Entity Recommendations in Web Search

Roi Blanco[1], Berkant Barla Cambazoglu[1], Peter Mika[1], and Nicolas Torzec[2]

[1] Yahoo! Labs
Diagonal 177, 08018
Barcelona, Spain
{roi,barla,pmika}@yahoo-inc.com
[2] Yahoo! Labs
701 First Avenue
Sunnyvale, California, USA
torzecn@yahoo-inc.com

Abstract. While some web search users know exactly what they are looking for, others are willing to explore topics related to an initial interest. Often, the user's initial interest can be uniquely linked to an entity in a knowledge base. In this case, it is natural to recommend the explicitly linked entities for further exploration. In real world knowledge bases, however, the number of linked entities may be very large and not all related entities may be equally relevant. Thus, there is a need for ranking related entities. In this paper, we describe Spark, a recommendation engine that links a user's initial query to an entity within a knowledge base and provides a ranking of the related entities. Spark extracts several signals from a variety of data sources, including Yahoo! Web Search, Twitter, and Flickr, using a large cluster of computers running Hadoop. These signals are combined with a machine learned ranking model in order to produce a final recommendation of entities to user queries. This system is currently powering Yahoo! Web Search result pages.

1 Introduction

While there are many situations in which users know exactly what they are looking for and would like immediate answers, in other cases they are willing to explore and extend their knowledge. This is the case, for example, when learning about people in the news, following a long term interest in music, movies or sports or when exploring destinations for a future travel. There are many tools that help search users with finding the most precise formulation of their initial query, including suggestions for query expansions and information boxes providing direct answers. Similarly, we need tools for users who would like to browse.

Traditional search assistance tools support exploration by suggesting related queries, which is based on observing co-occurrences of exact queries in query sessions [6]. Yet, we know that most queries have simple, but recognizable internal structure and semantics. In particular, previous analysis has shown that over 50% web search queries pivot around a single entity that is explicitly named

H. Alani et al. (Eds.): ISWC 2013, Part II, LNCS 8219, pp. 33–48, 2013.

in the query [11]. These queries name an entity by one of its names and might contain additional words that disambiguate or refine the intent.

This observation leads to an alternate way of offering search assistance. Once we are able to identify the real-world entity that is being referenced in a query, and link it to a knowledge base, we can provide recommendations of related entities based on the relationships explicitly encoded in the knowledge base. Since knowledge bases also encode the types of entities and their relationships in a domain, such a tool can provide powerful additional features such as grouping related entities by type and explaining the relationships that are being presented to the user.

In this paper, we introduce a semantic search assistance tool named Spark, which exploits public knowledge bases from the semantic web, in combination with proprietary data, to provide related entity suggestions for web search queries. Our entity recommendation task is fundamentally a ranking task: given the large number of related entities in the knowledge base, we need to select the most relevant ones to show based on the current query of the user. Unlike the standard task addressed in semantic search where most related work has focused on, our goal is not to find information related to the user's current query but to recommend possible future queries to explore.

Our goal is to design a system that is scalable, efficient, and provides the best possible results for a large variety of queries. In the following, we will describe our system, and how we addressed the particular challenges involved. We show how we combine public RDF datasets as well as private resources to create a knowledge base that is both extensive and high quality. We also describe how we mine support for relationships from usage logs as well as user generated content, and how we combine this evidence in a single ranking function that works across entities and entity types. Lastly, we discuss how we disambiguate (link) entities when resolving queries online.

2 Related Entity Recommendations

We are given a (telegraphic) keyword query q that references (either by its name or an alias) a real-world entity e_i. We are given a knowledge base $K = \{(s, p, o)\}$ of subject, predicate and object triples where all subjects as well as objects are resources and it is possible to uniquely identify a single resource $r_i \in R$ as the representation of the entity, where R is the set of all resources in the KB, i.e., $R = \{s|(s, p, o) \in K\} \cup \{o|(s, p, o) \in K\}$. Further, we are given resources $\{r_j|\exists p : (r_i, p, r_j), r_j \in R\}$, i.e., resources related by asserted triples in the knowledge base. Our goal is to find the resource r_i, i.e., to disambiguate the query entity e_i, as well as rank the related resources $\{r_j\}$ to match the feedback from expert evaluators who provide judgment for a subset of the triples. Note that we do not consider for ranking all resources in the knowledge base but rely on the asserted triples. This way we only need to compute scores for $|(s, p, o)|$ items, the number of unique triples, instead of $|R|^2$ items, which is the number of potential resource pairs.

In building our system, we address a number of challenges that are not inherent in the above description but are potentially useful to other practitioners building similar systems. In reality, there is no single publicly available knowledge base that could serve the query needs of all of our users. Thus, our knowledge base needs to be constructed from multiple existing datasets, both public and private, to provide coverage and it needs to be regularly updated to address freshness, in particular in fast moving domains. Second, we are interested in solving this problem in the context of a web search engine. This means that we are also concerned with the user impact of the system, in particular what percentage of the query volume can be addressed and how users interact with the results. It also means that we need a system that is efficient both in offline processing and in the online serving stage.

3 Spark: An Entity Recommender System

In the remainder of this paper, we describe our system called Spark, which addresses the problem introduced above. The architecture of our system is shown in Fig. 1. The remainder of this section is organized in parts that match the components of this architecture. In Section 3.1, we describe how we construct our knowledge base. In Section 3.2, we characterize the features that provide evidence for the relevance of entities and relations. These features are combined using the state-of-the-art machine learning framework described in Section 3.3. The resulting ranking function is applied to all of the knowledge base to compute recommendations. In Section 3.4, we address our solution for serving the resulting data to end users, including disambiguation.

3.1 Knowledge Base Creation

Spark takes a large entity graph as input and applies a ranking function to extract a weighted subgraph consisting of the most important entities, their most important related entities, and their respective types. This entity graph is drawn from a larger Yahoo! knowledge graph, a unified knowledge base that provides key information about all the entities we care about and how they relate to each other.

Knowledge Acquisition. Entities, relations, and information about them are extracted from multiple complementary data sources. Data acquisition and information extraction are done on an ongoing basis, automatically. Data sources consist of web extractions, structured data feeds, and editorial content. Both open data sources and closed data sources from paid providers are leveraged. Reference data sources such as Wikipedia and Freebase provide background information for a wide variety of domains while domain-specific data sources provide rich information for domains such as Movie, TV, Music, or Sport. We use wrappers for extracting information from structured data feeds but use more advanced information extraction systems for other types of data sources. Wikipedia and

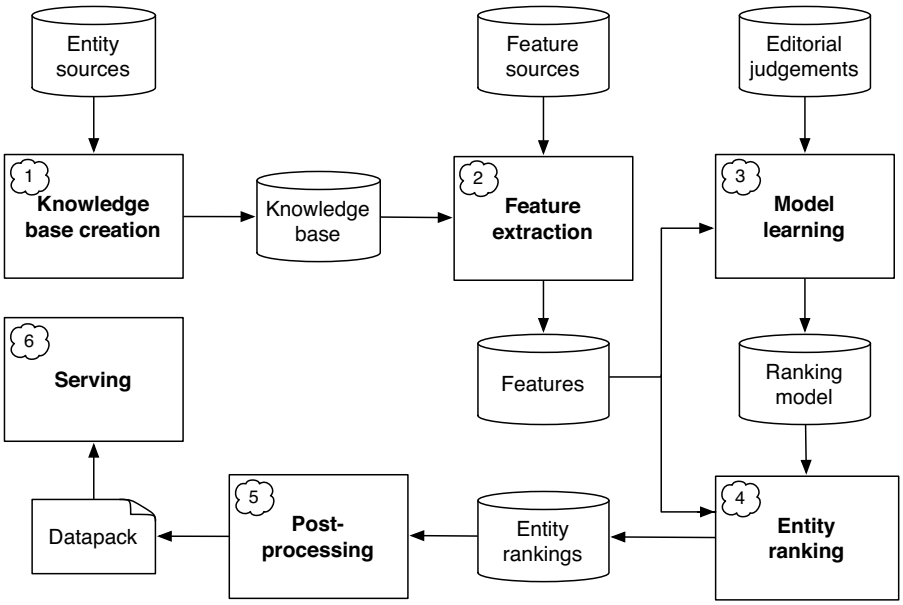

Fig. 1. High-level architecture of the Spark entity recommender system

Freebase are especially challenging because of their size, heterogeneity, complexity, and ever-changing nature. We monitor Wikipedia and Freebase continuously, and fetch new content (i.e., web extraction or RDF data dumps) on an ongoing basis. We extract structured data from Wikipedia (e.g., infoboxes, links, tables, lists, categories) using the DBpedia extraction framework and complementary information extraction solutions developed at Yahoo!. Most of our knowledge acquisition systems are distributed systems running on Hadoop.

Knowledge Base Construction. All of the entities, relations, and information that we extract are integrated and managed centrally in a unified knowledge base. Within this knowledge base, knowledge is modeled as a property graph with a common ontology. Our ontology was developed over 2 years by the Yahoo! editorial team and is aligned with schema.org. It consists of 250 classes of entities and 800 properties for modeling the information associated to them. When ingested into the knowledge base, entities, relations, and information associated to them are aligned with our common ontology. The knowledge base persists knowledge in a native graph database. Entities typically have an ID, a canonical name, aliases, alternate keys, types, and data properties. Relations typically have an ID, a subject, an object, a type, and data properties. We use editorial curation and knowledge reconciliation techniques (aka, record linkage, coreference resolution, link discovery) to match, de-duplicate and link together entities that refer to the same thing, especially across different data sources. Siloed, incomplete, inconsistent, and possibly inaccurate information are turned into a rich, unified, disambiguated knowledge graph. Today's knowledge graph focuses on the

Table 1. Spark input graph

Domain	# of entities	# of relations
Movie	205,197	9,642,124
TV	88,004	17,126,890
Music	294,319	77,673,434
Notable Persons	585,765	89,702
Sport	75,088	1,281,867,144
Geo	2,195,370	4,655,696
Total	3,443,743	1,391,054,990

domains of interest of key Yahoo! site, including the News domain (various types of entities), the Movie domain (movies, actors, directors, etc.), the TV domain (TV shows, actors, hosts, etc.), the Music domain (albums, music artists, etc.), the Sport domain (leagues, teams, athletes, etc.), and the Geo domain (countries, towns, points of interests, etc.).

Knowledge Export. In order to power Spark, we run offline graph queries on the knowledge graph to select domain-specific subgraphs, enrich them with additional relationships derived from the subgraphs (e.g. we leverage transitivity and materialize derived relationships), and export the resulting subgraphs to Spark. Overall, the graph that Spark uses as input consists of 3.5M entities and 1.4B direct and indirect relations from the Movie, TV, Music, Sport and Geo domains. Table 1 provides domain-specific numbers.

3.2 Feature Extraction

For every triple in the knowledge base, Spark extracts over 100 features. The extracted features can be grouped under three main headings: co-occurrence, popularity, and graph-theoretic features. Spark also extracts a few additional features that do not fall into these three categories. Some features are unary, i.e., relate to the importance of an entity on its own while some features are binary and capture the strength of the relation between entities.

Co-occurrence Features. The features in this set are motivated by the fact that entities which frequently occur together in a given set of observations (i.e., sets of short text pieces) are more likely to be related to each other. In Spark, we use three different text sources to extract the co-occurrence information: Yahoo! Web Search, Twitter, and Flickr. In case of Yahoo! Web Search, each query is treated as an individual observation, and we identify pairs of entities that appear together in the query. For example, in query "flight from barcelona to madrid", "Barcelona" and "Madrid" are identified as two entities that occur together.[1] In addition to query terms, we extract more coarse-grained co-occurrence information, relying on search sessions of users. In this case, all query terms issued in a search session form a single observation. In case of Twitter and Flickr,

[1] To recognize entities, we extract all possible subsequences of terms from the text and check for their presence in a dictionary which is built using the input entity data.

the observations correspond to tweets and user tags associated with photos, respectively.

For every given related entity pair, we compute a number of statistical features using the co-occurrence information, separately, for each distinct sets of observations mentioned above. The two important features are joint and conditional occurrence probabilities. Since the latter is not symmetric, we also compute the reverse conditional probability as another feature. In addition, we compute these features at the level of users by treating all observations associated with a user as an individual observation. Finally, we extract some statistical features that exploit various probability distributions, such as probabilistic mutual information, KL divergence, and entropy.

Popularity Features. The popularity features simply represent the frequency of an entity in a given data source. We compute the frequency in two different ways, based on the entity string (e.g., "brad pitt") or the Wikipedia ID associated with the entity (e.g., "Brad_Pitt"). In the former case, the frequency information is obtained from the previously mentioned sets of observations: queries, query sessions, tweets, and photo tags. We also compute the number of matching results in Yahoo! Search when the entity string is used as a query. In the latter case, we identify the Wikipedia URL that corresponds to the entity and compute the frequency at which this URL is viewed in web search results. Similarly, we compute the click frequency of the URL. Note that all popularity features are computed both for the subject and the object of the triple.

Graph-Theoretic Features. We compute features on two types of graphs. We first build an entity graph, where vertices represent entities (more specifically, entity IDs) and there is an edge between two vertices if the corresponding entities are connected through a relationship. We also form a hyperlink graph obtained from a large web page collection. In both graphs, we run the PageRank algorithm and compute authority scores for entities. We use the entity graph to compute also the number of shared vertices (common neighbors) between two entities.

Other Features. The additional features include types of entities and types of their relations as well as the number of terms in the entity string. We also create features using various linear combinations of the features mentioned before.

Feature extraction is implemented as a series of Hadoop MapReduce jobs, where we reuse basic statistics computed at the beginning of the pipeline to speed up the computation of features that rely on similar statistics. The extracted feature vectors are the sole input to the ranking process described next.

3.3 Ranking

Spark makes use of learning to rank approaches in order to derive an efficient ranking function for entities related to a query entity. In general, systems that are able to accommodate a large number of features benefit from automated approaches to derive a way to combine feature values into a single score. This is at the expense of needing enough quality training data to be able to generalize well and perform meaningful predictions.

Formally, the goal of the ranking system in Spark is to learn a function $h(\cdot)$ that generates a score for an input query q_i and an entity e_j that belongs to the set of entities related to query $e_j \in \mathcal{E}^{q_i}$. Both q_i and e_j are represented as a feature vector $\mathbf{w_{ij}}$ that contains one entry per extracted feature. The input of the learning process consists of training data of the form $\{T(q_i) = \{\mathbf{w_{ij}}, l_{ij}\}\}_{q_i \in \mathcal{Q}}$, where $l_{ij} \in L$ is a manually assigned label from a pre-defined set. Spark uses a five-level label scale ($l \in$ *Bad, Fair, Good, Perfect, Excellent*), and the assignment from examples (q_i, e_j) is done manually by professional editors, according to a pre-defined set of judging guidelines. The query set $\mathcal{Q}$ is comprised of editorially picked entities and random samples from query logs, which is expected to mimic the actual entity and query distribution of the live system. The training set might also contain preference data. In this case, the labels indicate that an entity is preferred over another entity for a particular query: $\{T(q_i) = \{\mathbf{w_{ij}}, l_{ij}\}\}_{q_i \in \mathcal{Q}}$. The ranking function has to satisfy the set of preferences as much as possible and, at the same time, is has to match the label so that a particular loss function (in our case, square loss) is minimized for a set of test examples.

We employ stochastic gradient boosted decision trees (GBDT) [3,4] for entity ranking (similar to [13]). In brief, gradient tree boosting creates an ensemble of decision trees (weak learners) using an iterative boosting procedure. At each iteration, the algorithm creates a new regression tree that is fitted to the gradient of the loss function. Among the advantages over other learning methods (shared by decision trees) is that they are easy to interpret. In general, it is possible to calculate the relative importance of each input variable (feature) and which are more influential in computing the function h [4]. On the other hand, stochastic GBDTs can be trained on a randomly selected subset of the available data and are less prone to overfitting.[2] GBRank is a variant of GBDT that is able to incorporate both label information and pairwise preference information into the learning process [12], and is the function of choice we adopted for ranking in Spark.

3.4 Post-processing and Serving

Disambiguation. The final system needs to provide a mapping between queries and the entity that must be triggered. In Spark, query strings used as triggers are mainly derived from entity names and a fixed set of context terms, e.g. "brad pitt actor" for the entity "Brad_Pitt". We also use a list of aliases computed from query logs, which, for example, provides a mapping between the alias "jlo" and the entity "Jennifer_Lopez".

In post-processing, we address the issue of disambiguation among triggers that may refer to different entities. In practice, certain entity strings may match multiple entities (e.g., "brad pitt" may refer to the actor entity "Brad_Pitt" or the boxer entity "Brad_Pitt_(boxer)"). Moreover, there may be cases with a

[2] Overfitting refers to a statistical model picking up random noise from spurious patterns in the data, and it is unable to predict unseen examples well. In general, this happens when the model is unnecessarily complex (it has too many free variables).

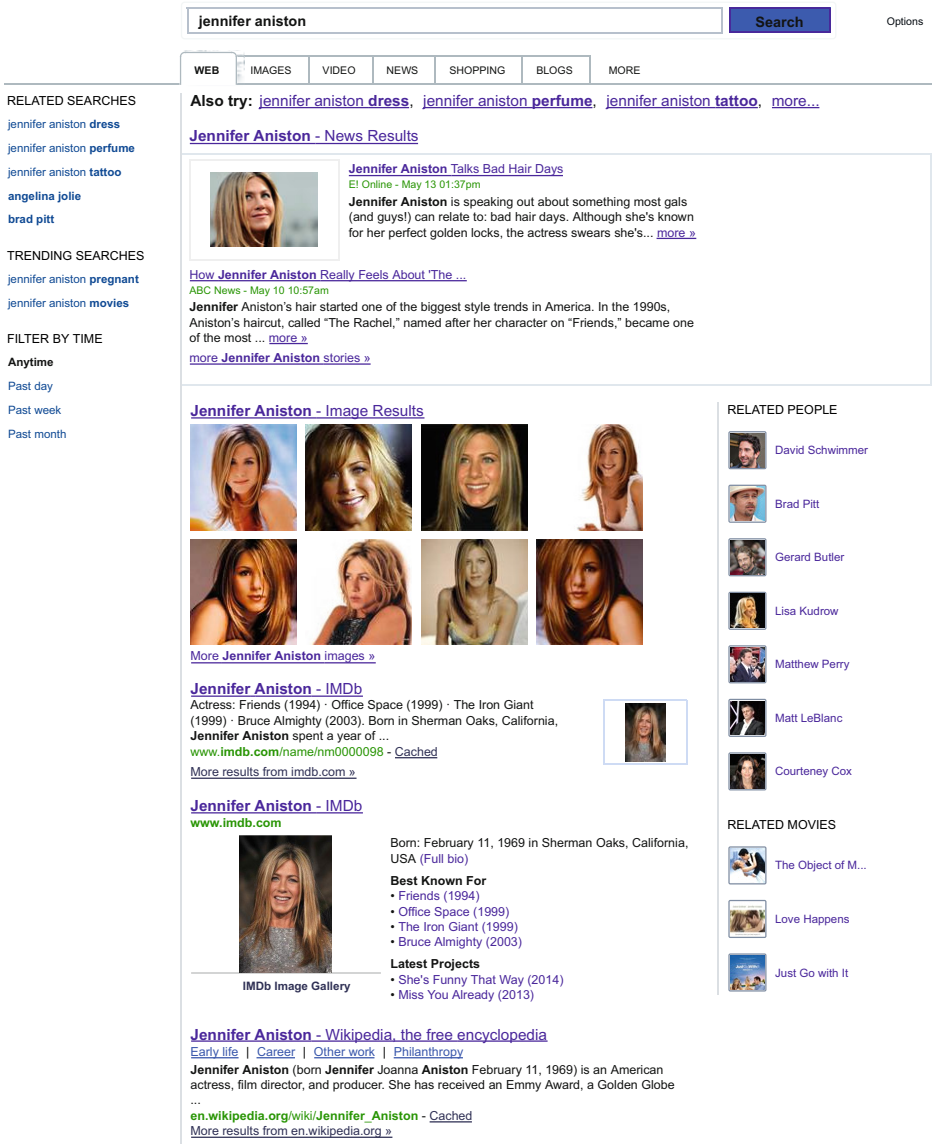

Fig. 2. Search result page for the query "jennifer aniston". Spark results (persons and movies) appear on the right.

common meaning for the string (e.g., the entity "XXX_(movie)" is not the most likely intent for query string "xxx"). Hence, the problem here is to identify the most likely intent for a given entity string. To this end, we adopt a simple yet very effective disambiguation strategy. We define the view count of an entity as the number of times its respective Wikipedia page is viewed in the search results. Given the set of entities matching a query string, we pick only the entity with the highest view count as the intended meaning. Moreover, through a simple linear model using as parameters the view count of the entity and the frequency of the query string, we decide whether the entity corresponds to the most likely meaning in order to avoid matching common concepts. For example, the query "management" should not trigger the entity "Management_(film)" because there exists a more general concept of management represented by the Wikipedia entity "Management". The datapack contains a mapping between a query string and the ranking of an entity only if the entity is the most likely meaning for the query string with sufficiently high probability.

Serving. Once the final datapack is generated, it is deployed in the frontend system. This involves loading the materialized rankings in the datapack into an online serving system, which is essentially a fielded inverted index over subjects, predicates, and objects. This system provides state-of-the-art (<100 ms) retrieval latency for online query processing and can sustain sufficient throughput when serving production query traffic. The user interface of Spark currently displays only the names of related entities and small thumbnail images which correspond to the displayed entity.

Figure 2 shows the related entity recommendations made by Spark in Yahoo! web search. Related entities in the context of web search are historically called facets (see e.g. [13]) due to their similarity in display to faceted search. This is somewhat of a misnomer, in that these facets are not narrowing or broadening the original query. In our case, after clicking one of the related entities a new query is launched with the related entity.

4 Evaluation

In the following, we report on some of the evaluations we have carried out using both relevance assessments and usage data. These evaluations are regularly repeated for each new release of the system, for example, when adding new or improved data to the knowledge base or incorporating new features into the ranking function.

4.1 Relevance Assessment

We carried out a number of experiments to assess ranking with GBrank within the context of our task. In the following, we describe how we optimize the model, the training data required, how it influences performance, and what are the most important features, as decided by the model, to assess relevance.

Table 2. Number of labeled examples

Type	# of examples	Proportion
Locations	24,843	46.61%
People	23,198	43.52%
Movies	4,346	8.15%
TV shows	785	1.47%
Sport teams	120	0.22%
Total	53,292	100.00%

GBDTs have a number of tunable parameters that will have an impact on the final performance. Thus, we split the training data into ten folds and decide on the final values for those parameters using cross-validation. These parameters include the number of nodes and number of trees, which in general are the ones that impact performance the most in our setting. The parameter values used for the final model are the ones that maximize a particular choice of a relevance metric. We settled on Normalized Discounted Cumulative Gain (NDCG) [5] as the final performance metric we optimize for, as it is capable of handling different relevance grades. NDCG is defined for a cut-off level k as:

$$nDCG@k = \frac{DCG@k}{DCG_{\text{ideal}}@k}, \tag{1}$$

where

$$DCG@k = \sum_{i=0}^{k} \frac{2^{g(l_i)} - 1}{\log_2 g(l_i) + 1}\ , \tag{2}$$

and $DCG_{\text{ideal}}@k$ is the maximum attainable DCG value, $g(l_i)$ is the gain assigned to label l_i.

In order to collect training data, we regularly sample queries submitted to Yahoo! Web Search. The entities evaluated for a particular query are bootstrapped from the current production models of Spark. Results are evaluated manually by a group of trained expert editors on different dimensions, most notably including relevance, freshness, and timeliness. The editorial judgments came after several sessions in which they agree to a fine-grain set of guidelines. The judgments were not finalized until the inter-judge agreement is sufficiently high.

Table 2 breaks down the amount of training data per query class (recall that each query belongs to a particular entity class, as it must contain a particular entity for Spark to trigger). There is a clear bias in the training data towards popular entity types. Given that in general more training data reflects in a better ranking outcome, we ask for more labeled examples of those types.

Table 3 reveals the important features, as decided by GBDT (see [4] for a detailed explanation on how importance is computed). We observe that type is the most important feature for the model. This is somewhat expected since some relation types (e.g., being the spouse of someone) are much stronger than others (e.g., acting together in a movie). We note that this feature is picked by the model for the first split and this also reflects the fact that different signals for ranking naturally differ in their importance across entity types, as noted in [1].

Table 3. Top-10 features for ranking sorted by their importance

Rank	Feature description	Importance
1	Type of the relation	100.0
2	Frequency of the related entity in Flickr photo tags	70.3
3	Frequency of the related entity in search query terms	54.8
4	Conditional probability in Flickr photo tags	51.6
5	PageRank of the query entity in the entity graph	45.8
6	Probability of the related entity in the search sessions	44.6
7	Number of words in the related entity name	39.1
8	Number of search results matching the related entity	35.9
9	Conditional probability in search sessions	33.6
10	PageRank of the related entity in the entity graph	33.4

Table 4. Retrieval performance per query type

	NDCG		
Type	@1	@5	@10
Locations	0.776	0.820	0.872
Movies	0.910	0.958	0.960
People	0.863	0.906	0.937
Sports	0.941	0.941	0.955
TV shows	0.882	0.954	0.967

Signals computed from the entity graph also make a remarkable contribution to the final ranking model, as well as co-occurrence features, especially those computed over Flickr data and user query sessions.

In Table 4, we display NDCG values per query type. Performance numbers are cross-validated by splitting the available data into ten folds, each fold using 1/10th of the data for testing and the rest for training. The final values are averaged throughout the ten folds. Note that ranking related entities for locations is significantly more difficult than for any other entity type, despite of being the one having the largest fraction of training data.

Figure 3 shows how NDCG is impacted when the amount of training data is increased (the x-axis reflects the number of examples). The initial performance values are high because the evaluated editorial data is computed over an instance of the model that is already well performing. In any case, the increase is log-linear with respect to the number of training examples, meaning that improvements are exponentially harder to achieve just by adding more training data.

4.2 Usage Evaluation

Click-through rate (CTR) and coverage are two commonly used metrics for assessing the impact of a recommendation system on final users and query volume. CTR is defined as $\frac{\text{clicks}_{\text{spark}}}{\text{views}_{\text{spark}}}$ and coverage is defined as $\frac{\text{views}_{\text{spark}}}{\text{queries}}$, where the variables refer to the number of clicks on the Spark module, the number of views (queries that triggered the Spark module), and the total number of queries

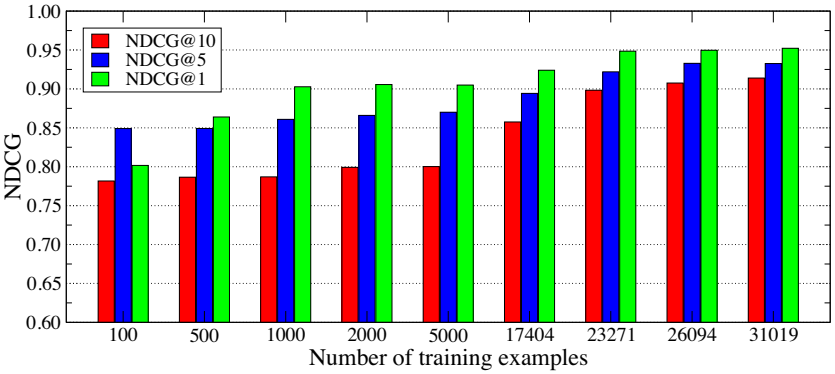

Fig. 3. Retrieval performance with varying amount of training data

submitted to the search engine. The coverage metric indicates the fraction of queries for which we display an entity ranking in the result page. The CTR metric indicates the likelihood that the user will click on an entity link or image when a Spark ranking is displayed in the result page. The product of these two metrics gives us the fraction of query volume generated by Spark.

Figures 4 and 5 display the point-wise values of CTR and coverage metrics for a period of 82 days, which is broken up into two parts: the first half is before Spark was launched in production (replacing the previous system) and the second half is right after Spark was deployed (marked on the graphs) in February, 2013. Due to the confidential nature of the data, the values shown on the y axes are normalized by the mean values. There are a number of take away messages from these results. The coverage trend before and after deployment is flat. This is because there is no change in the number of queries that trigger Spark during the observation period, except for the deployment day of Spark (we observe a quick jump in coverage on that day). There are two periods of system stabilization after deployment, which are apparent from the dips in coverage. Regarding CTR, before deployment, the module was gradually attracting less and less clicks over the time, most likely because the data was not fresh. The Spark deployment changed the CTR trend, due to improved ranking and higher data quality. There is a positive trend effect that is added up over time, because users were increasingly engaged with the renewed module.

5 Lessons Learned

Feature Importance. The performance of individual features show variation depending on the type of an entity or the type of the relation between entities. For example, for movies, useful popularity or co-occurrence information can be obtained by the features extracted from the query logs, whereas the features obtained from the photo tags are not very helpful for entities of this type. This explains the reason why the relation type is the most important feature.

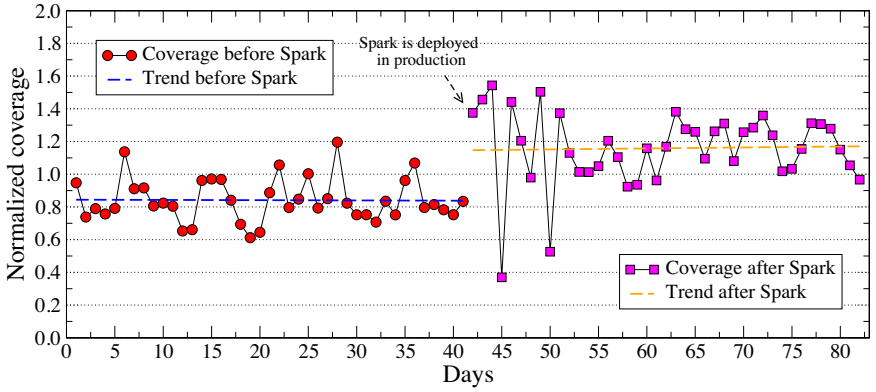

Fig. 4. Coverage before and after Spark is deployed in production

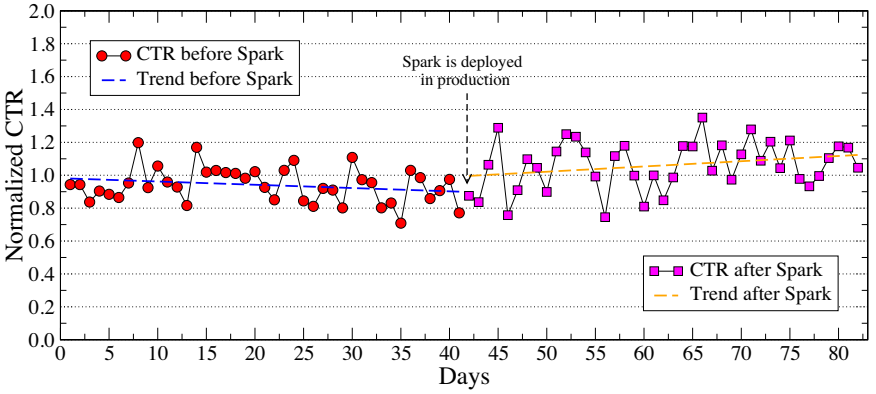

Fig. 5. CTR before and after Spark is deployed in production

Feature Sources. Flickr turns out to be the most important source of information for co-occurrence. It is relatively high quality and provides the highest coverage, especially for locations. Search logs provide limited co-occurrence information because most queries do not contain more than one entity. In general, the Twitter data is not very helpful because of the informal and noisy nature of the tweets.

Weak Relations. In our initial attempts, we experimented without any pruning on the relations. It later turned out that certain weak relations (for example, voice actors of the same movie) introduce too much noise into the generated rankings. Eventually, we excluded some weak relations from our input data.

Coverage vs. CTR. The trade-off between coverage and CTR is important as these metrics often act against each other: Increased coverage most often implies decreased CTR and vice versa. As an example, poor disambiguation implies increased coverage, but has a significant negative impact on CTR.

CTR per Type. We observe that person entities are more likely to be clicked by the users. Location entities are the second most important in terms of the CTR metric. In general, albums receive relatively fewer clicks.

6 Related Work

Previous methods for generating query recommendations (search assistance) have focused on the analysis of query logs at the level of entire query strings or tokens, without any concern for the linguistic or semantic structure of query strings [6]. However, more recent work has recognized that queries have internal structures and can be classified based on their semantic intent [8,9]. These observations have led to the development of the area of semantic search.

The broad area of semantic search in general refers to finding information in knowledge bases using unstructured, keyword queries typical for search engines. The task we address in this paper can be placed in the context of the related work as shown in Fig. 6. Within semantic search, three broad tasks have attracted attention so far. Entity search or ad-hoc object retrieval as defined by Pound et al. [11] requires finding the resources in a knowledge base that refer to an entity explicitly named in the query (e.g., finding the resource `http://dbpedia.org/resource/Brad_Pitt` for the query "brad pitt actor"). While this task may seem rather trivial, it encapsulates many of the key challenges in semantic search, in particular query interpretation. This task has been the focus of evaluations at the Semantic Search Challenge evaluations organized in 2010 and 2011.[3].

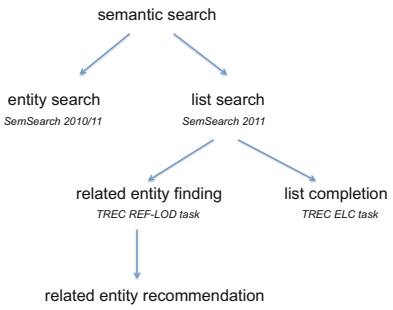

Fig. 6. Taxonomy of semantic search tasks

Semantic search becomes more complex and closer to question answering in general when the query only provides a description of the target entity. This is generally referred to as category or list search. An example is the query "actors who played in fight club". List completion is a variant of this task where some instances of the target list are explicitly given. Related entity finding is another variant where the list is defined by the list member's relationship to a single entity. The Related Entity Finding in Linked Open Data (REF-LOD) task at the TREC Entity Track in 2010 and 2011 are of this type. In this particular case, the type of relation to the target entity as well as the type of the target entity were both given as constraints. An example query from 2011 was "What recording companies now sell the Kingston Trio's songs?". Here, the name of a musical band is given, and the task is to find entities of a given type that are related by selling the songs of this band. In the related entity recommendation task we consider neither the relation type nor the type

[3] `http://semsearch.yahoo.com/`

of the target entity are specified as constraints. A further difference is that the systems in the Entity Track have had to perform relation extraction from text, because the relationship between the two entities were not explicitly specified in a knowledge base but had to be discovered from text. In our case, we are ranking relations that are explicitly defined in our Knowledge Base and consider only the task of ranking those relations. In other words, our task is recommendation as opposed to extraction or finding.

Although recommendations similar to ours appear on search result pages of Google and Bing, details of these systems have not been published. Previous versions of our system have been described in [7,13]. These papers have focused largely on ranking and provided limited descriptions of the overall process of generating entity recommendations. Our system has evolved considerably since the publication of these papers, including the ranking model, which does not rely any more on a click-based objective as described in [7]. There are a few other studies on entity recommendation as well [2,10]. However, in general, these studies are very limited in terms of their scale or target a different application domain. For example, the work in [10] relies on a single graph feature and evaluates it in a very specific domain. The system described in [2] suggest entities for a given long piece of text containing concepts (e.g., a web page).

7 Future Work

We have presented Spark, a semantic entity recommendation engine. Spark provides a ranking of related entities based on a knowledge base and signals mined from multiple sources, including the knowledge base itself as well as user-generated text from Yahoo! Web Search, Twitter and Flickr. Although no baselines exist for comparison, we have described our experiments in evaluating and optimizing Spark against manually created relevance assessments and usage data. Our framework is generally applicable to ranking related entities in RDF data, as it only relies on typed entities, relationships and natural language labels for entities. Though some of the sources we have used to extract ranking features may not be generally available, they can be substituted by others without changing the general architecture of the system.

Our system is currently live in Yahoo! Web Search in the US, as well as in other English language markets, Taiwan, Hong Kong and Spain. We are continuously improving the system by incorporating new data sources as well as devising new features. In the future, we are also planning to significantly extend the types of queries we are able to answer. At the moment, our system only handles entity queries such as "brad pitt actor", while in the future we would also like answer list queries such as "brad pitt movies" and "brad pitt movies 2010". The ultimate scope of our work is question-answering, i.e., answering arbitrary keyword queries regarding entities on the Web. Finally, the rankings in our system are computed in an offline manner without considering the users intent or interests. Our ongoing efforts are on generating the rankings online, taking into account any kind of feedback extracted from users' search sessions.

Acknowledgment. We would like to acknowledge members of the Yahoo! Search product management team (in particular, Libby Lin), our editorial support (Alice Swanberg) and the members of the Taiwan search engineering team (Gibson Yang, Rong-En Fan, Yikai Tsai).

References

1. Blanco, R., Mika, P., Vigna, S.: Effective and Efficient Entity Search in RDF Data. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 83–97. Springer, Heidelberg (2011)
2. Damljanovic, D., Stankovic, M., Laublet, P.: Linked data-based concept recommendation: comparison of different methods in open innovation scenario. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, O., Presutti, V. (eds.) ESWC 2012. LNCS, vol. 7295, pp. 24–38. Springer, Heidelberg (2012)
3. Friedman, J.H.: Stochastic gradient boosting. Computational Statistics and Data Analysis 38, 367–378 (1999)
4. Friedman, J.H.: Greedy function approximation: A gradient boosting machine. Annals of Statistics 29, 1189–1232 (2000)
5. Järvelin, K., Kekäläinen, J.: IR evaluation methods for retrieving highly relevant documents. In: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 41–48. ACM, New York (2000)
6. Jones, R., Rey, B., Madani, O., Greiner, W.: Generating query substitutions. In: Proceedings of the 15th International Conference on World Wide Web, pp. 387–396. ACM, New York (2006)
7. Kang, C., Vadrevu, S., Zhang, R., van Zwol, R., Pueyo, L.G., Torzec, N., He, J., Chang, Y.: Ranking related entities for web search queries. In: Proceedings of the 20th International Conference Companion on World Wide Web, pp. 67–68. ACM, New York (2011)
8. Lin, T., Pantel, P., Gamon, M., Kannan, A., Fuxman, A.: Active objects: actions for entity-centric search. In: Proceedings of the 21st International Conference on World Wide Web, pp. 589–598. ACM, New York (2012)
9. Mika, P., Meij, E., Zaragoza, H.: Investigating the semantic gap through query log analysis. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 441–455. Springer, Heidelberg (2009)
10. Passant, A.: dbrec: music recommendations using DBpedia. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part II. LNCS, vol. 6497, pp. 209–224. Springer, Heidelberg (2010)
11. Pound, J., Mika, P., Zaragoza, H.: Ad-hoc object retrieval in the web of data. In: Proceedings of the 19th International Conference on World Wide Web, pp. 771–780. ACM, New York (2010)
12. Zheng, Z., Zha, H., Zhang, T., Chapelle, O., Chen, K., Sun, G.: A general boosting method and its application to learning ranking functions for web search. In: Proceedings of Neural Information Processing Systems, pp. 1697–1704 (2008)
13. van Zwol, R., Pueyo, L.G., Muralidharan, M., Sigurbjörnsson, B.: Ranking entity facets based on user click feedback. In: Proceedings of the 4th IEEE International Conference on Semantic Computing (2010)