entities.

TREC-QA: We chose TREC-QA queries that had non-unique quantity answers: 16 from TREC-QA 2004 and 61 from TREC-QA 2007.

Misc.: 9 queries were contributed by W&M [21]. 36 QCQs were contributed by volunteers, who found ground truth X^q through careful Web search.

Growing our QCQ set is limited only by snippet-labeling effort (described next).

3.4 Snippet label y_i collection

We used Web search APIs to collect snippets. Unlike QA-oriented text indices, major Web search APIs do not allow us to ask for documents containing, say, a distance in feet within 20 tokens of the word *elephant*. This necessitated a two-step filtering approach. In the first step, we sent words, phrases and unit names in the QCQ to the engine. Response URLs were fetched, tokenized, and quantities annotated. Quantities that matched the QCQ unit, and were within one sentence (or a maximum token window) from a query word were retained, with their snippet context. We retained a total of about 15,000 snippets over 162 queries. We will make this data available in the public domain.

For training, a selection of 100 snippets per QCQ were presented, using a browser-based GUI, for manual labeling of $y_i \in \pm 1$, the relevance of snippet i. Six volunteers, including the authors, annotated the snippets. There were (infrequent) inconsistencies between the contributed answer quantities and y_i labels. I.e., snippets with quantities not in the ground truth ranges were sometimes marked relevant, mostly because the Web has a more up-to-date ground truth. We did not attempt to make these consistent, insisting that a robust algorithm must take this in stride.

3.5 Response and comparative evaluation

QCQ systems may return a ranked list of snippets, with the quantities highlighted (Figure 1). The advantage is that the user can glance over and judge the snippets directly. Traditional criteria [13], such as Mean Average Precision (MAP) or Normalized Discounted Cumulative Gain (NDCG) can then be used directly. (Mean Reciprocal Rank or MRR is not appropriate for QCQs because it does not give credit for comprehensive coverage of consensus values.)

Alternatively, to display many promising quantities within scarce real estate, QCQ systems may report a list of x-intervals, each subject to the user-provided relative width constraint. Evidence snippets can be shown if an interval is clicked. Evaluating a list of intervals, or comparing a system that ranks snippets with one that ranks intervals, are new challenges. We will discuss these in Sections 5 and 6.

4. PRIOR APPROACHES AND INSIGHTS

We describe existing approaches that can be adapted for QCQs, culminating in a comparison shown in Figure 5.

4.1 Using Web search directly

The minimal baseline (that any useful QCQ system must beat) is to send QCQ words/phrases to a search engine, get the top snippets, scan them for qualifying quantities with proper units, and list them if they appear within a stipulated distance of at least one query token. A listed quantity x is judged correct if it matches (or is contained by) a ground truth quantity (or interval).

Such snippet-level evaluation gives very poor MAP and NDCG (below 0.15), partly because search engines have no mechanism to promote to top ranks those snippets that contain quantities of specified types and query words. We can be generous and give credit for unsupported but correct quantities anywhere on the pages (not just reported snippets), which is what we show in Figure 5. We use two major engines (called Web1 and Web2). Our algorithms are better at promoting relevant snippets to top positions, comfortably beating the generous evaluation of Web search engines.

4.2 Snippet-level RankSVM

We tried several techniques for learning [10, 22, 13] a snippet-level w given snippets (z_i, y_i) (x_i is ignored here), with the score $w^\top z_i$ used for ranking snippets. We found standard pairwise RankSVM [10] (formulation given below) as good as direct optimizers of MAP [22] or NDCG [5].

$$\min_{w,\xi} \tfrac{1}{2} w^\top w + C \sum_{i:y_i=1} \sum_{j:y_j=-1} \xi_{ij} \quad \text{subject to} \qquad (1)$$

$$\forall i \text{ s.t. } y_i = 1, \forall j \text{ s.t. } y_j = -1, \begin{cases} \xi_{ij} \geq 0 \\ w^\top z_i + \xi_{ij} \geq w^\top z_j + 1 \end{cases}$$

$\sum_{i,j} \xi_{ij}$ upper bounds the number of pair preferences violated and C balances between violations and $|w|$. Figure 5 compares the accuracy of various baseline algorithms. (For all RankSVM-style learning algorithms in this paper, five-fold cross validation was used and the best value of C in (1) was picked from among $\{10^{-3}, 10^{-2}, .1, 1, 10\}$.)

Figure 5 shows that RankSVM is generally better than Web1 and Web2. As for MAP, remember that Web1 and Web2 are given massive advantage while RankSVM snippets are evaluated stringently. However, a closer look at RankSVM (next) provides key actionable insight.

4.3 Vertical bands in $w^\top z_i$ vs. x_i scatter

RankSVM considers only $w^\top z_i$ scores, but how do these relate to corresponding x_is? Figure 3 plots scatters of $w^\top z_i$ (y-axis) against x_i (x-axis) for three representative queries. For visual uniformity across queries, both axes have been scaled to $[0, 1]$. Snippets are also called "points". If optimization (1) were perfect, all good points would lie higher along the y-axis than all bad points. This is rarely the case: although z_i was designed with considerable care, de-

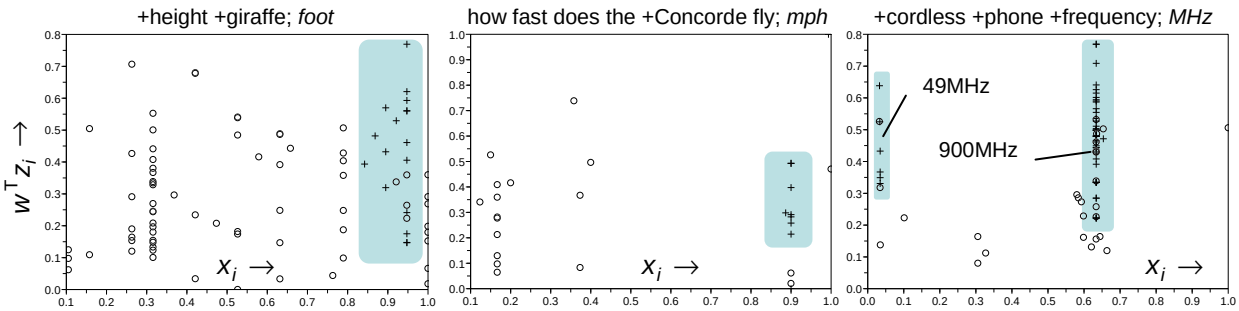

Figure 3: Scatter of $w^\top z_i$ against x_i for representative QCQs; relevant (irrelevant) points marked '+' ('○').

cent separation between relevant and irrelevant snippets is never achieved on the basis of $w^\top z_i$ alone.

But the scatter plots also show a valuable clue: *relevant points often cluster in vertical bands.* From Figure 3 and the simplified sketch in Figure 4, it seems that for each query, one of few *upward-open rectangular strips* capture most good snippets with very few bad snippets.

It is natural to ask at this point why we cannot use decision trees (which naturally find rectangle discriminators) or SVMs with nonlinear kernels. The reason is that the width and location of the semi-open rectangles (equivalently, parameters of non-linear kernels) change from query to query. Parameters learnt by decision trees or nonlinear SVMs will not generalize across diverse queries. We need a more non-parametric approach.

4.4 Wu and Marian's system (W&M)

A first approach to integrating x from snippets is to take weighted majority votes, similar to exploiting redundancy in QA. W&M accumulates a score for each distinct x from snippets where x occurs. The snippet score is determined by the following considerations:

- It decays geometrically with the rank assigned by the search engine to the source page.
- It decreases reciprocally with the number of candidate quantities on the source page.
- It decreases exponentially with the number of duplicate/mirror pages and pages from the same domain. (Search engines already enhance diversity and eliminate duplicates, so this rarely fires.)
- It decreases reciprocal to the shortest distance between the quantity and a query token (lexical proximity).

Score aggregation happens only on exact equality of x. Figure 5 shows that W&M is consistently worse than RankSVM. Often, relevant snippets are found at quite poor ranks, because the whole-page ranking imposed by Web1 and Web2 are often not suited for QCQs. Recall again that Web1 and Web2's accuracy may be substantial overestimates.

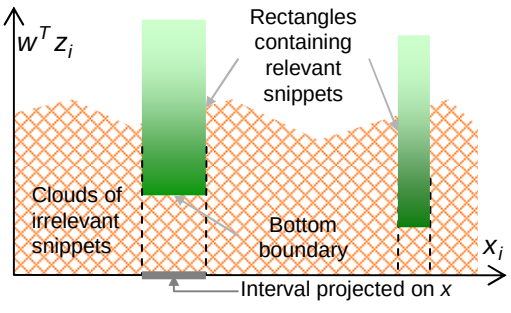

Figure 4: Our proposed "hypothesis class" of semi-open rectangles.

4.5 Laplacian smoothing

A second way to combine x_i and $w^\top z_i$ is via a graph Laplacian approach [18]. Each snippet is made a node in a graph $G = (V, E)$. Each node/snippet has an associated feature vector z_i as before, inducing a (noisy) *local score* $w^\top z_i$. Meanwhile, the x_i values at nodes are used to define edges weights $R(i, j)$, inversely related to $|x_i - x_j|$.

The formulation seeks a model w while assessing a loss $(f_i - w^\top z_i)^2$ for deviations between *final scores* f_i and local scores, and a roughness loss $\sum_{\{i,j\}\in E} R(i,j)(f_i - f_j)^2$, where $f \in \mathbb{R}^{n\times 1}$ is the column vector of final scores. Finally, there is the usual training loss if the final score of a good snippet is less than the final score of a bad snippet. Training involves solving a quadratic program with linear constraints [18].

The design of edge weights R critically determines the algorithm, but there is no generic guideline. We tried the following reasonable definitions:

$x_i = x_j$ **equality:** Following W&M's majority semantics, we define $R(i, j) = 1$ if $x_i = x_j$ and 0 otherwise.

$|x_i - x_j|$ **distance:** $R(i, j) = \max\left\{0, 1 - \frac{|x_i - x_j|}{|x_i| + |x_j|}\right\}$

$|x_i - x_j|$ **decay:** $R(i, j)$ is defined as $\exp(-s\|x_i - x_j\|)$ or $\exp(-s(x_i - x_j)^2)$, where s is a tuned **spread parameter** (inverse variance).

Snippet cosine: Following the pseudorelevance feedback [18] setting, we ignore x_i, x_j and use cosine similarity between the text of snippets i and j as $R(i, j)$. Snippet text is represented as a binary vector over token space. The intuition is that if snippet texts for i and j are similar, they should have similar score.

Figure 5 summarizes accuracies of all approaches discussed thus far. Laplacian smoothing with the "decay" option gives modest gains over Web1, Web2, RankSVM, and W&M. The gains are limited by two factors. First, the Laplacian formulation assesses the roughness penalty on *all* edges, even those between snippets putatively labeled irrelevant. For

		MAP	NDCG@1	NDCG@5	NDCG@10
	Web1	0.375	0.338	0.362	0.380
	Web2	0.350	0.413	0.357	0.377
	RankSVM	0.369	**0.450**	0.412	0.406
	W&M	0.306	0.247	0.303	0.322
Laplacian	Equality	0.384	0.369	0.353	0.382
Laplacian	Distance	0.407	0.413	0.401	0.420
Laplacian	Decay	**0.421**	0.433	**0.422**	**0.435**
Laplacian	Cosine	0.375	0.438	0.396	0.405

Figure 5: Initial results (bold $\Longrightarrow$ max in column).

```
1: inputs: snippet set S with x_i and w^T z_i values, interval
   width tolerance parameter r
2: sort snippets S in increasing x_i order
3: for i = 1, ..., n do
4:   for j = i, ..., n do
5:     if x_j < (1 + r)x_i then
6:       let I = [x_i, x_j]
7:       merit ← GetIntervalMerit(S, I)
8:       maintain intervals with top-k merit values
9: for surviving intervals I in decreasing merit order do
10:   present snippets in I in decreasing w^T z_i order
```

Figure 6: Interval merit enumeration.

QCQ, we should favor smoothness of f_i only among relevant snippets. Second, there is no ready way to tune the width parameter s reliably across diverse queries and associated quantities. Our algorithms get around these issues.

5. LISTING AND SCORING INTERVALS

Instead of scoring and ranking snippets, we shift our focus to quickly enumerating and scoring rectangular regions as shown in Figure 4. We begin with searching for the position and width of a promising rectangle on the x-axis, i.e., searching over intervals $I = [x, x']$, with $x' \leq (1 + r)x$ as specified in Section 2.1. We will overload I to also mean a *set* of snippets. A snippet $s_i = (x_i, z_i)$ is said to belong to I if $x_i \in I$. In case a snippet mentions a range (such as **10–20 feet**), the snippet belongs to I if the range is contained in I.

For a query q with n_q snippets, there are at most $\binom{n_q+1}{2}$ functionally distinct (in terms of the snippets they contain) intervals on the x axis. Some of these intervals $I = (x, x')$ are too wide ($x' > (1 + r)x$) and can be discarded. Usually $r \ll 1$, so the enumeration of valid candidates $I \in \mathcal{I}_r$ can be done efficiently using a left-to-right sweep that takes close to linear time in practice. For simplicity Figure 6 shows a naive $O(n_q^2)$ enumeration of intervals.

Figure 4 suggests that we should also search over all possible bottom boundaries of I. In practice, this makes negligible difference. Our results in Section 7 may explain why this is the case.

5.1 Merit functions $GetIntervalMerit(S, I)$

As we enumerate over intervals I, we need to use the signal from $w^\top z_i$ for $i \in I$ and potentially $i \notin I$, to evaluate $GetIntervalMerit(S, I)$. If there is any useful signal in $w^\top z_i$, we should prefer intervals I such that points in I have generally larger values of $w^\top z_i$ than points not in I. Accordingly, we provide three choices of merit (to maximize over I):

$$\sum_{i:x_i \in I} w^\top z_i \qquad \text{(Sum)}$$

$$\sum_{i \in I} \sum_{j \notin I} (w^\top z_i - w^\top z_j) \qquad \text{(Diff)}$$

$$\sum_{i \in I} \sum_{j \notin I} \max\left\{0, w^\top z_i - w^\top z_j\right\} \qquad \text{(Hinge)}$$

Observe that terms in (Diff) can be positive or negative; favorable and unfavorable score pairs can cancel out. This is prevented in (Hinge). In machine learning one *minimizes* hinge *loss* rather than *maximize* hinge *gain*, but in QCQ, the former leads to tiny proposed relevant clusters that are often incorrect.

5.2 Snippet-level evaluation experiments

We compare three algorithms: the best two approaches from Figure 5 (RankSVM and Laplacian Decay) and interval merit enumeration (for which the snippet-level model w was trained using RankSVM). For MAP (Figure 7), we vary interval width tolerance r (shown as a percentage). For NDCG (Figure 8) we hold $r = 8\%$ and report NDCG at ranks $1 \ldots 10$. Note that $r = 0$ means an interval of width zero, but this can contain multiple snippets if they mention the exact same quantity. RankSVM and Laplacian Decay do not depend on r.

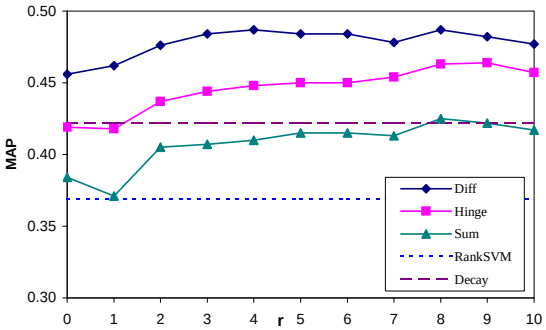

Figure 7: Interval merit evaluation (MAP).

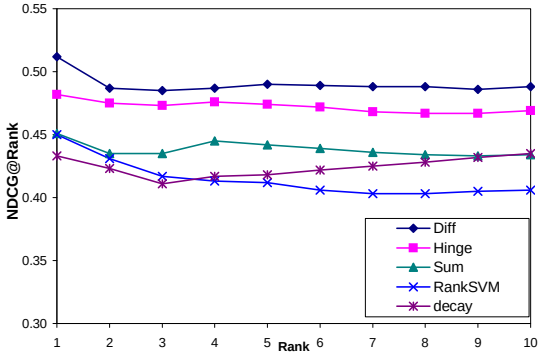

Figure 8: Interval merit evaluation (NDCG).

Interval merit beats all baselines. First, interval enumeration with (Diff) beats all other approaches by a wide margin. Interval enumeration with (Hinge) is second, still beating all others.

Effect of r. (Diff) and (Hinge) show significant boost in accuracy as r is increased beyond 0. (Diff) is stable between $r = 3\%$ and 9%. This is direct evidence that robust aggregation over x_i values is critical to success.

(Diff) *better than* (Hinge). Occasionally, avoiding deep NLP leads to systematic pollution from irrelevant but dense intervals. E.g., for the QCQ `+giraffe +height: foot`, an irrelevant cluster (as per predominant human interpretation) develops around 6 feet thanks to snippets like this: "newborn *giraffe* calves begin their lives by falling from a *height* of **6 feet**", "A young *giraffe* has to survive a fall of **six feet**", or "A *giraffe*'s legs alone are taller than many humans—about **6 feet**". These intervals have a lower average $w^\top z_i$ and (Diff) reveals this better than (Hinge).

6. LEARNING TO RANK INTERVALS

In the previous section we proposed a way to score intervals, based on aggregating $w^\top z_i$ scores of snippets inside and outside the intervals. In this section we design a learner that directly learns to rank intervals instead of individual

snippets.

As in Section 5, we will use relative tolerance r to define $\mathcal{I}_r$, the set of candidate intervals satisfying r. We already know that $|\mathcal{I}_r| = O(n_q^2)$.

Every candidate interval $I \in \mathcal{I}_r$ will be represented by an *interval feature vector* $\hat{z}_I$. The interval ranker will learn a corresponding scoring model vector $\hat{w}$.

6.1 Interval features $\hat{z}_I$

Unlike in snippet-level RankSVM, we are at liberty to define *collective* features of intervals, rather than just aggregate $\{z_i : x_i \in I\}$, in simple ways as in Section 5. Specifically, a simple average of feature vectors may fail to capture certain significant clustering in the z_i space. There may be much stronger clues to guess how good an interval is.

For example, an interval is good if most of the points in the interval are relevant to the query, if the interval has high merit (as defined in Section 5.1) and most of the points in the interval have consensus on a quantity or there are relative few distinct quantities. We capture these clues by designing a set of additional features that are collective across an interval. We call them *interval features*:

1. Whether all snippets in I contain some query word
2. Whether all snippets in I contain the minimum IDF query word
3. Whether all snippets in I contain the maximum IDF query word
4. Number of distinct words found in snippets in I
5. Number of words that occur in all snippets in I
6. One minus the number of distinct quantities mentioned in snippets in I, divided by $|I|$
7. Number of snippets in I, divided by n_q
8. Three features corresponding to the three merit functions defined in Section 5.1, which require w to compute.

Apart from the above interval features we also append to $\hat{z}_I$ the vector average of the feature vectors z_i with $i \in I$.

6.2 Interval relevance and preferences

Recall that we want to learn to compare intervals, but our ground truth y_i is collected over snippets. The next piece is to define a relevance score over each interval $I \in \mathcal{I}_r$. We assign a relevance score to an interval I based on the fraction of relevant snippets in I. I.e., if I has n_I^+ relevant snippets and n_I snippets overall, then its relevance score is defined as n_I^+/n_I. Thus, snippet-level y_i labels determine the relevance score of intervals.

For two intervals I and I', if the relevance score of I is larger than that of I', we assert a pairwise preference $I \succ I'$ between the intervals. These interval comparisons will replace individual snippet comparisons in (1). (Other algorithms [22, 5, 13] may be used in place of RankSVM.)

Initial experience with the algorithm shown in Figure 9 suggested that we were generating too many preference pair constraints based on insignificant interval relevance differences. We improved both training speed and accuracy by discretizing interval relevance to an ordinal scale of 0–10. In other words, the relevance of an interval was defined as $\lfloor 10 n_I^+/n_I \rfloor$. We tried between 5 and 10 ordinal levels and the accuracy was not very sensitive to the number of levels.

Suppose the interval ranker learns model $\hat{w}$. Given a test query, $\mathcal{I}_r$ is enumerated as before. Then the intervals in $\mathcal{I}_r$ are ranked by decreasing $\hat{w}^\top \hat{z}_I$. If a snippet list must be provided, we run down the intervals in decreasing $\hat{w}^\top \hat{z}_I$ order, and order snippets within each interval using snippet score $w^\top z_i$.

1: **inputs:** snippets s_i with labels y_i, tolerance r
2: **for** each interval $I \in \mathcal{I}_r$ **do**
3: compute the relevance of I using snippet labels y_i
4: compute feature vector $\hat{z}_I$
5: generate interval pair preferences $I \succ I'$
6: set up a RankSVM problem involving intervals:

$$\min_{\hat{w},\hat{\xi}} \tfrac{1}{2}\hat{w}^\top\hat{w} + C\sum_{I\succ I'} \hat{\xi}_{I,I'} \quad \text{s.t.} \qquad \text{(IntervalRank)}$$

$$\forall I \succ I': \quad \hat{w}^\top \hat{z}_I - \hat{w}^\top \hat{z}_{I'} \geq 1 - \hat{\xi}_{I,I'}; \quad \hat{\xi}_{I,I'} \geq 0$$

7: train using RankSVM to get $\hat{w}$
8: **return** $\hat{w}$

Figure 9: Interval training algorithm.

6.3 Experimental results

6.3.1 Snippet-level comparison

We compare the best algorithm from Section 5, viz., (Diff) merit score for intervals, against the (IntervalRank) algorithm presented in this section.

Figure 10 compares MAP obtained by IntervalRank vs. Diff as width tolerance r is varied. IntervalRank is better, reaching a MAP of 0.511 against 0.421 by Laplacian smoothing and 0.369 by RankSVM. The story with NDCG (Figure 11) is almost similar, the gains increasing with rank. IntervalRank achieves NDCG@10 of 0.513 against 0.435 by Laplacian smoothing and 0.406 by RankSVM.

We did an ablation study by removing one feature from all $\hat{z}_I$ at a time. The maximum MAP reduction was for feature #6. This shows that aggregating evidence from snippets supporting intervals is critical.

6.3.2 Comparison with TREC-QA participants

Direct comparison is impossible: the corpora are different. In terms of precision-at-1, for our sample of TREC-QA 2007,

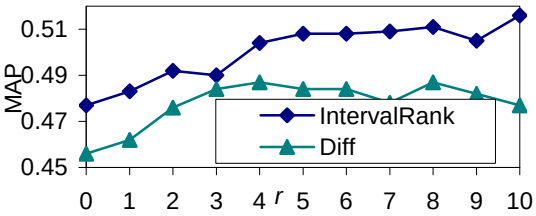

Figure 10: Comparison of Merit-Diff and interval ranking algorithms (MAP).

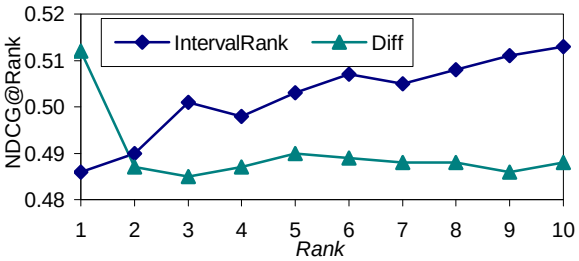

Figure 11: Comparison of Merit-Diff and interval ranking algorithms (NDCG).

	# Intervals →				
Algo, measure	1	2	3	4	5
IntervalRank recall	0.521	0.581	0.637	0.647	0.685
Lapl. decay recall	0.510	0.569	0.614	0.634	0.655
RankSVM recall	0.458	0.514	0.554	0.596	0.618
IntervalRank prec	0.443	0.432	0.416	0.388	0.371
Lapl. decay prec	0.382	0.367	0.350	0.330	0.316
RankSVM prec	0.330	0.312	0.298	0.294	0.284

Figure 12: Interval-oriented evaluation.

we are second-best. For our sample of TREC-QA 2004, we are at rank 5 out of 63 teams. While not very meaningful for QCQ, this shows that our system is competitive wrt precision-at-1.

6.3.3 Interval-oriented evaluation

Our algorithms rank intervals, but to evaluate them wrt snippet-level y_i ground truth, we iterated through intervals by decreasing $\hat{w}^\top z_I$, listing snippets $i \in I$ by decreasing $w^\top z_i$. Snippet-level NDCG or MAP is suitable when users inspect snippet lists [19]. If a QCQ system presents a list of intervals, the user may inspect at most a small number of evidence snippets per interval, so snippet-level MAP or NDCG may not accurately reflect cognitive burden. We propose recall and precision criteria that recognize an interval, not a snippet, as a unit of attention. Suppose there are n^+ snippets marked relevant for a QCQ, and our algorithm A outputs $I_1, \ldots, I_m$, where I_j contains n_j snippets, of which k_j are good. The *interval-oriented precision* of A at interval rank j is defined as $(k_1 + \cdots + k_j)/(n_1 + \cdots + n_j)$. The *interval-oriented recall* is defined as $(k_1 + \cdots + k_j)/n^+$. To compare with a snippet-listing algorithm A' we simply line up the first $n_1 + \cdots + n_j$ snippets, assume that A' reported intervals $I'_1, \ldots, I'_m$, and evaluate similar to $I_1, \ldots, I_m$. Note that IntervalRank cannot cheat at recall using arbitrarily large r, because precision will plummet. Results in Figure 12 show that collective interval scoring and presentation can increase both recall and precision, particularly for the top few intervals. Laplacian decay is between RankSVM and IntervalRank.

7. QUANTITY-IMPUTED LABELING

We have assumed throughout that the label y_i is known for each training snippet ("complete" supervision). However, it is much more natural and efficient to train a QCQ system based on ground truth quantity set X^q and z_i. Another advantage of this form of "partial" training is that we can semi-automatically glean training data from social media, such as Wikipedia Infoboxes.

Suppose we sloppily impute y_i values using X^q: any snippet with $x_i \in X^q$, or contained in a range in X^q, is considered relevant. These imputed $\tilde{y}_i$s may conflict with "true" y_is (if available). How drastically might $w, \hat{w}$ deteriorate because of using $\tilde{y}_i$s to train our system in place of y_is?

We sampled queries leading to 14,562 y_i-labeled snippets. $\tilde{y}_i$ gave only 571 false positives and 395 false negatives. These modest fractions may explain why modeling the bottom boundary of rectangles in Figure 4 did not make a significant difference. Figure 13 shows the effect of imputed training on test MAP score. The drop in test accuracy is very mild. Our algorithm continues to beat all baselines.

	y_i known	y_i imputed
RankSVM	0.369	0.361
Merit-Diff	0.487	0.475
IntervalRank	0.511	0.480

Figure 13: Effect of imputation on test MAP.

8. CONCLUSION

We introduced QCQs, and proposed algorithms for returning consensus intervals in response to QCQs. We showed that corroborative ranking of intervals is more accurate than ranking snippets independently. We next hope to improve our system by replacing search APIs with our own quantity index on Web-scale corpora.

9. REFERENCES

[1] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *ICDL*, pages 85–94, 2000.

[2] K. Balog, L. Azzopardi, and M. de Rijke. A language modeling framework for expert finding. *Information Processing and Management*, 45(1):1–19, 2009.

[3] R. C. Bunescu and R. J. Mooney. A shortest path dependency kernel for relation extraction. In *EMNLP Conference*, pages 724–731. ACL, 2005.

[4] M. J. Cafarella, C. Re, D. Suciu, O. Etzioni, and M. Banko. Structured querying of web text: A technical challenge. In *CIDR*, pages 225–234, 2007.

[5] O. Chapelle, Q. Le, and A. Smola. Large margin optimization of ranking measures. In *NIPS 2007 Workshop on Machine Learning for Web Search*, 2007.

[6] T. Cheng, X. Yan, and K. C. Chang. EntityRank: Searching entities directly and holistically. In *VLDB Conference*, pages 387–398, Sept. 2007.

[7] C. L. A. Clarke, G. V. Cormack, and T. R. Lynam. Exploiting redundancy in question answering. In *SIGIR*, pages 358–365, 2001.

[8] O. de Kretser and A. Moffat. Effective document presentation with a locality-based similarity heuristic. In *SIGIR Conference*, pages 113–120, 1999.

[9] H. Fang and C. Zhai. Probabilistic models for expert finding. In *ECIR*, pages 418–430, 2007.

[10] T. Joachims. Optimizing search engines using clickthrough data. In *SIGKDD Conference*, pages 133–142. ACM, 2002.

[11] T. Joachims, H. Li, T.-Y. Liu, and C. Zhai, editors. *Learning to Rank for Information Retrieval*, Amsterdam, 2007. SIGIR Workshop.

[12] J. Ko, E. Nyberg, and L. Si. A probabilistic graphical model for joint answer ranking in question answering. In *SIGIR Conference*, pages 343–350, 2007.

[13] T.-Y. Liu. Learning to rank for information retrieval. Tutorial at SIGIR, 2008.

[14] T.-Y. Liu, T. Qin, J. Xu, W. Xiong, and H. Li. LETOR: Benchmark dataset for research on learning to rank for information retrieval. In *LR4IR Workshop*, 2007.

[15] V. Moriceau. Numerical data integration for cooperative question-answering. In *EACL Workshop on Knowledge and Reasoning for Language Processing*, pages 42–49, 2006.

[16] D. Petkova and W. B. Croft. Proximity-based document representation for named entity retrieval. In *CIKM*, pages 731–740. ACM, 2007.

[17] J. Prager, S. Luger, and J. Chu-Carroll. Type nanotheories: a framework for term comparison. In *CIKM*, pages 701–710. ACM, 2007.

[18] T. Qin, T.-Y. Liu, X.-D. Zhang, D.-S. Wang, W.-Y. Xiong, and H. Li. Learning to rank relational objects and its application to Web search. In *WWW Conference*, pages 407–416, 2008.

[19] S. Robertson. A new interpretation of average precision. In *SIGIR Conference*, pages 689–690. ACM, 2008.

[20] F. Wu and D. S. Weld. Automatically semantifying Wikipedia. In *CIKM*, pages 41–50, 2007.

[21] M. Wu and A. Marian. Corroborating answers from multiple web sources. In *WebDB: Tenth International Workshop on the Web and Databases*, 2007.

[22] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *SIGIR Conference*, pages 271–278, 2007.