Ranking Related News Predictions

Nattiya Kanhabua*
Dept. of Computer Science
Norwegian University of
Science and Technology
Trondheim, Norway
nattiya@idi.ntnu.no

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

Michael Matthews
Yahoo! Research
Barcelona, Spain
mikemat@yahoo-inc.com

ABSTRACT

We estimate that nearly one third of news articles contain references to future events. While this information can prove crucial to understanding news stories and how events will develop for a given topic, there is currently no easy way to access this information. We propose a new task to address the problem of retrieving and ranking sentences that contain mentions to future events, which we call *ranking related news predictions*. In this paper, we formally define this task and propose a learning to rank approach based on 4 classes of features: term similarity, entity-based similarity, topic similarity, and temporal similarity. Through extensive evaluations using a corpus consisting of 1.8 millions news articles and 6,000 manually judged relevance pairs, we show that our approach is able to retrieve a significant number of relevant predictions related to a given topic.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Retrieval models*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Performance evaluation (efficiency and effectiveness)*

General Terms

Algorithms, Experimentation, Performance

Keywords

News predictions, Future events, Sentence retrieval and ranking

1. INTRODUCTION

Predicting the future has long been the holy grail in the financial world. The leaders of large organizations need to analyze information related to the future in order to identify the key challenges that can directly affect their organizations. However, it is not just businesses that care about the future - all people have anticipation and curiosity about the future. Canton [8] describes the future trends that can influence our lives, our jobs, our businesses, and even our world. These include the energy crisis, the global financial crisis, politics, health care, science, securities, globalization, climate changes, and technologies. When people read news stories on any of these topics whether it is an article about war in the middle east or the latest health care plan, they are naturally curious about potential future events. How long will the war last? How much will it cost? What happens if we do nothing at all? This obsession with the future is also reflected in the news articles themselves - our analysis of one year worth of news from over 100 sources indicates that nearly one third of news articles contain at least one statement made about a future date.

Accessing this information in an intuitive way would greatly improve how people read and understand news. In this paper, we define a new task we call *ranking related news predictions* that directly addresses this problem by finding all predictions related to a news story in a news archive and ranking them according to their relevance to the news story. This task is motivated by the desire of news sites to increase user engagement by providing content that directly addresses the information needs of users. By providing links to relevant content, new sites can keep users on their site longer thus increasing the likelihood that users will click on revenue generating links and also improving user satisfaction. For a wide range of news events from natural disasters to political unrest in the middle east, the information need - the question most on people's minds - is what is going to happen next. This new task is a first step toward helping people answer this very question by finding and linking to predictions that are relevant to the user.

Our query is extracted from a news article currently read by a user, and is composed of a bag of *entities* or *terms*. Using an automatically-generated query, predictions are retrieved, ranked over the time dimension, and presented to the user. Note that there are a number of future-related information analyzing tools including *Recorded Future*[1], and *Time Explorer* [22]. *Recorded Future* extracts predictions from different sources (news publications, blogs, trade publications, government web sites, and financial databases). A user creates a query by selecting a topic of interest (e.g. a topic about "Financial Markets"), and then specifying an en-

*Work performed while intern at Yahoo! Research

[1] https://www.recordedfuture.com/

SIGIR'11, July 24–28, 2011, Beijing, China.

tity (people, companies, or organizations) from a set of "predefined" entities. The system will then retrieve predictions related to the selected topic and entity. A major difference with our system is that *Recorded Future* requires a query specified in advance, while our system automatically creates a query for the user based on the news article being read and it is not limited to "predefined" entities. *Time Explorer* is a search engine that allows users to see how topics have evolved over time and how they might continue to evolve in the future. The system extracts predictions from document collections and allows users to search for them using ad-hoc queries. However, neither *Time Explorer* nor *Recorded Future* provide details of how predictions are ranked nor do they evaluate performance in a formal setting as we do here.

The main contributions of this paper are: 1) the first formalization of the *ranking related news predictions* task, 2) an evaluation dataset with over 6000 relevance judgments from the New York Times Annotated Corpus[2] with queries that are selected from real-world future trends [8] 3) a learned ranking model incorporating four classes of features including term similarity, entity-based similarity, topic similarity, and temporal similarity 4) an in-depth analysis of feature selection to guide further research in the *ranking related news predictions* task.

The organization of the rest of the paper is as follows. In Section 2, we explain our system architecture, and outline the models for annotated documents, predictions as well as queries. In Section 3, we propose four classes of features used for learning a ranking model. In Section 4, we describe our ranking model. In Section 5, we evaluate the proposed ranking model. In Section 6, we give an overview of related work. Finally, in Section 7, we conclude and outline future work.

2. PROBLEM DEFINITION

In this section, we outline the system architecture, and give the formalization of the models for annotated documents, predictions, and queries.

2.1 System Architecture

Figure 1 depicts our system which retrieves a set of predictions (sentences containing future dates) related to a given news article. Predictions can be extracted from a temporal document collection – any collection that contains timestamped documents, e.g., personal emails, news archives, company websites and blogs. In this work, we automatically extract predictions from news archives using different annotation tools. Our *document annotation process* includes tokenization, sentence extraction, part-of-speech tagging, named entity recognition, and temporal expression extraction. The result of this process is a set of sentences annotated with named entities and temporal expressions, which will be indexed as *predictions* for further processing or retrieval.

A key component of the annotation process is the extraction of temporal expressions using a time and event recognition algorithm. The algorithm extracts temporal expressions mentioned in a document and normalizes them to dates so they can be anchored on a timeline. As explained in [1], there are three types of temporal expressions: explicit, implicit and relative. An explicit temporal expression mentioned in a document can be mapped directly to a time point or interval, such as, dates or years on the Gregorian calendar. For example, "July 04, 2010" or "January 01, 2011" are explicit temporal expressions. An implicit temporal expression is given in a document as an imprecise time point or interval. For example, "Independence Day 2010" or "New Year Day's 2011" are implicit expressions that can be mapped to "July 04, 2010" or "January 01, 2011" respectively. A relative temporal expression mentioned in a document can be resolved to a time point or interval using a time reference - either an explicit or implicit temporal expressions mentioned in a document or the publication date of the document itself. For example, the expressions "this Monday" or "next month" are relative expressions which we map to exact dates using the publication date of the document.

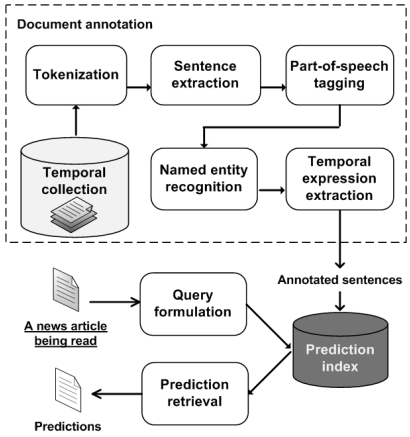

Figure 1: Prediction retrieval system architecture.

Instead of having an explicit information need provided, we automatically generate a query. In this case, we assume that the user's information needs lie in *the news article being read* by the user, and a query will be extracted from this news article (further details are given in Section 2.4). For a given news article, we retrieve predictions that are relevant to the news article, that is, relevant sentences containing future dates with respect to the publication date of the news article being read.

Retrieved predictions are ranked by the degree of relevance, where a prediction is "relevant" if it is future information about *the topics of the news article*. Note that we do not give any specific instructions about how the dates involved are related to relevance. Nevertheless, we hypothesize that predictions extracted from more recent documents are more relevant. In this paper, we use a machine learning approach [20] for learning the ranking model of predictions. This involves identifying different classes of features (see Section 3) to measure the relevance of a prediction with respect to the news article.

2.2 Annotated Document Model

Our document collection contains a number of news articles defined as $C = \{d_1, \ldots, d_n\}$. We treat each news article as a bag-of-words (an unordered list of terms, or features), $d = \{w_1, \ldots, w_n\}$. $time(d)$ is a function given the creation or publication date of d. Some of our proposed features are extracted from annotated documents, which are defined

[2] http://www.ldc.upenn.edu/Catalog/docs/LDC2008T19/new_york_times_annotated_corpus.pdf

Table 1: Example of a prediction with field/value pairs.

Field	Value
ID	1136243_1
PARENT_ID	1136243
TITLE	*Gore Pledges A Health Plan For Every Child*
TEXT	*Vice President* **Al Gore** *proposed today to guarantee access to affordable health insurance for all children by 2005, expanding on a program enacted two years ago that he conceded had had limited success so far.*
CONTEXT	*Mr. Gore acknowledged that the number of Americans without health coverage had increased steadily since he and President Clinton took office.*
ENTITY	Al Gore
FUTURE_DATE	2005
PUB_DATE	1999/09/08

as follows. Each document d, has an associated annotated document $\hat{d}$, which will consist of three sets, $\hat{d}_e$, $\hat{d}_t$, $\hat{d}_s$: a set of named entities $\hat{d}_e = \{e_1, \ldots, e_n\}$, where each entity $e_i \in \mathcal{E}$ and $\mathcal{E}$ is the complete set of entities (typed as person, location, and organization) in the collection; a set of annotated temporal expressions $\hat{d}_t = \{t_1, \ldots, t_m\}$ and a set of sentences $\hat{d}_s = \{s_1, \ldots, s_z\}$

2.3 Prediction Model

A prediction p can be viewed as a sentence containing field/value pairs of annotation information and we define d^p as the *parent* document where p is extracted from. We define several fields for a prediction including ID, PARENT_ID, TITLE, ENTITY, FUTURE_DATE, PUB_DATE, TEXT, and CONTEXT. The field ID specifies a prediction's unique number, PARENTID and TITLE represent a unique number and the title of d^p respectively ENTITY contains a set of annotated entities $p_{entity} \subset \hat{d}_e$, FUTURE_DATE consists of "future" temporal expressions p_{future} annotated in p, PUB_DATE is the publication date of the *parent* document d^p and TEXT is a prediction's text p_{txt} or the sentence of p. Note that each prediction must contain at least one "future" temporal expression, that is, $p_{future} \neq \emptyset$. In addition, we explicitly model the context of the prediction p_{ctx}, represented by the field CONTEXT and defined as surrounding sentences of the main sentence [6]. In our work, we define the context p_{ctx} as the sentence immediately before and the one immediately after p_{txt}. Table 1 contains an example of a prediction with its field/value pairs.

2.4 Query Model

As mentioned earlier, a query q is automatically generated from a news article being read d^q; q is composed of two parts: keywords q_{text}, and the time of query q_{time}. The keywords q_{text} are extracted from d^q in three ways resulting in three different types of queries.

The first type of query is a bag of *entities*, noted as "entity query" or Q_E where its q_{text} is composed of the *top-m* entities (ranked by frequency) extracted from d^q. Intuitively, we want to know whether using only *key* entities frequently mentioned in the news article can retrieve relevant predictions with high precision or not. For example, given an actual document about "President Bush and the Iraq war", we extract Q_E with $q_{text} = \langle George\ Bush, Iraq, America \rangle$. At retrieval time, q_{text} will be matched with the ENTITY field of the predictions.

The second query is denoted "term query" or Q_T where its q_{text} is composed of *top-n* terms (ranked by term weighting, i.e., TF-IDF) extracted from d^q. Q_T is considered a bag of *terms* important to both d^q (locally) and the whole collection (globally). In contrast to the previous query type, Q_T aims at retrieving predictions related to the topics of news article, which can be represented as *a set of informative terms*. As an example, the Q_T with $q_{text} = \langle poll, bush, war, iraq \rangle$ is extracted from the same document used in the Q_E example above. In this case, q_{text} will be matched with the TEXT field of the predictions.

The last type is called "combined query" or Q_C where its q_{text} is a combination of both *top-m* entities and *top-n* terms formed by concatenating Q_E and Q_T. We discuss how we select *top-m* and *top-n* in Section 5.

The last component of the query is the *temporal criteria* or q_{time} used for retrieving predictions on the time dimension; q_{time} is composed of two different time constraints. The first constraint is specified in order to retrieve only predictions that are *future* relative to the publication date of query's parent article, or $time(d^q)$. The second constraint indicates that those predictions must belong to news articles published before $time(d^q)$. Both time constraints will be represented using a time interval, i.e., $[t_b, t_e]$, where t_b is a beginning time point and t_e is an ending time point, and $t_e > t_b$. In all cases, the first constraint is $(time(d^q), t_{max}]$, and the second constraint is $[t_{min}, time(d^q)]$, where $(time(d^q), t_{max}] = [time(d^q), t_{max}] - \{time(d^q)\}$, and t_{max} and t_{min} are the maximum time in the future and the minimum time in the past respectively. At retrieval time, the first constraint will be matched with the field FUTURE_DATE of predictions, whereas the second constraint will be matched with the field PUB_DATE of predictions.

3. FEATURES

In this section, we present features used for learning a ranking model for related news predictions. The model will be described in Section 4. We propose several classes of features to capture the similarity between a news article query q and a prediction p, i.e., term similarity, entity-based similarity, topic similarity, and temporal similarity. The detailed description of each class will be given next.

3.1 Term Similarity

Since a prediction is defined with multiple fields, we employ the fielded searching provided with Apache Lucene search engine. The first term similarity feature *retScore* is the default similarity scoring function[3] of Lucene, which is a variation of the tf-idf weighting scheme.

A disadvantage of *retScore* is that it will not retrieve any predictions that do not match the query terms. This issue is exacerbated in sentence retrieval by the fact that we have to retrieve short fragments of text which might refer to the query terms using anaphora or other linguistic phenomena. One technique to overcome this problem is to use query expansion/reformulation using synonyms or different words with very similar meanings. It has also been shown that

[3] http://lucene.apache.org/java/2_9_3/api/core/org/apache/lucene/search/Similarity.html

extending a sentence structure by its surrounding *context* sentences and weighting them using a field aware ranking function like bm25f consistently improves sentence retrieval [6]. Therefore, rather than reformulating a query, we will retrieve a prediction by looking at the CONTEXT and TITLE fields, in addition to the TEXT field. Thus, even if the TEXT field does not match exactly with a query term, p can receive a score if either the CONTEXT or TITLE field match the query term.

In our case, instead of weighting differently keyword matches in the title or body of a Web-page, we assign a different importance to matches in the sentence itself or its context. The second term similarity feature *bm25f* can be computed as follows.

$$\begin{aligned} bm25f(q,p,F) &= \sum_{w_i \in q} \frac{weight(w_i,p)}{k_1 + weight(w_i,p)} \cdot idf(w_i) \\ weight(w_i,p) &= \sum_{f \in F} \frac{freq(w_i,f) \cdot boost(f)}{(1-b_f) + b_f \cdot \frac{l_f}{avl_f}} \\ idf(w_i) &= \log \frac{N_P - n_{w_i} + 0.5}{n_{w_i} + 0.5} \end{aligned} \quad (1)$$

where l_f is the field length, avl_f is the average length for a field f, b_f is a constant related to the field length, k_1 is a free parameter and $boost(f)$ is the boost factor applied to a field f. N_P is the total number of predictions and n_{w_i} is the number of prediction containing w_i, and $F = \{\text{TEXT}, \text{CONTEXT}, \text{TITLE}\}$. We discuss parameter settings in Section 5.1.

3.2 Entity-based Similarity

This feature class is aimed at measuring the similarity between q and p by measuring the similarity of the entities they each contain. Note that, this class is only applicable for a query consisting of entities, that is, Q_E and Q_C, and it is ignored for Q_T. The first feature *entitySim* compares a string similarity between the entities of q and p_{entity} using the Jaccard coefficient, which can be computed as follows.

$$entitySim(q,p) = \frac{|q \cap p_{entity}|}{|q \cup p_{entity}|} \quad (2)$$

where p_{entity} is a set of entities, $|q \cap p_{entity}|$ and $|q \cup p_{entity}|$ are the size of intersection and union of entities of q and p.

Thus, the higher the overlap between the entities of a prediction and the query, the higher the prediction will be ranked for the query. We also want to rank predictions by using features that are commonly employed in an *entity ranking* task. For example, an entity is relevant if it appears in the title of a document, or it always occurs as a subject of sentence. We will employ *entity ranking* features by assuming that the more relevant entities a prediction contains, the more relevant it is. The entity-based features will be extracted and computed relative to the parent document of a prediction (d^p) or on the prediction itself (p).

Features extracted from documents are *title*, *titleSim*, *senPos*, *senLen*, *cntSenSubj*, *cntEvent*, *cntFuture*, *cntEventSubj*, *cntFutureSubj*, *timeDistEvent*, *timeDistFuture* and *tagSim*. Features extracted from predictions are *isSubj* and *timeDist*. The value of all features is normalized to range from 0 to 1, unless otherwise stated. First, the feature *title* indicates whether an entity e is in the title of d^p.

$$title(e,d^p) = isInTitle(e,d^p) \quad (3)$$

A value is 1 if e appears in the title of d^p, or 0 if otherwise. *titleSim* is a string similarity between e and the title.

$$titleSim(e,d^p) = \frac{|e \cap title(d^p)|}{|e \cup title(d^p)|} \quad (4)$$

senPos gives the position of the 1^{st} sentence where e occurs in d^p.

$$senPos(e,d^p) = \frac{len(d^p) - pos(firstSen(e))}{len(d^p)} \quad (5)$$

where $len(d^p)$ gives the length of d^p in words. $pos(s_y)$ is the position of a sentence s_y in d^p. *senLen* gives the length of the first sentence of d that contains e.

$$senLen(e,d^p) = \frac{len(firstSen(e))}{\max_{s_y \in d^p} len(s_y)} \quad (6)$$

cntSenSubj is the number of sentences where e is a subject. We run a dependency parser over the sentences in order to determine whether an entity is a subject of not.

$$cntSenSubj(e,d^p) = \frac{1}{|\mathcal{S}_e|} \sum_{s_y \in \mathcal{S}_e} isSubj(e,s_y) \quad (7)$$

where $\mathcal{S}_e$ is a set of all sentences of e in d^p. $isSubj(e,s_y)$ is 1 if e is a subject of s_y. *cntEvent* is the number of event sentences (or sentences annotated with dates) of e.

$$cntEvent(e,d^p) = \frac{1}{|\mathcal{E}_d^p|} \sum_{s_z \in \mathcal{E}_d^p} \sum_{s_y \in \mathcal{S}_e} isEqual(s_z,s_y) \quad (8)$$

where $\mathcal{E}_d^p$ is a set of all event sentences in d^p. $isEqual(s_z,s_y)$ returns 1 if s_z equals to s_y. *cntFuture* is the number of sentences with a mention of a future date. *cntEventSubj* is the number of event sentences that e is a subject.

$$cntEventSubj(e,d^p) = \frac{1}{|\mathcal{E}_d^p|} \sum_{s_z \in \mathcal{E}_d^p} isSubj(e,s_z) \quad (9)$$

Similarly, *cntFutureSubj* is the number of future sentences that e is a subject. *timeDistEvent* is a measure of the distance between e and all dates in d^p.

$$timeDistEvent(e,d^p) = \frac{1}{|\mathcal{E}_e|} \sum_{s_z \in \mathcal{E}_e} avg(normist(e,s_z)) \quad (10)$$

where $normDist(e,s_z) = \frac{1}{|\mathcal{T}_{s_z}|} \sum_{t_k \in \mathcal{T}_{s_z}} \frac{maxDist(s_z) - dist(e,t_k)}{maxDist(s_z)}$. $dist(w_i,w_j) = |pos(w_i) - pos(w_i)| - 1$. $\mathcal{E}_e$ is a set of all event sentences of e, and $\mathcal{T}_{s_z}$ is a set of all temporal expressions in s_z. $dist(w_i,w_j)$ is a distance in words between terms w_i and w_j. $maxDist(s_z)$ is a maximum distance between terms in s_z. $timeDistFuture(e,d^p)$ is a distance of e and all future dates in d^p computed similarly to *timeDistEvent*. *tagSim* is a string similarity between e and an entity tagged in d^p.

$$tagSim(e,d^p) = \max_{e_n \in \mathcal{N}_d^p} \frac{|e \cap e_n|}{|e \cup e_n|} \quad (11)$$

where $\mathcal{N}_d^p$ is a set of all entities tagged in d^p. *tagSim* is only applicable for a collection provided with manually assigned tags (e.g., the New York Times Annotated Corpus).

$isSubj(e,p)$ is 1 if e is a subject with respect to a prediction p, and $timeDist(e,p)$ is a distance of e and all future dates in p computed similarly to *timeDistEvent*. All features in this class are parameter-free.

3.3 Topic Similarity

This class of features is aimed to compare the similarity between q and p on a higher level by representing them using topics. Examples of topics are "health care reform", "financial crisis", and "global warming". Several works [7, 32] have proposed to model a document with a low dimensionality, or to use topics rather than terms. We will use latent Dirichlet allocation (LDA) [7] to model a set of topics. LDA is based on a generative probabilistic model that models documents as mixtures over an underlying set of topic distributions. In general, topic modeling consists of two main steps. The first step is to learn topic models from training data. LDA requires the parameter N_z or the number of topics to be specified. After a model is trained, the next step is to infer topics from the learned topic model outputting a topic distribution for the prediction.

Wei and Croft [32] incorporated topic modeling for ad-hoc retrieval, and showed that linearly combining LDA with the query likelihood model outperformed non-topic models like the unigram model. We incorporate LDA into the retrieval process differently from Wei and Croft in two ways. First, instead of combining LDA scores with the original retrieval score, we represent q and p as vectors of topic distributions and compute the topic-based similarity using a cosine similarity between two vectors. Second, we explicitly take the time dimension into modeling topics because topics distributions can evolve over time. Intuitively, topics keep changing over time according to different trends.

We apply topic modeling to future retrieval in three main steps: 1) learning a topic model, 2) inferring topic models, and 3) measuring topic similarity.

Learning a topic model. We take into account the time dimension for learning topic models. As shown in Figure 2, we create training data by partitioning the document collection D_N into sub-collections (or document snapshots) with respect to time. In other words, we group documents by year of publication, and randomly select documents as training data, called a training data snapshot D_{train,t_k} at time t_k. Note that, we can also use more sophisticated approaches for modeling topics over time as presented in [31]. However, we will leave this study for future work.

Topic model inference. Using learned models from the previous step, we determine the topics for q and p from their contents. This process is called *topic inference*, which represents a query and a prediction by a distribution of topics (probabilities). For example, given a topic model ϕ, a prediction p can be represented as $p_\phi = p(z_1), \ldots, p(z_n)$, where $p(z)$ gives a probability of a topic z obtained from ϕ. Because our topic models are learned from different time periods, a question is which model snapshot we use for inference. Note that, q and p must be inferred from the same model snapshot in order to be comparable. We select a topic model for inferring in two ways. First, we select a topic model from a time snapshot $time(d^q)$ which corresponds to the publication date of the news article parent of q. Second, a topic model is selected from a time snapshot t which corresponds to the publication date of the news article making prediction p, or the $time(d^p)$. Moreover, a prediction p will be inferred in three different ways depending on the contents used: 1) only text p_{txt}, 2) both text p_{txt} and context p_{ctx}, and 3) the parent document d^p. For a query q, the contents of its parent document d^q will be used for inference.

In addition to using all N_z topics for inference, we will also select only top-k topics ranked by the importance. The idea is that measuring the topic similarity using too many topics may not be as accurate as using only the most important topics. We use *coverage* and *variation* proposed in [29] for ranking topics. A topic coverage $\mu(z)$ assumes that topics that cover a significant portion of the corpus content are more important than those covering little content, while a topic variation $\sigma(z)$ considers topics that appear in all the documents to be too generic to be interesting, although they have significant content coverage. $\mu(z)$ and $\sigma(z)$ are computed using a mean and a standard deviation over topic distributions, and the final score for ranking topic is a multiply of $\mu(z)$ and $\sigma(z)$. The calculation $\mu(z)$ and $\sigma(z)$ for a topic z at time t_k is given as:

$$\mu(z) = \frac{1}{\sum_{i=1}^{N_D} len(d_i)} \sum_{i=1}^{N_D} len(d_i) \cdot p_i(z) \tag{12}$$

$$\sigma(z) = \sqrt{\frac{1}{\sum_{i=1}^{N_D} len(d_i)} \sum_{i=1}^{N_D} len(d_i) \cdot (p_i(z) - \mu(z))^2} \tag{13}$$

where N_D is the number of documents in a training set at time t_k, or $|D_{train,t_k}|$. $p_i(z)$ gives a probability of a topic z in a document d_i and $len(d_i)$ is the document length of d_i. A final score for ranking a topic z can be computed as:

$$rank(z) = \mu(z)^{\lambda_1} \cdot \sigma(z)^{\lambda_2} \tag{14}$$

where the parameters λ_1 and λ_2 indicate the importance of $\mu(z)$ and $\sigma(z)$. If $\lambda_1 = 1$ and $\lambda_2 = 0$, the ranking is determined purely by topic coverage. On the contrary, if $\lambda_1 = 0$ and $\lambda_2 = 1$, the ranking emphasizes topic variance.

Measuring topic similarity. Given a topic model ϕ, the topic similarity can be calculated using a cosine similarity between a topic distribution of query q_ϕ and a topic distribution of prediction p_ϕ as follows.

$$\begin{aligned} topicSim(q,p) &= \frac{q_\phi \cdot p_\phi}{||q_\phi|| \cdot ||p_\phi||} \\ &= \frac{\sum_{z \in Z} q_{\phi_z} \cdot p_{\phi_z}}{\sqrt{\sum_{z \in Z} q_{\phi_z}^2} \cdot \sqrt{\sum_{z \in Z} p_{\phi_z}^2}} \end{aligned} \tag{15}$$

We denote a topical feature using $LDA_{i,j,k}$, where i is one of the two different methods for selecting model snapshot: $i = 1$ for selecting a topic model from a time snapshot $time(d^q)$, and $i = 2$ for selecting from a time snapshot $time(d^p)$; j is one of the three different ways of using the contents for inference: p_{txt}, p_{ctx}, or d^p. Finally, k refers to whether we use *all* of only top-k of topics for inference. Thus, this results in 12 (=3*2*2) LDA-based features in total.

3.4 Temporal Similarity

As mentioned earlier, we explicitly exploit temporal expressions in ranking. To measure the temporal similarity between a query and a prediction, we employ two features proposed in previous work: *TSU* [16] and *FS* [15].

We will represent our model of time using a time interval $[b, e]$ having a begin point b and the end point e. The actual value of any time point, e.g., b or e in $[b, e]$, is an integer or the number of time units (e.g., milliseconds or days) passed (or to pass) a reference point of time (e.g., the UNIX epoch).

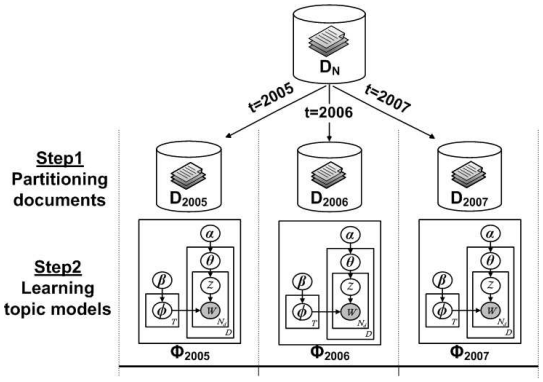

Figure 2: LDA topic snapshots based on time.

The first feature TSU is defined as the probability of generating the time of query q_{time} from the document creation date $time(d)$. TSU can be computed as follows.

$$TSU = DecayRate^{\lambda \cdot \frac{|q_{time} - time(d)|}{\mu}} \tag{16}$$

where $DecayRate$ and λ are constants, $0 < DecayRate < 1$ and $\lambda > 0$. μ is a unit of time distance. Intuitively, the probability obtained from this function decreases proportional to the distance between q_{time} and $time(d)$, that is, a document with its creation date closer to q_{time} will receive a higher probability than a document with its creation date farther from q_{time}.

We apply TSU for measuring the temporal similarity between q and p based on two assumptions. First, we assume that p is more likely to be relevant if its parent time $time(d^p)$ is closer to the time of query article $time(d^q)$. Our first temporal feature, denoted TSU_1, will be calculated similarly to Equation 16 resulting the following function.

$$TSU_1(q,p) = DecayRate^{\lambda \cdot \frac{|time(d^q) - time(d^p)|}{\mu}} \tag{17}$$

The second assumption, denoted TSU_2, is that a prediction is more likely to be relevant if its future dates p_{future} are closer to the publication date of query article $time(d^q)$. If there are more than one future dates associated to p, a final score will be averaged over scores of all future dates p_{future}. The temporal distance of TSU_2 of q and p is defined as follows.

$$TSU_2(q,p) = \frac{1}{N_f} \sum_{t_f \in p_{future}} DecayRate^{\lambda \cdot \frac{|time(d^q) - t_f|}{\mu}} \tag{18}$$

where t_f is a future date in p_{future} and N_f is the number of all future dates.

In addition to TSU_1 and TSU_2, we can measure the temporal similarity between q and p using a fuzzy membership function, which is originally proposed by Kalczynski and Chou [15].

We adapt the original fuzzy set function in [15] by using its parent time $time(d^p)$ and the time of query article $time(d^q)$. We denote this feature as FS_1, and it can be computed as follows.

$$FS_1(q,p) = \begin{cases} 0 & \text{if } time(d^p) < \alpha_1 \vee time(d^p) > time(d^q), \\ f_1(time(d^p)) & \text{if } time(d^p) \geq \alpha_1 \wedge time(d^p) < time(d^q), \\ 1 & \text{if } time(d^p) = time(d^q). \end{cases} \tag{19}$$

where $f_1(time(d^p))$ is equal to $\left(\frac{time(d^p)-\alpha_1}{time(d^q)-\alpha_1}\right)^n$ if $time(d^p) \neq time(d^q)$, or 1 if $time(d^p) = time(d^q)$.

We define the second temporal feature based on a fuzzy set by using the prediction's future dates p_{future} and the publication date of query article $time(d^q)$. Similarly, if a prediction p has more than one future date, a final score will be averaged over scores of all dates p_{future}. The second temporal feature FS_2 is defined as follows.

$$FS_2(q,p) = \frac{1}{N_f} \sum_{t_f \in p_{future}} \begin{cases} 0 & \text{if } t_f < time(d^q) \vee t_f > \alpha_2, \\ 1 & \text{if } t_f = time(d^q), \\ f_2(t_f) & \text{if } t_f > time(d^q) \wedge t_f \leq \alpha_2. \end{cases} \tag{20}$$

N_f is the number of all future dates in p_{future}, and t_f is a future date, i.e., $t_f \in p_{future}$. $f_2(t_f)$ is equal to $\left(\frac{\alpha_2 - t_f}{\alpha_2 - time(d^q)}\right)^m$ if $t_f \neq time(d^q)$, or 1 if $t_f = time(d^q)$. n and m are constants. α_1 and α_2 are the minimum and maximum time of reference with respect to q_{time}. α_1 is calculated by subtracting the time offset s_{min} from from q_{time} , and α_2 is calculated by adding the offset s_{max} to q_{time}.

4. RANKING MODEL

Given a query q, we will rank a prediction p using a ranking model obtained by training over a set of labeled query/prediction pairs using a learning algorithm. An unseen query/prediction pair (q,p) will be ranked according to a weighted sum of feature scores:

$$score(q,p) = \sum_{i=1}^{N} w_i \times x_i \tag{21}$$

where x_i are the different features extracted from p and q, N is the number of features, and w_i are the weighting coefficients. The goal of the algorithm is to learn the weights w_i using a training set of queries and predictions, in order to minimize a given loss function. Learning to rank algorithms can be categorized into three approaches: pointwise, pairwise, and listwise approaches [20]. The pointwise approach assumes that retrieved documents are independent, so it predicts a relevance judgment for each document and ignores the positions of documents in a ranked list. The pairwise approach considers a pair of documents, and relevance prediction is given as the relative order between them (i.e., pairwise preference). The listwise approach considers a whole set of retrieved documents, and predicts the relevance degrees among documents. For a more detailed description of each approach, please refer to [20].

We employ the listwise learning algorithm SVM^{MAP} [34]. The algorithm trains a classifier using support vector machines (SVM), and it determines the order of retrieved documents in order to directly optimize Mean Average Precision (MAP). In addition, we also experimented with other learned ranking algorithms: RankSVM [14], SGD-SVM [36], PegasosSVM [28], and PA-Perceptron [9]. However, these algorithms do not perform as well as SVM^{MAP} in our experiments. Thus, we will only discuss the results obtained from SVM^{MAP} in the next section.

Table 2: Examples of future-related topics.

Politics	Environment	Space
president election	global warming	Mars
Iraq war	energy efficiency	Moon
Science	**Physics**	**Health**
earthquake	particle Physics	bird flue
tsunami	Big Bang	influenza
Business	**Sport**	**Technology**
subprime	Olympics	Internet
financial crisis	World cup	search engine

5. EXPERIMENTS

In this section, we evaluate the retrieval effectiveness of our proposed ranking model using three different query formats. We will first describe the experimental settings followed by an explanation of the results and a detailed discussion.

5.1 Experimental Settings

Temporal document collection. We used the New York Times Annotated Corpus for our document collection, which contains 1.8 million documents covering the period from January 1987 to June 2007. In order to extract predictions and features, a series of language processing tools, including OpenNLP (for tokenization, sentence splitting and part-of-speech tagging, and shallow parsing), the SuperSense tagger (for named entity recognition) and TARSQI Toolkit (for extracting temporal expressions from documents). Given the importance of time to our system, we note that the temporal expression extraction of TARSQI has a reported performance of 0.81 F1 on the Time Expression Recognition and Normalization task[4].

We employed the Apache Lucene search engine for both indexing and retrieving predictions. The statistics of extracted data are as follows. There are 44,335,519 sentences and 548,491 are predictions. There are 939,455 future dates, and an average future date per prediction is 1.7 and the standard deviation is 0.92. Among 1.8 million documents, more than 25% of all documents contain at least one prediction (i.e., a reference to the future). In order to determine this percentage over a broader range of news sources, we performed the same analysis on 2.5 million documents from over 100 news sources from Yahoo! News for the one year period from July 2009 to July 2010 and found over 32% of the documents contained at least one prediction.

Future-related queries. There is no gold standard available to evaluate the task of ranking related news prediction. We manually selected 42 query news articles from the New York Times that cover the future-related topics shown in Table 2. The actual queries (Q_E, Q_T and Q_C) used for retrieving predictions are extracted from these news articles.

Relevance assessments. Human assessors were asked to evaluate query/prediction pairs (e.g., relevant or non-relevant) using 5 levels of relevance: 4 for *excellent* (very relevant prediction), 3 for *good* (relevant prediction), 2 for *fair* (related prediction), 1 for *bad* (non-relevant prediction), and 0 for *non prediction* (incorrect tagged date). The last option was presented because there are predictions incorrectly annotated with time (this is an error produced by the annotation tools). More precisely, an assessor was asked to give a relevance score $Grade(q, p, t)$ where (q, p, t) is a triple of a query q, a prediction p, and a future date t in p. Consider the following prediction about the topic "global warming" and the publication date of the news article is 2007/02/21:

> *Formal ratification of the pact – which commits the union to reduce emissions of "greenhouse gases" by 8 percent of 1990 levels during the five-year period from* **2008** *through* **2012** *– now goes to the European Council of heads of state and government, which could act as early as this month at the union summit in Barcelona.*

The prediction contains two future dates (as highlighted in bold). Hence, an assessor has to give judges to two triples corresponding to q, p and *both* future dates. A triple (q, p, t) is considered **relevant** if $Grade(q, p, t) \geq 3$, and it is considered non-relevant if $1 \leq Grade(q, p, t) \leq 2$. Relevance level 0 is not included in the evaluation[5]. These judgments are normalized by a query/prediction pair (q, p) since we are interested in presenting a prediction for all future dates, regardless of their number. That is, a query/prediction pair (q, p) is relevant if and only if there is at least one relevant triple (q, p, t), and a prediction is **non-relevant** if all triples are non-relevant. Our assumption is that predictions extracted from more recent documents are more relevant.

In total, assessors judged 52 queries and for each one of them we retrieved up to 100 sentences that contained predictions. On average 94 sentences with future mentions were retrieved, with an average of 1.2 future dates per prediction. Finally, assessors evaluated 4,888 query/prediction pairs (approximately 6,032 of triples)[6].

Our machine learning ranking models operate in a supervised manner, and as such, they need training data for learning. We created training data using cross validation by randomly partitioned query articles into N_F folds. We used $N_F - 1$ query/prediction from other folds for training a ranking mode and the remaining fold for testing. We removed queries with zero relevant results, and we obtained $N_F = 3, 4, 5$ for Q_E, Q_C, Q_T respectively.

Parameter setting. We set the boost factors on an independent experiment as $boost(\text{TEXT}) = 5.0$, $boost(\text{CONTEXT}) = 1.0$, and $boost(\text{TITLE}) = 2.0$. We use the recommended values for the constants $b = 0.75$ for all fields, and $k_1 = 1.2$ [26]. For LDA-based features, we trained a yearly model snapshot by selecting 4% of all documents in each year. For each document, we filtered out terms occurring in less than 15 documents and the 100 most common terms. We learn a topic model for each document snapshot by employing Stanford Topic Modeling Toolbox[7], and the number of topics for training LDA N_z is fixed to 500 and the number of topics for inference k is 200. A learning algorithm we use is the collapsed variational Bayes approximation to the LDA objective (CVB0LDA) [2]. All other parameters are default values of the topic modeling. Using CVB0LDA

[4] http://timex2.mitre.org/tern.html

[5] We are interested in assessing the performance of the ranking algorithm and not the annotation tools. However, we note the overall system will be impacted by the annotation errors.

[6] The evaluation collection we have created is available for download at http://www.idi.ntnu.no/~nattiya/data/sigir2011/futurepredictions.zip.

[7] http://nlp.stanford.edu/software/tmt/tmt-0.3/

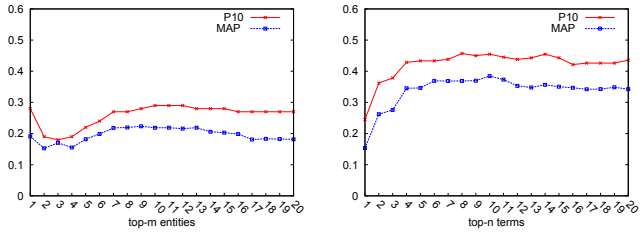

Figure 3: P@10 and MAP performance of Q_E (left) when varying top-m entities, and Q_C (right) when varying top-n terms.

Table 3: The effectiveness of each method when using all queries; †,∓ indicates statistical improvement over the corresponding simple methods using t-test with significant at $p < 0.1$,$p < 0.05$ respectively.

Method	P@1	P@3	P@10	MRR	MAP
Q_E	0.300	0.333	0.290	0.473	0.219
Q_T	0.643	0.579	0.455	0.760	0.385
Q_C	0.500	0.561	0.427	0.656	0.231
Re-Q_E	0.500	0.499	0.360	0.629	0.266
Re-Q_T	0.738†	0.619	0.462	0.831†	0.387
Re-Q_C	0.773∓	0.682∓	0.455	0.841∓	0.271

required high CPU and memory, but needed fewer iterations and had faster convergence rates than a collapsed Gibbs sampler [12], which requires less memory during training.

For both TSU_1 and TSU_2, $DecayRate = 0.5$, $\lambda = 0.5$ and $\mu = 2y$ are used where y the number of years. For both FS_1 and FS_2, $n = 2$, $m = 2$, $s_{min} = 4y$ and $s_{max} = 2y$ are used. So, $\alpha_1 = time(d^q) - 4y$ and $\alpha_2 = time(d^q) + 2y$.

Methods for comparison. We experiment with the three different ways of constructing the query Q_E, Q_T, and Q_C. The baseline for retrieval is Lucene's default ranking function and our queries incorporate two time constraints as explained in Section 2.4. We re-rank the baseline results using SVM^{MAP} yielding *Re-*Q_E, *Re-*Q_T and *Re-*Q_C. For the application of ranking related news predictions, we prefer top-precision retrieval performance metrics over recall-based metrics: a user will be typically interested in a few top predictions even though there are many predictions retrieved. Consequently, we envision a user interface that contains little space for displaying related predictions. Thus, we will measure the retrieval effectiveness by the precision at 1, 3 and 10 (P@1, P@3, and P@10 respectively), Mean Reciprocal Rank (MRR), and Mean Average Precision (MAP). We report the average performance over N_F folds to measure the overall performance, for each query type.

5.2 Experimental Results

The three types of queries (Q_E,Q_T, and Q_C) are composed of either *top-m* entities or *top-n* terms, or both. We first establish which are good m and n values for each one of the types. Instead of varying m and n in re-ranking, we select the m and n that give a reasonable improvement in a hold-out set (where we randomly divided queries into two folds). Therefore, we will use only one fixed version of m and n for the rest of our experiments. We select the values of m and n by performing a preliminary analysis as follows. First, by looking at P@10 and MAP, we select the value of m that yields the best performance using only Q_E to retrieve predictions for each varying m. As shown in Figure 3 (left), $9 \leq m \leq 12$ give almost no difference in terms of P@10. In spite of that, we choose the number of entities $m = 11$ because it is slightly better than the other values. Next, we find the optimal value of n by observing the performance of Q_C when m is fixed to 11 and the value of n is varied. As depicted in Figure 3(right), there is very slight difference in P@10 for $9 \leq n \leq 11$; We choose the number of terms $n = 10$ because it obtains the best in MAP among them.

The retrieval effectiveness of simple methods and their corresponding re-ranking methods are displayed in Table 3. These results are averaged over queries retrieving at least one relevant prediction. In general, Q_T gains the highest effectiveness in all measurements followed by Q_C and Q_E and the feature-based re-ranking approach improves the effectiveness for all query types. In addition, *Re-*Q_C has the highest effectiveness over other re-ranking methods for P@1 and P@3, while *Re-*Q_T gains the highest effectiveness for the rest of all metrics.

Q_E and Q_C pose a problem in not retrieving any relevant result of our judged pool among the first 100 for a large number of queries, which makes it impossible for the machine learning model to improve the ranking. However, we still want to compare the performance between the different variations of the query (Q_E, Q_C, Q_T). Therefore, we use a subset of queries that contained *at least* one relevant result among all the different methods. The results are shown in Table 4 where we compare all other methods against Q_E because we have observed that Q_E performs worst among them. As seen from the results of each re-ranking method, our proposed features improve the effectiveness for all corresponding simple methods. In particular, the re-ranking method *Re-*Q_C outperforms the simple method Q_E significantly. However, *Re-*Q_E did not provide a significant improvement over Q_E. The results show that, for the same set of queries, using entities alone are limited while terms alone are able to retrieve most of relevant predictions.

Interestingly, when looking at the same sub-set of queries with relevant predictions, the re-ranking approach *Re-*Q_C outperforms every other method, even if the plain retrieval Q_T is superior to Q_C. This is an indicator that entity-based features are able to produce higher quality results but only for a certain type of topics. We performed an error analysis to determine why Q_E is unable to retrieve relevant predictions. In general, Q_E fails for a topic that cannot be represented using only people, locations, or organizations. For example, for the topic about "the Europeans agreement of gas emissions", the top-5 Q_E is
$\langle$*European Union*, *Brussels*, *Finland*, *Germany*, *Hungary*$\rangle$ and the top-5 Q_T is $\langle$*european*, *emission*, *target*, *climate*, *brussels*$\rangle$. In this case, Q_E is unable to represent the *key* terms "emission" and "climate", and thus fails to retrieve many relevant predictions that match those terms.

Similarly, for the query topic about "Clinton health care reform", Q_E is represented using the named entity *Clinton* (the terms "health care" and "reform" are not annotated as entities). When matching, all predictions containing the entity *Clinton* are matched which will return many documents that are not related to "health care" and "reform".

Table 4: The effectiveness of each method when using a subset of queries; †,∓,⋆ indicates statistical improvement over the method Q_E using t-test with significant at $p < 0.1$, $p < 0.05$, $p < 0.01$ respectively.

Method	P@1	P@3	P@10	MRR	MAP
Q_E	0.300	0.333	0.290	0.473	0.219
Q_T	0.500	0.533	0.430	0.638	0.219
Q_C	0.600†	0.533†	0.360	0.727†	0.163
Re-Q_E	0.500	0.499	0.360	0.629	0.266
Re-Q_T	0.700	0.600	0.410	0.762	0.236
Re-Q_C	1.000∓	0.714∓	0.443	1.000	0.303⋆

Table 5: Top-5 features with highest weights and lowest weights for each query type.

Q_E		Q_T		Q_C	
Feature	W_i	Feature	W_i	Feature	W_i
tagSim	1.00	*bm25f*	1.00	$LDA_{1,parent,k}$	1.00
FS_1	0.97	*retScore*	0.60	*retScore*	0.99
TSU_2	0.88	$LDA_{1,parent,k}$	0.55	$LDA_{1,parent,all}$	0.96
$LDA_{1,txt,k}$	0.87	$LDA_{2,parent,k}$	0.51	*bm25f*	0.93
$LDA_{1,txt,all}$	0.82	$LDA_{1,parent,all}$	0.49	*isSubj*	0.87
cntSenSubj	0.01	*timeDistEvent*	-0.03	*cntEventSen*	-0.02
cntEventSubj	0.01	*timeDistFuture*	-0.11	*querySim*	-0.05
isInTitle	0.00	*cntEventSen*	-0.12	*cntFutureSen*	-0.10
cntEventSen	0.00	*cntFutureSen*	-0.12	*timeDistFuture*	-0.14
querySim	-0.01	*senLen*	-0.16	*senLen*	-0.18

5.3 Feature Analysis

We analyzed feature weights obtained from the learning algorithm SVM^{MAP} in order to understand better what is the importance of the different features,. Note that, in order to compare the weights among different queries, we performed normalization by diving with the maximum value of all weights for each query. Column w_i in Table 5 displays the top-5 features with highest and lowest weights for each query type.

At least two topic-based features of all query types are in the top-5 features with highest weight, and therefore topic-based features play an important role in the re-ranking model. Although *retScore* and *bm25f* measure the similarity on a term level, they help to re-rank predictions when incorporated into the machine learning model. as seen in the top-5 features for Q_T and Q_C. The feature that received the highest importance value for the Q_E type is *tagSim*, which measures the similarity between entities in a prediction and manually tagged entities. This indicates that tagged entities in a query document can precisely represent user information needs. The temporal features FS_1 and FS_1 also play an important role for Q_E.

Features in top-5 features with lowest weights are those from the entity-based class. Recall that these features are extracted in order to measure the importance of entities annotated in a prediction with respect to their respective parent documents. However, the results show that these features are not good enough for discriminating between relevant and non-relevant predictions.

6. RELATED WORK

Our related work includes sentence retrieval, entity ranking, temporal ranking, and domain-specific predictions.

Sentence retrieval is the task of retrieving a relevant sentence related to a query. Different application areas of sentence retrieval are mentioned in the book of Murdock [24] and references therein, including, for example, question answering [30], text summarization, and novelty detection. Surdeanu et al. [30] applied supervised learning to rank a set of short answers (sentences) matched a given question (query) by using different classes features. Li and Croft [19] proposed to detect novelty topics by analyzing sentence-level information (sentence lengths, named entities, and opinion patterns). Generally, because sentences are much smaller than documents and thus have limited content compared to documents, the effectiveness of the retrieval of sentences is significantly worse. To address this problem, Blanco and Zaragoza [6] proposed to use the context of sentences in order to improve the effectiveness of sentence retrieval.

There have been a number shared tasks with the goal of furthering research in the area of entity ranking. For instance, the TREC 2008 Enterprise track was created with the objective to find experts (or people) related to a given topic of interest. The INEX Entity Ranking track [10] was launched with the task of finding a list of relevant entities (represented by Wikipedia articles) for a given topic. Recently, the TREC 2009 Entity track was introduced, and the task is to find related entities (represented by homepages) given a topic (called a source entity). The difference between the TREC 2009 Entity and the previous tracks is that it allows a relation and a target entity type to be explicitly specified. There are various approaches to ranking entities by using language models [4], voting models [21], and entity-based graph models [35].

Many ranking models exploiting temporal information have been proposed, including [5, 11, 18, 23]. Li and Croft [18] experimented with time-based language models by assigning a document prior using an exponential decay function of its creation date, such that the more recent documents obtain the higher probabilities of relevance. Diaz and Jones [11] build a temporal profile of a query from the distribution of document publication dates. They use time dependent features derived from these profiles that improve the ranking of temporal queries.

Berberich et al. [5] integrated temporal expressions into query-likelihood language modeling, which considers uncertainty inherent to temporal expressions in a query and in documents, i.e., two temporal expressions can refer to the same time interval even when they are not exactly equal. Metzler et al. [23] mined query logs to identify implicit temporal information needs and presented a time-dependent ranking model for certain types of queries.

There is much research in domain-specific predictions such as stock market predictions [27, 33] and recommender systems [17, 25]. The first aims at predicting stock price movements by analyzing financial news, while the latter applies collaborative filtering algorithms for recommending books, videos, movie, etc. based on users' interests.

The future retrieval problem was first presented by Baeza-Yates [3]. He proposed to extract temporal expressions from news, index news articles together with temporal expressions, and retrieve future information (composed of text and future dates) by using a probabilistic model. A document score is given as a multiplication of a *keyword* similarity and a time confidence, i.e., a probability that the document's events will actually happen. The limitation of this original work is that it is evaluated using a small data set and only a year granularity is used.

The more recent work on the future-related information retrieval is presented by Jatowt et al. [13]. In contrast to our work, they do not focus on relevance and ranking future-related information retrieval. They presented an analytical tool for extracting, summarizing and aggregating future-related events from news archives, but did not perform an extensive evaluation, only calculating averaged precision on a small set of generated results.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we demonstrated that future related information is abundant in news stories and defined the task of *ranking related future predictions*. The main goal of this task is to improve user access to this information by selecting the predictions from a news archive that are most relevant to a given news article. We created an evaluation dataset with over 6000 relevance judgments and addressed this task using a learning to rank methodology incorporating four classes of features including term similarity, entity-based similarity, topic similarity, and temporal similarity that outperforms a strong baseline system. Finally, we performed an in-depth analysis of feature importance. Possible future work includes time-dependent query classification using query logs, combining multiple sources (Wikipedia, blogs, home pages, and tweets) of future-related information, sentimental analysis for future-related information and evaluating the effectiveness of the predictions in a real-world news application.

8. ACKNOWLEDGMENTS

We would like to thank Hugo Zaragoza for his help at the early stages of this paper. This work is partially supported by the EU Large Scale Integrated Project LivingKnowledge (contract no. 231126).

9. REFERENCES

[1] O. Alonso, M. Gertz, and R. Baeza-Yates. On the value of temporal information in information retrieval. *ACM SIGIR Forum*, 41(2):35–41, 2007.

[2] A. Asuncion, M. Welling, P. Smyth, and Y. W. Teh. On smoothing and inference for topic models. In *Proceedings of UAI'2009*, 2009.

[3] R. Baeza-Yates. Searching the future. In *Proceedings of ACM SIGIR workshop MF/IR 2005*, 2005.

[4] K. Balog, L. Azzopardi, and M. de Rijke. A language modeling framework for expert finding. *Inf. Process. Manage.*, 45(1):1–19, 2009.

[5] K. Berberich, S. Bedathur, O. Alonso, and G. Weikum. A language modeling approach for temporal information needs. In *Proceedings of ECIR'2010*, 2010.

[6] R. Blanco and H. Zaragoza. Finding support sentences for entities. In *Proceeding of SIGIR'2010*, 2010.

[7] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[8] J. Canton. *The Extreme Future: The Top Trends That Will Reshape the World in the Next 20 Years*. Plume, 2007.

[9] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *J. Mach. Learn. Res.*, 7:551–585, 2006.

[10] G. Demartini, A. P. Vries, T. Iofciu, and J. Zhu. *Overview of the INEX 2008 Entity Ranking Track*. 2009.

[11] F. Diaz and R. Jones. Using temporal profiles of queries for precision prediction. In *Proceedings of SIGIR'2004*, 2004.

[12] T. L. Griffiths. Finding scientific topics. *Proceedings of the National Academy of Science*, 101:5228–5235, Jan. 2004.

[13] A. Jatowt, K. Kanazawa, S. Oyama, and K. Tanaka. Supporting analysis of future-related information in news archives and the web. In *Proceedings of JCDL'2009*, 2009.

[14] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of KDD'2002*, 2002.

[15] P. J. Kalczynski and A. Chou. Temporal document retrieval model for business news archives. *Inf. Process. Manage.*, 41, 2005.

[16] N. Kanhabua and K. Nørvåg. Determining time of queries for re-ranking search results. In *Proceedings of ECDL'2010*, 2010.

[17] N. Lathia, S. Hailes, L. Capra, and X. Amatriain. Temporal diversity in recommender systems. In *Proceeding of SIGIR'2010*, 2010.

[18] X. Li and W. B. Croft. Time-based language models. In *Proceedings of CIKM'2003*, 2003.

[19] X. Li and W. B. Croft. Improving novelty detection for general topics using sentence level information patterns. In *Proceedings of CIKM'2006*, 2006.

[20] T.-Y. Liu. Learning to rank for information retrieval. *Found. Trends Inf. Retr.*, 3(3):225–331, 2009.

[21] C. Macdonald and I. Ounis. Searching for expertise: Experiments with the voting model. *Comput. J.*, 52(7):729–748, 2009.

[22] M. Matthews, P. Tolchinsky, R. Blanco, J. Atserias, P. Mika, and H. Zaragoza. Searching through time in the new york times. In *Bridging Human-Computer Interaction and Information Retrieval*, 2010.

[23] D. Metzler, R. Jones, F. Peng, and R. Zhang. Improving search relevance for implicitly temporal queries. In *Proceedings of SIGIR'2009*, 2009.

[24] V. Murdock. *Exploring Sentence Retrieval*. VDM Verlag Dr. Mueller e.K., 2008.

[25] M. J. Pazzani and D. Billsus. The adaptive web. pages 325–341, 2007.

[26] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of SIGIR'1994*, 1994.

[27] R. P. Schumaker and H. Chen. Textual analysis of stock market prediction using breaking financial news: The azfin text system. *ACM Trans. Inf. Syst.*, 27:12:1–12:19, March 2009.

[28] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of ICML'2007*, 2007.

[29] Y. Song, S. Pan, S. Liu, M. X. Zhou, and W. Qian. Topic and keyword re-ranking for lda-based topic modeling. In *Proceeding of CIKM'2009*, 2009.

[30] M. Surdeanu, M. Ciaramita, and H. Zaragoza. Learning to rank answers on large online qa collections. In *Proceedings of ACL-08: HLT*, 2008.

[31] X. Wang and A. McCallum. Topics over time: a non-markov continuous-time model of topical trends. In *Proceedings of KDD'2006*, 2006.

[32] X. Wei and W. B. Croft. Lda-based document models for ad-hoc retrieval. In *Proceedings of SIGIR'2006*, 2006.

[33] D. Wu, G. P. C. Fung, J. X. Yu, and Q. Pan. Stock prediction: an event-driven approach based on bursty keywords. *Frontiers of Computer Science in China*, 3(2):145–157, 2009.

[34] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *Proceedings of SIGIR'2007*, 2007.

[35] H. Zaragoza, H. Rode, P. Mika, J. Atserias, M. Ciaramita, and G. Attardi. Ranking very many typed entities on wikipedia. In *Proceedings of CIKM'2007*, 2007.

[36] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of ICML'2004*, 2004.