Fast and Space-Efficient Entity Linking in Queries

Roi Blanco
Yahoo Labs
Barcelona, Spain
roi@yahoo-inc.com

Giuseppe Ottaviano
ISTI-CNR
Pisa, Italy
giuseppe.ottaviano@isti.cnr.it

Edgar Meij
Yahoo Labs
Barcelona, Spain
emeij@yahoo-inc.com

ABSTRACT

Entity linking deals with identifying entities from a knowledge base in a given piece of text and has become a fundamental building block for web search engines, enabling numerous downstream improvements from better document ranking to enhanced search results pages. A key problem in the context of web search queries is that this process needs to run under severe time constraints as it has to be performed *before* any actual retrieval takes place, typically within milliseconds.

In this paper we propose a probabilistic model that leverages user-generated information on the web to link queries to entities in a knowledge base. There are three key ingredients that make the algorithm fast and space-efficient. First, the linking process ignores any dependencies between the different entity candidates, which allows for a $O(k^2)$ implementation in the number of query terms. Second, we leverage hashing and compression techniques to reduce the memory footprint. Finally, to equip the algorithm with contextual knowledge without sacrificing speed, we factor the distance between distributional semantics of the query words and entities into the model.

We show that our solution significantly outperforms several state-of-the-art baselines by more than 14% while being able to process queries in sub-millisecond times—at least two orders of magnitude faster than existing systems.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing

General Terms

Experimentation, Measurement

Keywords

Entity Linking; Wikipedia; Queries; Web Search

WSDM'15, February 2–6, 2015, Shanghai, China.

ACM 978-1-4503-3317-7/15/02 ...$15.00.
http://dx.doi.org/10.1145/2684822.2685317.

1. INTRODUCTION

Commercial web search engines are presenting increasingly advanced and rich user experiences that include displays of answers, facts, entities, and other structured results. Such web search user experiences are centered around understanding and displaying information pertinent to entities present in or meant by the query, since users increasingly expect to find the actual answers and/or entities that satisfy their information need, rather than merely the documents that mention them. One critical step in this process is understanding the entities that are mentioned in queries and linking them to a reference Knowledge Base (KB). Such linking has to be performed on a very limited time and space budget as it needs to happen before any actual search process commences.

Linking free text to entities—often referred to as *entity linking*—typically comprises three steps: (i) identifying candidate mentions, i.e., which part(s) of the text to link, (ii) identifying candidate entities for each mention, and (iii) disambiguating the candidate entities based on some notion of context and coherence [14]. Recent research has made extensive use of open KBs or entity repositories such as Freebase, IMDB, Wikipedia, as well as private or proprietary ones [5]. Most linking methods from the literature, however, assume that the input text is relatively clean and grammatically correct and that it provides sufficient context [7, 11]. Queries, on the other hand, are short, noisy, and full of shorthand and other ungrammatical text, and provide very limited context for the words they contain. Hence, it is not obvious that automatic entity linking methods that have been shown to work well on news articles or web pages perform equally well on this domain. Moreover, the efficiency aspects of linking systems are not a focal point in the literature. Entity linking for web search queries poses some interesting technical challenges due to the sheer volume of the data, its dynamic nature, the creative language usage, and the required tradeoff between speed and disambiguation quality. Entity linking for queries has not received considerable attention in the literature, except for type spotting for named entity recognition [13], linking queries to semantic web concepts [19], and providing a full structured representation of the query [34].

In this paper we propose a new probabilistic model and algorithm for entity linking in web search queries that extracts a large number of candidate *aliases* for entities from click-through information and anchor text. In order to keep these large amounts of data manageable, we compress all candidates using state-of-the-art hashing and bit encoding techniques. Our algorithm is able to swiftly detect the entities in the query because we drop all the dependence assumptions

from potential candidate entities. In practice, this means that the model might not able to distinguish between situations such as "brad pitt seven" and "brad pitt olympics" if they are not present in our alias set[1] However, we are able to impose contextual knowledge by introducing a new *contextual relevance model* that uses learned representations of query words and entities, and it is able to quickly compute a relevance measure between a string of text and an entity. This way, the algorithm is able to link entities with just a forward-backward scanning procedure that can be implemented efficiently using dynamic programming in $O(k^2)$, where k is the number of query terms.

The remainder of this paper is organized as follows. Section 2 reviews some works related to ours. Section 3 introduces the probabilistic model, along with the contextual model. Section 4 presents experiments and results comparing our approaches with state of the art baselines in terms of linking quality and speed. The paper concludes in Section 5.

2. RELATED WORK

Automatically generating links to a knowledge base is one way of providing semantics to digital items. Entity linking has become an ubiquitous way to add semantics to different media types, most notably in text [22, 25] and across different genres, such as news, archives, or tweets [6, 8, 20]. The comprehensiveness, popularity, and free access to Wikipedia has made it a rich source and the most popular target knowledge base for entity linking.

Many methods for entity linking are heavily inspired by those from the field of word sense disambiguation [27]. The main challenge entity linking systems have to face is ambiguity, i.e., how to distinguish among different entities with the same label and also a possible lack of specificity, in which case more generic and possibly less meaningful entities are identified. These issues are generally addressed by identifying segments, key phrases or weighting terms based on, e.g., document-frequency inspired heuristics. For instance, a simple and frequently taken approach for linking text to concepts is to perform lexical matching between (parts of the text) and the titles [9, 21], an approach related to keyword-based interfaces to databases [36].

Milne and Witten [25] introduced one of the earliest papers on linking text to Wikipedia, which uses machine learning and unambiguous candidates for disambiguation. Among those methods, "commonness" is a popular unsupervised baseline for either short or long textual fragments [11, 20]. We use it as a baseline and describe it further in Section 4.3. In this line of work, Ratinov et al. [33] propose the use of a "local" approach (e.g., commonness) to generate a disambiguation context and then apply "global" machine learning for disambiguation. Pilz and Paaß [30] extended previous bag of word approaches for disambiguation with Latent Dirichlet Allocation generated topics, comparing topic distributions of source document with candidate entities. Recent approaches make extensive use of topic modeling, for example Houlsby and Ciaramita [16] perform inference with a variant of LDA in which each topic corresponds to a Wikipedia article, i.e., an entity. Below we include the work of Cheng and Roth [7] (which is an extension of [33]) as a representative baseline of an inference-based method. This approach tries to embed context and ensure topical coherence by linking a set of semantically-related mentions to a set of semantically-related concepts simultaneously.

The first works geared specifically towards disambiguating entities in queries focus almost exclusively on Named Entity Recognition but applied to a domain different than the usual long text found news and articles [26]. For instance Guo et al. [13] propose a language model that includes weak supervision to learn relationships, e.g., `lyrics` to `music`. Moving forward, Pantel et al. [29] predict entity type distributions in Web search queries via so-called *intents* (strings that give contextual clues about the entity type) mined from query logs. We too make use of query logs for gathering evidence, although we aim at disambiguation and not type spotting. Hu et al. [17] propose a similar approach through query intent classification. Sawant and Chakrabarti [34] assume that entities have already been annotated in text and tackle the related task of extracting query target types and words that are type hints. Pound et al. [31, 32] focus on retrieving an ordered selection of attributes, taken from a static KB, and ranking syntactic relationships from queries via a collective assignment problem.

In the case of queries—especially in the context of web search—one would need to add entity annotations as quickly as possible in order to be of any use to subsequent processes such as actual retrieval. Topic modeling approaches which typically need to perform online parameter estimation are too time-consuming. Furthermore, they are of little help when there is limited context such as in the case of queries. Little work exist on entity linking methods for queries to date. Meij et al. [19] perform entity linking in the context of the semantic web and include session history-based features to provide additional context. Proof of the increasing interest from industry and academia in query entity linking in the context of web search, however, is the recent ERD challenge[2]. In this paper we use a large (∼2.5K queries) test set from a commercial web search engine that is easily available for download.

Other relevant streams of work are related to query log mining [2], providing a signal orthogonal to that of textual corpora. For instance, Paşca [28] mined instances of semantic classes from query logs using information extraction and Alfonseca et al. [1] mined query logs to find attributes of entity instances. Once equipped with a reliable entity linking system, it is possible to provide deeper query analyses of user patterns and web usage [15].

None of the previous works deals with the problem of linking entities in the context of web search, i.e, operating in the shortest amount of time possible with additional storage constraints, nor they introduce richer *semantic* contexts as our approach does.

3. MODELING ENTITY LINKING

For our entity linking model we establish a connection between entities and their *aliases* (which are their textual representations, also known as *surface forms*) by leveraging anchor text or user queries leading to a click on the Web page that represents the entity. In the context of this paper we focus on using Wikipedia as KB and therefore only consider anchor text within Wikipedia and clicks from web search results on Wikipedia results—although it is general enough to

[1]Brad Pitt is the name of both a US Celebrity/Actor and a lesser known Australian boxer.

[2]`http://web-ngram.research.microsoft.com/ERD2014/`

also accommodate other sources of information. The problem we address consists of automatically segmenting the query and simultaneously selecting the right entity for each segment. Our Fast Entity Linker (FEL) tackles this problem by computing a probabilistic score for each segment-entity pair and then optimizing the score of the whole query. Note that we do not employ any supervision and let the model and data operate in a parameterless fashion; it is however possible to add an additional layer that makes use of human-labeled training data in order to enhance the performance of the model. We leave investigating such an additional step for future work. In the remainder of this section we detail our specific model, as well as how we incorporate context through a novel contextual relevance model and the efficiency optimizations we apply, including advanced compression techniques.

3.1 Fast Entity Linker

We begin by introducing our model, which we describe using the following random variables, assuming as an event space $S \times E$ where S is the set of all sequences and E the set of all entities known to the system:

s	is a sequence of terms $t \in s$
$\mathbf{s}$	represents a segmentation (sequence of sequences of terms) where $s \in \mathbf{s}$ is drawn from the set S
$\mathbf{e}$	represents a set of entities $e \in \mathbf{e}$, where each e is drawn from the set E
a_s	indicates if s is an alias
$a_{s,e}$	indicates if s is an alias pointing (linking/clicked) to e
c	indicates which collection acts as a source of information query log or Wikipedia (c_q or c_w)
$n(s,c)$	is the count of s in c
$n(e,c)$	is the count of e in c

Let q be the input query, which we represent with the set S_q of all possible segmentations of its tokens $t_1 \cdots t_k$. The algorithm will return the set of entities $\mathbf{e}$, along with their scores, that maximizes

$$\operatorname*{argmax}_{\mathbf{e} \in E} \log P(\mathbf{e}|q) = \operatorname*{argmax}_{\mathbf{e} \in E, \mathbf{s} \in S_q} \sum_{e \in \mathbf{e}, s \in \mathbf{s}} \log P(e|s) \ . \quad (1)$$

In Eq. 1 we assume independence of the entities given a query segment. Each individual entity/segment probability is then estimated as:

$$\begin{aligned} P(e|s) &= \sum_{c \in \{c_q, c_w\}} P(c|s) P(e|c,s) \\ &= \sum_{c \in \{c_q, c_w\}} P(c|s) \sum_{a_s = \{0,1\}} P(a_s|c,s) P(e|a_s,c,s) \\ &= \sum_{c \in \{c_q, c_w\}} P(c|s) \Big[P(a_s = 0|c,s) P(e|a_s = 0,c,s) \\ &\quad + P(a_s = 1|c,s) P(e|a_s = 1,c,s)) \Big]. \end{aligned} \quad (2)$$

The maximum likelihood probabilities are the following (note that in this case $P(e|a_s = 0, c, s) = 0$ and therefore the right hand side of the summation cancels out):

$$P(c|s) = \frac{n(s,c)}{\sum_{c'} n(s,c')} \quad (3)$$

$$P(a_s = 1|c,s) = \frac{\sum_{s:a_s=1} n(s,c)}{n(s,c)} \quad (4)$$

$$P(e|a_s = 1,c,s) = \frac{\sum_{s:a_{s,e}=1} n(s,c)}{\sum_{s:a_s=1} n(s,c)} \ . \quad (5)$$

Those maximum likelihood probabilities can be smoothed appropriately using an entity prior. Using add-one and Dirichlet priors smoothing the probabilities results in:

$$P(e|c) = \frac{n(e,c) + 1}{|E| + \sum_{e \in E} n(e,c)} \quad (6)$$

$$P(e|a_s,c,s) = \frac{\sum_{s:a_{s,e}=1} n(s,c) + \mu_c \cdot p(e|c)}{\mu_c + \sum_{s:a_s=1} n(s,c)} \ . \quad (7)$$

In this case $P(e|c) = P(e|a_s = 0, c, s)$, and $P(a_s = 0|c,s) = 1 - P(a_s = 1|c,s)$. Similarly, we smooth $P(c|s)$ using Laplace smoothing (add-one smoothing).

Note that some terms might not be covered by any segment s and therefore not link to any entity. To handle these segments, we define a special entity `not_linked` and set $P(\texttt{not\_linked}|s) = \ell$ where ℓ is a global hyperparameter that can tuned to set the propensity of the algorithm to link segments.

An alternative to Eq. 1 would be to select the segmentation that optimizes the score of the top ranked entity:

$$\operatorname*{argmax}_{\mathbf{e} \in E, \mathbf{s} \in S_q} \max_{e \in \mathbf{e}, s \in \mathbf{s}} P(e|s). \quad (8)$$

Both Eq. 1 and Eq. 8 are instances of the same general segmentation problem, defined as follows. Given a sequence of terms $\mathbf{t} = t_1 \cdots t_k$, denote any segment of the sequence with

$$[t_i t_{i+1} \ldots t_{i+j-1}], \forall i, j \geq 0. \quad (9)$$

Let $\gamma(s)$ be any scoring function that maps segments to real numbers, then the maximum score of a segmentation is defined as follows:

$$\begin{aligned} &m(t_1, t_2, \ldots, t_k) = \\ &\max \Big(\phi\left(m(t_1), m(t_2, \ldots, t_k)\right), \phi\left(\gamma([t_1 t_2]), m(t_3 \ldots t_k)\right) \\ &\qquad , \ldots, \phi\left(\gamma([t_1 \ldots, t_{k-1}]), m(t_k)\right), \gamma([t_1 \ldots t_k]) \Big), \end{aligned} \quad (10)$$

where $m(t_1) = \gamma([t_1])$ and $\phi(a,b)$ is an associative aggregation function, such as $\phi(a,b) = a + b$ in the case of Eq. 1 and $\phi(a,b) = \max(a,b)$ in the case of Eq. 8. Since the scoring function $s(\cdot)$ only depends on the given segment and not on the others, the segmentation with maximum score can be computed in $O(k^2)$ time using dynamic programming.

We instantiate the problem above would be the following scoring function

$$\gamma(s) = \textsc{highestscore}(s,q) = \max_{e \in E} \log P(e|s,q), \quad (11)$$

that, given a segment s of the query and a *context* q (the query itself), returns the highest score among the entities associated with the alias s; when the segment does not match any alias, and hence it has no associated entities, it returns a value that is an identity for the aggregation function $\phi(\cdot,\cdot)$; it also returns an identity if the segment is empty. In the basic FEL model the context is ignored, so $P(e|s,q)$ is just

Algorithm 1 Entity-linking algorithm

Require: A user query q, a function HIGHESTSCORE$(\cdot)$, and an aggregation function $\phi(\cdot,\cdot)$.

```
p ← TOKENIZE(q)
l ← LENGTH(p)
maxscore[] ← new array[l + 1]
previous[] ← new array[l + 1]
for i = 0 to l do
  for j = 0 to i do
    score ← φ(maxscore[j], HIGHESTSCORE(p[j : i], q))
    if score > maxscore[i] then
      maxscore[i] ← score
      previous[i] ← j
    end if
  end for
end for
return maxscore[l]
```

$P(e|s)$. Algorithm 1 details our full algorithm. Note that it makes $O(k^2)$ calls to the HIGHESTSCORE function, which is acceptable since the number of terms in a query is usually very small.

3.2 Modeling Context

In our algorithm above, the probability of an entity given an alias is computed independently of the other aliases that match the query or the surrounding words. As we pointed out, this is a design choice to enable a fast linking algorithm. However, the context of the alias can help to disambiguate the candidate entities. For example, a user submitting the query "Hollywood lyrics" is probably interested in the song rather than the geographical location; if the only matching alias is `Hollywood`, the algorithm will not be able to disambiguate between candidates.

To exploit the contextual information in the remainder of the query we introduce a *contextual relevance model*, that is, we estimate the probability that the entity e is relevant to the context $\mathbf{t}$, in this case the whole query. This probability can be incorporated into our model by factoring it into the HIGHESTSCORE function. To this aim, we need to compute $P(e|s,q)$ where s is the string segment and q is the context $t_1 \cdots t_k$. Then $P(e|s,q) = \frac{P(e)P(s,q|e)}{P(s,q)}$ and, assuming independence between q and s and conditional independence given e, we get $P(e|s,q) = \frac{P(e)}{P(q)P(s)} \cdot P(q|e)P(s|e)$, which is equal to $P(e|s)\frac{P(q|e)}{P(q)}$. The first factor is estimated as in Eq. 2. To estimate the second factor efficiently we again assume conditional independence on e and write it as $\prod_i \frac{P(t_i|e)}{P(q)}$. We are thus interested in estimating $P(t|e)$, i.e. the probability that the term t is relevant to the entity e.

We estimate $P(t|e)$ using a multiset $\mathcal{R}_e$ of words t that are known to be relevant to e; in our experiments, we use the words in the first section of the Wikipedia article on e. A simple approach would be to just count the number of occurrences of t, $P(t|e) = \frac{|\{t \in \mathcal{R}_e\}|}{|\mathcal{R}_e|}$, which would make the model a multi-class Naive Bayes classifier. This approach has two problems, however. First, the number of parameters (that is, the collection of multisets $\mathcal{R}_e$), is essentially as large as the collection of texts. Second, it does not take into account the semantic similarity of words, namely that if a word t_1 is relevant to an entity e and t_2 is close in meaning to t_1, it is likely that t_2 is relevant to e as well. We therefore use *continuous representations of words*, specifically the `word2vec` embeddings [24]. These embeddings map words to vectors of real numbers so that words that are close in meaning are mapped to vectors close in cosine distance. The vectors are computed in a completely unsupervised fashion text by exploiting the distributional semantics hypothesis. That is, words that co-occur often with the same words are close in meaning. We denote as $v_t \in \mathbb{R}^D$ the vector that represents the word t. We use these word vectors to model $P(t|e)$ as a binary logistic regression classifier, that is, we map each entity e to a vector $v_e \in \mathbb{R}^{D+1}$ and define $P(t|e) = \sigma([v_t\ 1] \cdot v_e)$, where $\sigma(x) = \frac{1}{1+e^{-x}}$. Since we need to compute the product of the probabilities $P(t_i|e)$ in order to score each entity e with a query, the overall complexity is $O(kD)$ operations. Below we refer to this method as *LR*.

Each classifier v_e is trained with L_2-regularized logistic regression to distinguish the multiset $\mathcal{R}_e$ (the positive examples) from the unigram distribution from the whole collection (the negative examples). Since this would require training on a set of examples as large as the dictionary, we use a method known as *negative sampling* [23]: we sample ρ words from the unigram distribution as negative examples, so that the overall number of examples is bounded by $O(|\mathcal{R}_e|)$. The vector v_e, then, is computed as the maximum of the function

$$\sum_{t \in \mathcal{R}_e} \log \sigma([v_t\ 1] \cdot v_e) + \sum_{t \in \mathcal{N}_e} \log \sigma(-[v_t\ 1] \cdot v_e) - \lambda \|v_e\|_2^2, \quad (12)$$

where $\mathcal{N}_e$ is the multiset of $\rho|\mathcal{R}_e|$ negative samples and λ is the regularization parameter. In Section 4.2 we detail how to tune ρ and λ in an unsupervised manner.

Note that the whole model consists of one vector of D numbers for each word in the dictionary and one vector of $D+1$ numbers for each entity in the knowledge base. Even using a 32-bit floating point number representation, the space occupancy would be $4(E(D+1) + WD)$, where E is the number of entities and W the number of words, regardless of the size of the sets $\mathcal{R}_e$. In Section 3.3.2 we describe a method to reduce this even further.

An alternative approach to model the relevance of e for a query is to define the entity vector v_e as the centroid of the vectors representing the words in $\mathcal{R}_e$, that is

$$v_e = \frac{1}{|\mathcal{R}_e|} \sum_{t \in \mathcal{R}_e} v_t, \quad (13)$$

and similarly the query vector v_q as the centroid of the vectors of the words of the query, that is $v_q = \frac{1}{k}\sum_i v_{t_i}$, and defining the relevance as the cosine $\cos(v_q, v_e)$. Aggregating the vectors of bags of words by taking their centroid is a widely used approach with continuous word representations, due to an increased efficiency [24].

The main advantage is higher efficiency when scoring a query against a set of entities; since the vector v_q is computed only once, scoring an entity requires only $O(D)$ operations, as opposed to $O(kD)$ for LR. Since the number of query terms is always small in practice, the difference is almost negligible. On the other hand, the cosine distance cannot be interpreted as a probability and it provides no measure of confidence. A simple way to verify this is to visualize the multisets $\mathcal{R}_e$ as clusters in the high-dimensional space: their centroids are a good indication of the location of the clusters, but they contain no information about their spread. For example, if a query vector is equidistant from a very broad cluster and

a very tight cluster, the broad cluster should have higher confidence, but the centroid method cannot distinguish the two. In fact, as we will see below, Centroid is outperformed by LR in both a benchmark task and in the actual entity linking task.

3.3 Optimizations

In this section we detail the three types of optimizations we employ, including early stopping and two kinds of compression.

3.3.1 Early Stopping

With the added contextual scoring HIGHESTSCORE($p[i:j], q$) must now return, among all the entities that match the alias $p[i:j]$, the one that maximizes $P(e|s)\frac{P(q|e)}{P(q)}$, so the probability must be computed for each entity. First, note that we can remove the denominator $P(q)$ from the computation, since it does not depend on the entity. Then, the score to compute becomes $P(e|s)P(q|e)$. The first component is the basic FEL score, the second is the contextual relevance. The contextual relevance might take longer to compute than the FEL score, because it involves retrieving the entity vectors from the model data structure and computing several vector-vector products. However, we are only interested in retrieving the highest-scored entity. We can significantly reduce the number of score computations by early-stopping the process in a safe way. We do this by noting that $P(q|e)$ is at most 1; hence, if e_* is the top-scoring entity and e a candidate entity, if $P(e|s) < P(e_*|s)P(q|e_*)$ then a fortiori the full score of e cannot be higher than that of e_*. With this in mind, we can process the entities sorted by decreasing score $P(e|s)$ and stop computing the contextual relevance score as soon as $P(e|s)$ is smaller than the full score of the current top-scoring entity. This technique reduces the overall runtime of the algorithm by a factor of 5.

3.3.2 Compressing the Vectors

We now turn to the data structure used to store the word and entity vectors. The data structure represents a general mapping from strings to vectors, which we can split in two parts: a mapping from n strings to numeric identifiers in $[0, n)$, and the actual vectors, which is convenient to see as a matrix $V \in \mathbb{R}^{n \times D}$, whose rows are the vectors. The mapping can be easily represented with a minimal perfect hash function, which computes the identifier of a string in constant time and guarantees that no collisions can occur between strings of the key set. Such a function can however return arbitrary values on other strings; to make the collision probability negligible a constant-sized *signature* is associated to each string, so that it can be checked whether the string being looked up was present in the key set. This data structure is commonly referred to as a signed minimal perfect hash function for which we use the Sux4J implementation.[3]

To store the matrix V we adopt standard techniques from vector quantization and signal compression (for reference see [12]) and quantize the entries of the matrix with an uniform dead-zone quantizer, that is, an element x is quantized as $\text{sgn}(x)\lfloor |x|/q \rfloor$ for a given quantization parameter q. We use the same q for all the elements in the matrix, and choose the largest value that yields a target error bound; specifically, we target a relative error in L_2 norm of the vectors of 0.1, which produced no measurable loss in accuracy of the vectors in our experiments. The integers obtained from quantization are then encoded with Golomb codes; since the columns of the matrix might have different statistics, we use a different Golomb modulus for each column. We concatenate the encodings of each vector into a single bit stream, and store their starting positions in an Elias-Fano monotone sequence data structure that allows to retrieve them in constant time.

To further improve the compression of the word vectors we note that transforming them with any operation that preserves the mutual cosine distances does not affect their quality. We can therefore apply the orthogonal Karhunen–Loève transform [12] to the vectors before compressing them, without having to apply the inverse transform at decoding time. The transform improves the compression by about 10% without affecting the overall accuracy. It is however not possible to apply the same technique to the entity vectors, since it would not preserve the scalar products against the word vectors unless the inverse transform is applied in decoding, which would be prohibitively slow.

Overall, the compressed vector representations take 3.44 bits per entry for the word vectors, 3.42 for the Centroid vectors, and 3.83 for the LR, which is almost 10 times smaller than using 32-bit floating points numbers.

[3] `http://sux.dsi.unimi.it/`

	Number	Size
Alias strings	114M	7.24 bytes/alias
Entity strings	4.6M	25.22 bytes/alias
Entity values (rows 5,6)	9.2M	3.72 bits/value
Alias values (rows 1-4, 7, 8)	912M	5.32 bits/value

Table 2: Size of compressed features and strings.

3.3.3 Compressing the Features

We also generate a compressed data structure to hold the information about aliases and entities. The numerical features required by the model are summarized in Table 1. The data structure is a hash table represented as follows. Each key of the table corresponds to a different alias and the values are split into two parts: entity-independent features (rows 1-4) stored as a monotone sequence of integers, and a sequence of N entity-dependent features (rows 5-8), one per candidate entity. For compactness, entities in the table are represented with a numerical id although we hold a separate identifier to string map stored as a front-coded list. We store integer values using Elias-Fano monotone sequences [10]. However, and given that the number of entities is several orders of magnitude smaller than the number of aliases, we store the alias-independent features (rows 5,6) in its own Elias-Fano list, indexed by entity id. The alias strings are perfectly-hashed in a similar fashion to the word vectors and we hold an additional (compressed) list of cut pointers indicating the boundaries of the per alias information in the compressed list of values. The size of the different components of the hash table is detailed in Table 2.

Note that once the scoring function is fixed it would be sufficient to store just the mapping from aliases to (entity, score) pairs; however, this would require to recompute the data structure if the score function needs to be modified. In this case we favor flexibility over space, since the savings would be very small and the time to compute the scores is negligible.

	Features included for every alias
1	Number of times the alias was submitted as a query $n(s, c_q)$
2	Number of times the alias resulted in a click $\sum_{s:a_s=1} n(s, c_q)$
3	Number of times the alias was found in Wikipedia's text $n(s, c_w)$
4	Number of times the alias was present inside anchor text $\sum_{s:a_s=1} n(s, c_w)$
	Features included for every entity
5	Number of times the entity's Wikipedia page was clicked after a query was submitted $n(e, c_q)$
6	Number of times the entity's Wikipedia page was linked $n(e, c_q)$
	Alias and entity features
7	Number of times the alias resulted in a click in the entity's Wikipedia page e, $\sum_{s:a_{s,e}=1} n(s, c_q)$.
8	Number of times the alias occurred in an anchor pointing to the entity's Wikipedia page e, $\sum_{s:a_{s,e}=1} n(s, c_w)$

Table 1: Features in the data structure.

4. RESULTS AND DISCUSSION

This section describes the experiments we performed to assess effectiveness and efficiency of our entity linking model for web search queries. Prior to that, we describe the queries and data used for evaluation and detail the baselines we compare our methods against. Additionally, we describe a procedure for automatically tuning the hyperparameters of the context model and a comparison between the performance of Centroid and LR vectors, independent of the linking task.

4.1 Data

All the experiments use Wikipedia as a knowledge base and Yahoo's Webscope[4] search query log to entities, which is a publicly available editorial set that contains annotations for 2583 queries distributed across 980 user sessions. The dataset contains manually identified links to entities in the form of Wikipedia articles and provides the means to train, test, and benchmark such systems using manually created, gold standard data. In order to build our data structures, we use Yahoo query logs spanning 18 months, and all of Wikipedia's anchor text extracted from a Wikipedia dump dated May 2014. The contextual vectors described in Section 3.2 are built using the same Wikipedia version. Note that for our current experiments all our data is using Wikipedia as a knowledge base, although it would easily be possible to extract information from user actions on other domains. The only requirement is that the entities' web pages are consolidated. One could, e.g., collect information about *actors* and *movies* by looking at clicks landing on `imdb.com` web pages. The final structure contains 4.6M entities, which comprises the set of those Wikipedia entries that correspond to articles proper, i.e., category, help, and discussion pages are discarded. After some basic normalization (punctuation stripping and case folding) the number of aliases mined from the anchor text and query logs totals 113M. It is interesting to examine the distribution of the number of entities that are potential targets for a given alias. Figure 1 displays the frequencies of the number of entities per alias for the whole entity-alias set. For instance, there are about 10^8 aliases that only point to one entity. The figure indicates a heavy-tailed distribution. However, this distribution is different if we take a look at how the number of candidate entities per alias differs across queries. Over a sample of 1M random queries the average number of entities per alias is 93.85 with a standard deviation of 335.284, which implies there is a large set of entities to disambiguate and the size of this set can vary greatly from query to query. If we break down this average and deviation by alias length (Figure 2), we observe that it drops dramatically when the length of the alias query segment referring to the entity is greater than 1. In practice this means that unigram aliases are more difficult to disambiguate even though they have a considerably higher frequency and that the linker has to somehow balance the tradeoff between longer aliases with a more precise meaning and more frequent smaller but ambiguous words. Conversely, Figure 3 plots the different number of aliases that are associated with entities, which is another heavy-tailed distribution. The majority of the aliases (86%) point to a single entity.

[4] `http://webscope.sandbox.yahoo.com/`

k	Centroid	LR
5	7.857	7.386
10	5.711	5.385
15	4.575	4.363
20	3.931	3.883

Table 3: Accuracy for the retrieval task.

The features compressed in the data structure are described in Table 1. All the features are integer values and stored in a contiguous block of memory as described in the next section. Per-entity features are also kept in contiguous blocks of memory and referenced appropriately from aliases features using the entity id, in order to avoid wasting storage unnecessarily. It is finally worth to remark that our models are almost parameter free—there is a smoothing parameter μ per input source (query logs, anchor text) and a regularization parameter for the word vectors. We self-tune the regularization parameters automatically using the method described in Section 4.2. The μ parameters have limited impact on the performance and were set to a fixed value of 10. For ranking entities we employ Eq. 1 in our linking algorithm.

4.2 Training Entity Vectors

We train the word vectors with the original `word2vec` code on a text collection of 3 billion words extracted from Wikipedia, using the standard vector dimensionality $D = 200$. To train the LR entity vectors we use the SciPy implementation of L-BFGS on the loss function Eq. 12, computing the

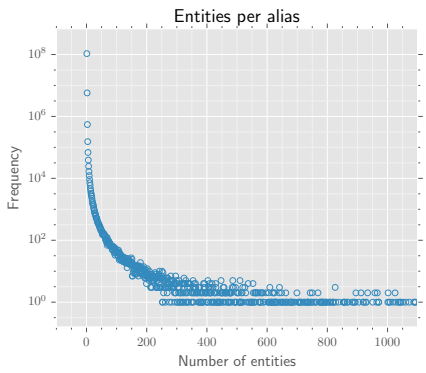

Figure 1: Distribution of the number of entities per alias (logscale).

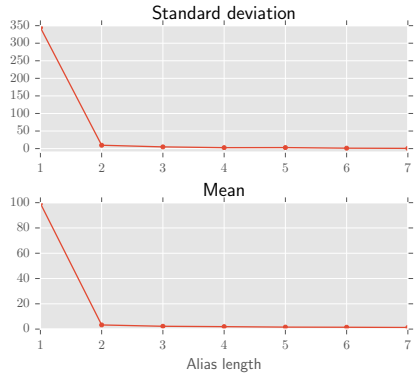

Figure 2: Standard deviation and mean of the number of entities per alias for different alias lengths.

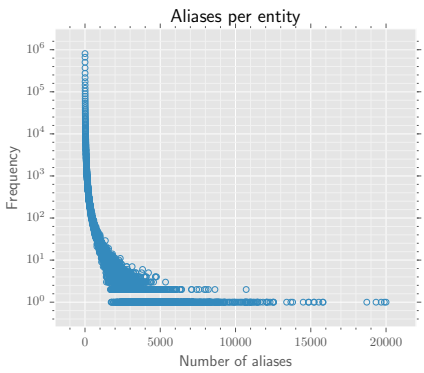

Figure 3: Distribution of the number of aliases per entity (logscale).

function and its gradients using Theano [3]. Note that the task is trivially parallelizable, as every entity vector can be computed independently. Computing vectors for all the 4.6 million entities in our knowledge base with the chosen hyperparameters takes about 12 hours on a machine with 32 cores. Given the scarcity and sparsity of labeled query examples we seek an unsupervised way to tune the hyperparameters λ and ρ (the regularization parameter and the number of negative samples). To this aim we define an artificial task which we call the *retrieval task*, and optimize the parameters on it. We define the retrieval task as follows. We sample a set of entities $\mathcal{E}_{\text{train}}$ among those whose multiset $\mathcal{R}_e$ has at least 50 words, and extract a subsample $\mathcal{E}_{\text{test}} \subset \mathcal{E}_{\text{train}}$. For each entity e in $\mathcal{E}_{\text{train}}$ we hold out k words from $\mathcal{R}_e$ and train the entity vector on the remaining words. Then, for each entity e in $\mathcal{E}_{\text{test}}$ we use the k held out words to score all the entities in $\mathcal{E}_{\text{train}}$ and compute the rank of e in the induced ranking. Then, inspired by the standard discount gains in DCG metrics, we define the accuracy as the average logarithm of the ranks.

We observe that as the number of negative samples ρ increases the accuracy improves but the training time grows linearly; we find a satisfactory trade-off at $\rho = 20$, where the accuracy reaches a plateau. With respect to the regularization parameter, instead, we find a maximum at $\lambda = 10$. In Table 3 we report the accuracy numbers for Centroid and LR trained with the above hyperparameters, for different values of k, using 50K entities for $\mathcal{E}_{\text{train}}$ and $5,000$ entities for $\mathcal{E}_{\text{test}}$. Note that LR outperforms Centroid in all cases.

4.3 Entity Linking

We now compare the performance of our FEL models against several baselines: state-of-the-art systems (Wikifier and TAGME), a retrieval approach based on language models, and *commonness*. This latter method is a popular unsupervised baseline and has been shown to perform quite well on various kinds of input texts (tweets, news items, etc.) [11, 20]. It is defined as

$$cmns(e, s) = \frac{|L_{e,s}|}{\sum_{e'} |L_{e',s}|}, \tag{14}$$

where e is an entity, s a sequence of terms and $L_{e,s}$ the set of all links within Wikipedia with anchor text s that target the page that represents e. In order to process the queries using this method, we split each query into every single term n-gram starting from the longest possible, i.e., the whole query. Then we try to match each n-gram with anchor texts as found in Wikipedia. In the case an anchor text is found we score the entities based on *cmns* and ignore the smaller, constituent n-grams. Otherwise we recurse and try to match the (n-1)-grams. Information retrieval-based approaches (denoted LM) make use of a Wikipedia index and can rank the pages using their content. We indexed each Wikipedia article using different fields (title, content, anchors, first paragraph, redirects, and categories) and implemented the method by Kim and Croft [18] for ranking the different pages. This method computes a per-field weighting term based on the query. In effect, it determines the likelihood that a term is generated by a certain document field, based on all the documents in the collection. It then uses this likelihood to create a per-field-weighted query using Dirichlet priors smoothing. We note the performance of this method on its own is quite poor. We therefore enhanced it using a query-independent feature and rerank the obtained search results by the amount of clicks on a Wikipedia page as document prior (LM-Click). The method denoted *Bing* is a reference baseline that issues the query through the Bing search API and returns the first Wikipedia page as a result. This is a strong baseline in that commercial Web search engines use supervised, machine learned ranking models incorporating a large number of features, including features derived from the text of entity pages, the hyperlink graph and user interactions (click logs).

Ferragina and Scaiella [11] propose a method called TAGME with an explicit focus on short texts. They use a voting scheme in conjunction with heavy pruning of n-grams unrelated to the input text. This results in a fast method that is especially suitable for short text. As such we include the results of querying the TAGME REST API as a baseline.

Ratinov et al. [33] propose the use of a "local" approach (e.g., commonness) to generate a disambiguation context and then apply "global" machine learning for disambiguation. Wikifier[5] is an implementation of this state-of-the-art system which combines Wikipedia pages, gazetteers, and Wordnet. We denote our Fast Entity Linker as FEL, and the

[5] `http://cogcomp.cs.illinois.edu/page/demo_view/Wikifier`

	P@1	MRR	MAP	R-Prec
LM	0.0394	0.1386	0.1053	0.0365
LM-Click	0.4882	0.5799	0.4264	0.3835
Bing	0.6349	0.7018	0.5388	0.5223
Wikifier	0.2983	0.3201	0.2030	0.2086
TAGME	0.6682	0.7043	0.5458	0.5462
Commonness	0.7336	0.7798	0.6418	0.6464
FEL	0.7669	0.8092	0.6528	0.6575
FEL+Centroid	0.8035	0.8366	0.6728	0.6765
FEL+LR	**0.8352**	**0.8684**	**0.6912**	**0.6883**

Table 4: Performance of the different linking methods on the Webscope dataset . FEL outperforms significantly all the baselines (t-test, p<0.01, P@1) except for commonness. FEL+Centroid outperforms significantly all the baselines and FEL, and FEL+LR is significantly better than the rest.

two methods that use contextual vectors are FEL+Centroid and FEL+LR to indicate the different ways the vectors are learned.

4.3.1 Results

We evaluate the different systems using early precision metrics, i.e, Precision at rank 1 (P@1), Mean Reciprocal Rank (MRR), R-Precision (R-Prec) and also Mean Average Precision (MAP). We remark that our methods return the ranked list of entities that maximize the likelihood of a query segmentation; because this segmentation potentially contains several entities, we return one entity per identified query chunk (as a ranked list). In this respect, recall oriented metrics would promote methods that return multiple interpretations for the same query span (this would be partially the case for MAP). We report on statistical significance using a two-sided t-test with $p < 0.01$.

Table 4 shows the main entity linking results. With respect to the baselines, Wikifier was designed primarily for entity linking on longer texts and underperforms heavily with respect to the rest of the methods. TAGME is specifically designed for short texts and performs better, although both commonness and FEL outperform it by a large margin. Retrieval-based approaches are not competitive on their own. Retrieving entities using the Bing search API, which presumably uses a large amount of available signals for ranking, is the first competitive baseline, along with commonness. Somehow surprisingly, in our dataset this baseline outperformed every other baseline by a large margin. FEL clearly and significantly outperforms all the baselines, except for commonness. FEL+Centroid, however, significantly outperforms all the baselines as well as FEL, and FEL+LR is in turn significantly better than all other methods.

4.4 Runtime Performance

We now turn into comparing the execution times of different systems. All the experiments were conducted on a standard MacBook Pro, with 2.7 GHz Intel Core i7 cpu and 16GB of RAM. Timings were measured on a random sample of 1M queries and each experiment was repeated 5 times; below we report the average numbers. A natural candidate for a fast baseline would be an index-based structure that can perform quick look-ups of aliases with entity candidate information. In the open source domain, arguably the fastest inverted file-based system would be MG4J,[6] which in its most recent version has an index built around the idea of quasi-succinct monotone sequences [35]. We rank entities using a standard BM25 scorer over the title and body fields. We also compare with Wikifier, which uses Lucene indexes and several gazetteers as its basic data-structures. Wikipedia MG4J's indexes occupy approximately 2GB of compressed data (over 6GB including positional information), whereas Wikifier's gazetteers and indexes add up to over 6GB.

System	Average time	Data size
Wikifier	44.32 ms	6.8G
Retrieval (And)	13.63 ms	2.5G
Retrieval (Or)	210.13 ms	2.5G
FEL	0.14 ms	1.8G
FEL+Centroid	0.27 ms	2.3G
FEL+LR	0.40 ms	2.3G

Table 5: Entity linking efficiency. For FEL and FEL+{Centroid,LR} we include the size of all the compressed data structures. For *Retrieval* we include the size of the indexes without positional information, and for Wikifier the size of the indexes and gazetteers.

The results for the efficiency experiments are shown in Table 5. FEL outperforms the Wikifier approach 300-fold. When we compare FEL and FEL+Context, we observe that factoring the contextual vectors into the ranking adds an overhead of 0.13 ms in the case of Centroid and 0.26 ms in the case of LR. MG4J in AND mode—while being remarkably fast for a stand-alone index-based system—has a very low recall and it is not a viable solution for entity linking. In contrast, MG4J in OR mode is 4 orders of magnitude slower than FEL. We highlight that the vectors decompression procedure is responsible for the biggest part of the runtime overhead. The dynamic pruning procedure described in Section 3.3.1 alleviates this problem by a factor of 5, yielding average query times below half a millisecond. Without early termination, the contextual algorithm has an overhead of about 2 ms over FEL.

4.5 Error Analysis

We now illustrate some of the caveats and advantages of the proposed methods. Our FEL model makes mistakes of different kinds. A fraction of these errors come from typos in the queries that are not accounted for in the aliases that we use. When we incorporate a proprietary query spelling corrector before linking, the performance of our methods is increased by approximately 3% (MAP increases to 0.7000, P@1 to 0.8610). Examples of other kinds of errors are illustrated in Table 6. Some of them have to do with overly popular candidates, or whenever there are two concepts that are too similar (`Crossbreed` versus `Dog_Type`) and the provided context is not enough to discriminate among them. On the other hand, too much context might promote entities that are too specific, such as in the case of the Sheraton Hotel in Kansas. Table 7 displays some queries that were improved from the context-agnostic FEL when using FEL+LR. In those cases, the semantic descriptions taken from the abstracts of the entity pages are a signal strong enough to

[6] `http://mg4j.di.unimi.it`

Query	Answer	Correct Answer	Type of Error
designer dogs types	Dog_type	Crossbreed	Parallel concept
clinton falls asleep	Clinton_Falls_Township	Bill_Clinton	Wrong segmentation
candidate killed in storm	Storm_(Marvel_Comics)	Storm	Candidate too popular
france world cup 98 reaction	FIFA_World_Cup	FIFA_World_Cup_1998	Not enough context
2005 presidential election in Egypt	Elections_in_Egypt	2005_Elections_in_Egypt	Not enough context
kansas city hotels airport	Sheraton_Kansas_City_Hotel	Kansas_City	Too much context

Table 6: Different examples of mistakes made by the context aware system along with their types, top linked entity displayed.

Query	FEL answer	FEL+Context
install roof insulation	Insolation	Building_insulation
inventor of gunpowder	Gunpowder	History_of_gunpowder
us political map	Map	Red_states_and_blue_states
dj jobs	Jobs_(film)	Disc_jockey
buy used car parts online	Automobile	Used_car
what is the longest running tv show	Television	The_Simpsons

Table 7: Example queries that were *corrected* from the base system FEL when the contextual vectors were incorporated into the model, top linked entity displayed.

boost entity candidates that are more related to the query, e.g., History_of_gunpowder versus Gunpowder.

5. CONCLUSIONS

In this paper we have described a probabilistic model for entity linking on web search queries. This task already poses numerous challenges by itself but a major limitation in the context of web search is that we are heavily constrained from a runtime perspective, imposing time and space requirements whilst maintaining accuracy. Our method leverages information from query logs and anchor texts to automatically obtain a large number of aliases for the entities in a knowledge base and uses a probabilistic model to rank entities in query segments. Our model uses dynamic programming to generate the best entity segmentation and, in order to add information about the query context into the ranking, we devise a novel way of aggregating vectors that encode word distributional semantics. This new contextual vector model significantly outperforms various state-of-the-art baselines from the literature on a public test set comprised of 2.5K queries, while being able to provide sub-millisecond response times—several orders of magnitude faster than entity linking systems from the literature.

Furthermore, all the data structures used by our system are compressed using state-of-the-art hashing, quantization, and integer encoding techniques. This yields a final data-pack storing hundreds of millions of aliases for all 4.6M Wikipedia entities using less than 2GB of storage. We also note that the size of the word vector representations can be reduced by an order of magnitude without any execution overhead.

For future work, we would like to test the scalability of the system when considering hundreds of millions of entities and billions of aliases. This might require to employ static pruning algorithms to reduce the size of the compressed data structures [4]. Additionally, we generated the contextual vectors using Wikipedia but we could expand the input sources to accommodate for other types of data, such as queries or news, which could incorporate more interesting contextual relations that fall outside of encyclopedic knowledge (e.g., trends, gossip, etc.). Another interesting line of work could involve incorporating session-level features into the entity linking process in order to make use of a richer source of contextual information. Finally, we believe that our contextual entity models are general enough to be of value in different types of applications that make use of word aggregation distributional similarities.

Acknowledgments

The authors would like to thank Sebastiano Vigna and Hugues Bouchard for their kind help in implementing the system described in the paper, and their support with uncountable useful suggestions. The second author was supported by Midas EU Project (318786), eCloud EU Project (325091) and by the Yahoo Faculty Research and Engagement Program.

References

[1] E. Alfonseca, M. Pasca, and E. Robledo-Arnuncio. Acquisition of instance attributes via labeled and related instances. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2010.

[2] R. Baeza-Yates, C. Hurtado, and M. Mendoza. Query recommendation using query logs in search engines. In *Proceedings of the 2004 International Conference on Current Trends in Database Technology*, 2004.

[3] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[4] R. Blanco and A. Barreiro. Probabilistic static pruning of inverted files. *ACM Trans. Inf. Syst.*, 28(1):1:1–1:33, Jan. 2010.

[5] R. Blanco, B. B. Cambazoglu, P. Mika, and N. Torzec. Entity recommendations in web search. In *The*

Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, 2013.

[6] M. Bron, B. Huurnink, and M. de Rijke. Linking archives using document enrichment and term selection. In *Proceedings of the 15th International Conference on Theory and Practice of Digital Libraries: Research and Advanced Technology for Digital Libraries*, 2011.

[7] X. Cheng and D. Roth. Relational inference for wikification. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013*, 2013.

[8] S. Cucerzan. Large-scale named entity disambiguation based on wikipedia data. In *Proceedings of EMNLP-CoNLL 2007*, 2007.

[9] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J. A. Tomlin, and J. Y. Zien. Semtag and seeker: Bootstrapping the semantic web via automated semantic annotation. In *Proceedings of the 12th International Conference on World Wide Web*, 2003.

[10] P. Elias. Efficient storage and retrieval by content and address of static files. *J. ACM*, 21(2):246–260, 1974.

[11] P. Ferragina and U. Scaiella. Tagme: On-the-fly annotation of short text fragments (by wikipedia entities). In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, 2010.

[12] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Norwell, MA, USA, 1991. ISBN 0-7923-9181-0.

[13] J. Guo, G. Xu, X. Cheng, and H. Li. Named entity recognition in query. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2009.

[14] B. Hachey, W. Radford, J. Nothman, M. Honnibal, and J. R. Curran. Evaluating entity linking with Wikipedia. *Artificial Intelligence*, 194(0):130 – 150, 2013.

[15] L. Hollink, P. Mika, and R. Blanco. Web usage mining with semantic analysis. In *Proceedings of the 22nd International Conference on World Wide Web*, 2013.

[16] N. Houlsby and M. Ciaramita. A scalable gibbs sampler for probabilistic entity linking. In *Advances in Information Retrieval*. Springer, 2014.

[17] J. Hu, G. Wang, F. Lochovsky, J.-t. Sun, and Z. Chen. Understanding user's query intent with wikipedia. In *Proceedings of the 18th International Conference on World Wide Web*, 2009.

[18] J. Kim and W. B. Croft. A field relevance model for structured document retrieval. In *Advances in Information Retrieval - 34th European Conference on IR Research, ECIR 2012*, 2012.

[19] E. Meij, M. Bron, L. Hollink, B. Huurnink, and M. de Rijke. Mapping queries to the Linking Open Data cloud: A case study using DBpedia. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(4):418 – 433, 2011.

[20] E. Meij, W. Weerkamp, and M. de Rijke. Adding semantics to microblog posts. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, 2012.

[21] P. N. Mendes, A. Passant, P. Kapanipathi, and A. P. Sheth. Linked open social signals. In *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, 2010.

[22] R. Mihalcea and A. Csomai. Wikify!: Linking documents to encyclopedic knowledge. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, 2007.

[23] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[24] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS '13*, 2013.

[25] D. Milne and I. H. Witten. Learning to link with wikipedia. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, 2008.

[26] D. Nadeau and S. Sekine. A survey of named entity recognition and classification. *Linguisticae Investigationes*, 30(1):3–26, 2007.

[27] R. Navigli. Word sense disambiguation: A survey. *ACM Comput. Surv.*, 41(2):10:1–10:69, 2009.

[28] M. Paşca. Weakly-supervised discovery of named entities using web search queries. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, 2007.

[29] P. Pantel, T. Lin, and M. Gamon. Mining entity types from query logs via user intent modeling. In *The 50th Annual Meeting of the Association for Computational Linguistics*, 2012.

[30] A. Pilz and G. Paaß. From names to entities using thematic context distance. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, 2011.

[31] J. Pound, I. F. Ilyas, and G. Weddell. Expressive and flexible access to web-extracted data: A keyword-based structured query language. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, 2010.

[32] J. Pound, S. Paparizos, and P. Tsaparas. Facet discovery for structured web search: A query-log mining approach. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, 2011.

[33] L. Ratinov, D. Roth, D. Downey, and M. Anderson. Local and global algorithms for disambiguation to wikipedia. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, 2011.

[34] U. Sawant and S. Chakrabarti. Learning joint query interpretation and response ranking. In *Proceedings of the 22nd International Conference on World Wide Web*, 2013.

[35] S. Vigna. Quasi-succinct indices. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, 2013.

[36] J. X. Yu, L. Qin, and L. Chang. Keyword search in relational databases: A survey. *IEEE Data Eng. Bull.*, 33(1):67–78, 2010.