Learning Recurrent Event Queries for Web Search

Ruiqiang Zhang and **Yuki Konda** and **Anlei Dong**
Pranam Kolari and **Yi Chang** and **ZhaohuiZheng**
Yahoo! Inc
701 First Avenue, Sunnyvale, CA94089

Abstract

Recurrent event queries (REQ) constitute a special class of search queries occurring at regular, predictable time intervals. The freshness of documents ranked for such queries is generally of critical importance. REQ forms a significant volume, as much as 6% of query traffic received by search engines. In this work, we develop an improved REQ classifier that could provide significant improvements in addressing this problem. We analyze REQ queries, and develop novel features from multiple sources, and evaluate them using machine learning techniques. From historical query logs, we develop features utilizing query frequency, click information, and user intent dynamics within a search session. We also develop temporal features by time series analysis from query frequency. Other generated features include word matching with recurrent event seed words and time sensitivity of search result set. We use Naive Bayes, SVM and decision tree based logistic regression model to train REQ classifier. The results on test data show that our models outperformed baseline approach significantly. Experiments on a commercial Web search engine also show significant gains in overall relevance, and thus overall user experience.

1 Introduction

REQ pertains to queries about events which occur at regular, predictable time intervals, most often weekly, monthly, annually, bi-annually, etc. Naturally, users issue REQ periodically. REQ usually refer to:

Organized public events such as festivals, conferences, expos, sports competitions, elections: winter olympics, boston marathon, the International Ocean Research Conference, oscar night.

Public holidays and other noteworthy dates: labor day, date of Good Friday, Thanksgiving, black friday.

Products with annual model releases, such as car models: ford explorer, prius.

Lottery drawings: California lotto results.

TV shows and programs which are currently running: American idol, Inside Edition.

Cultural related activities: presidential election, tax return, 1040 form.

Our interest in studying REQ arises from the challenge imposed on Web search ranking. To illustrate this, we show an example in Fig. 1 that snapshots the real ranking results of the query, *EMNLP*, issued in 2010 when the authors composed this paper, on Google search engine. It is obvious the ranking is not satisfactory because the page about *EMNLP2008* is on the first position in 2010. Ideally, the page about *EMNLP2010* on the 6th position should be on the first position even if users don't explicitly issue the query, *EMNLP 2010*, because *EMNLP* is a REQ. The query, "EMNLP", implicitly, without a year qualifier, needs to be served the most recent pages about "EMNLP".

A better search ranking result cannot be achieved if we do not categorize "EMNLP" as a REQ, and provide special ranking treatment to such queries. Existing search engines adopt a fairly involved ranking algorithm to order Web search results by considering many factors. Time is an important factor but not the most critical. The page's ranking score mostly depends on other features such as tf-idf (Salton and McGill, 1983), BM25 (Jones

Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, pages 1129–1139,
MIT, Massachusetts, USA, 9-11 October 2010.

Figure 1: A real problematic ranking result by Google for a REQ query, "EMNLP". The EMNLP2010 page should be on the 1st position.

et al., 2000), anchor text, historical clicks, pagerank (Brin and Page, 1998), and overall page quality. New pages about EMNLP2010 obtain less favorable feature values than the pages of 2009 earlier in terms of anchor text, click or pagerank because they have existed for a shorter time and haven't accumulated sufficient popularity to make them stand out. Without special treatment, the new pages about "EMNLP2010" will typically not be ranked appropriately for the users.

Typically, a recurrent event is associated with a root, and spawns a large set of queries. Oscar, for instance, is a recurrent event about the annual Academy Award. Based on this, queries like "oscar best actress", "oscar best dress", "oscar best movie award", are all recurrent event queries. As such, REQ is a highly frequent category of query in Web search. By Web search query log analysis, we observe that there about 5-6% queries of total query volume belongs to this category.

In this work, we learn if a query is in the REQ class, by effectively combining multiple features. Our features are developed through analysis of historical query logs. We discuss our approaches in detail in Section 3. We then develop a REQ classifier where all the features are integrated by machine learning models. We use Naive Bayes, SVM and decision tree based logistic regression models. These models are described in Section 4. Our experiments for REQ classifier and Web search ranking are detailed in Section 5 and 6.

2 Related Work

We found our work were related to two other problems: general query classification and time-sensitive query classification. For general query classification, the task is to assign a Web search query to one or more predefined categories based on its topics. In the query classification contest in KDD-CUP 2005 (Li et al., 2005), seven categories and 67 sub-categories were defined. The winning solution (Shen et al., 2005) used multiple classifiers integrated by ensemble method. The difficulties for query classification are from short queries, lack of labeled data, and query sense ambiguity. Most popular studies use query log, web search results, unlabeled data to enrich query classification (Shen et al., 2006; Beitzel et al., 2005), or use document classification to predict query classification (Broder et al.,). General query classification is also studied for query intent detection by (Li et al., 2008).

There are many prior works to study the time sensitivity issue in web search. For example, Baeza-Yates *et al.* (Baeza-Yates et al., 2002) studied the relation between the web dynamics, structure and page quality, and demonstrated that PageRank is biased against new pages. In T-Rank Light and T-Rank algorithms (Berberich et al., 2005), both activity (i.e., update rates) and freshness (i.e., timestamps of most recent updates) of pages and links are taken into account for link analysis. Cho *et al.* (Cho et al., 2005) proposed a page quality ranking function in order to alleviate the problem of popularity-based ranking, and they used the derivatives of PageRank to forecast future PageRank values for new pages. Pandey *et al.* (Pandey et al., 2005) studied the tradeoff between new page exploration and high-quality page exploitation, which is based on a ranking method to randomly promote some new pages so that they can accumulate links quickly.

More recently, Dong *et al.* (Dong et al., 2010a)

proposed a machine-learned framework to improve ranking result freshness, in which novel features, modeling algorithms and editorial guideline are used to deal with time sensitivities of queries and documents. In another work (Dong et al., 2010b), they use micro-blogging data (e.g., Twitter data) to detect fresh URLs. Novel and effective features are also extracted for fresh URLs so that ranking recency in web search is improved.

Perhaps the most related work to this paper is the query classification approach used in (Zhang et al., 2009) and (Metzler et al., 2009), in which year qualified queries (YQQs) are detected based on heuristic rules. For example, a query containing a year stamp is an explicit YQQ; if the year stamp is removed from this YQQ, the remaining part of this query is also a YQQ, which is called implicit YQQ. Different ranking approaches were used in (Zhang et al., 2009) and (Metzler et al., 2009) where (Zhang et al., 2009) boosted pages of the most latest year while (Metzler et al., 2009) promoted pages of the most influential years. Similarly, Nunes *et al.* (Nunes, 2007) applied information extraction techniques to identify temporal expression in web search queries, and found 1.5% of queries containing temporal expression.

Dong *et al.* (Dong et al., 2010a) proposed a breaking-news query classifier with high accuracy and reasonable coverage, which works not by modeling each individual topic and tracking it over time, but by modeling each discrete time slot, and comparing the models representing different time slots. The buzziness of a query is computed as the language model likelihood difference between different time slots. In this approach, both query log and news contents are exploited to compute language model likelihood.

Diaz (Diaz, 2009) determined the newsworthiness of a query by predicting the probability of a user clicks on the news display of a query. In this framework, the data sources of both query log and news corpus are leveraged to compute contextual features. Furthermore, the online click feedback also plays a critical role for future click prediction.

Konig *et al.* (Knig et al., 2009) estimated the click-through rate for dedicated news search result with a supervised model, which is to satisfy the requirement of adapting quickly to emerging news event. Some additional corpora such as blog crawl and Wikipedia is used for buzziness inference. Compared with (Diaz, 2009), different feature and learning algorithms are used.

Elsas *et al.* (Elsas and Dumais, 2010) studied improving relevance ranking by detecting document content change to leverage temporal information.

3 Feature Generation

To better understand our work, we first introduce three terms. We subdivide all raw queries in query log into three categories: Explicit Timestamp, Implicit Timestamp, and No Timestamp. An Explicit Timestamp query contains at least one token being a time indicator. For example, *emnlp 2010*, *2007 December holiday calendar*, *amsterdam weather summer 2009*, *Google Q1 reports 2010*. These queries are considered to conatin time indicators, because we can regard {*2010, 2007, 2009*} as year indicator, *december* as *month* indicator, {*summer, Q1(first quarter)*} as *seasonal* indicator. To simplify our work, we only consider the *year* indicators, *2010, 2007, 2009*. Such *year* indicators are also the most important and most popular indicators, as noted in (Zhang et al., 2009). Any query containing at least one *year* indicator is an Explicit Timestamp query. Due to word sense ambiguity, some queries labeled as Explicit Timestamp by this method may have no connection with time such as *Windows Office 2007*, *2010 Sunset Boulevard*, or *call number 2008*. In this work, we tolerate this type of error because word sense disambiguation is a peripheral problem for this task.

Implicit Timestamp queries are resulted by removing all *year* indicators from the corresponding Explicit Timestamp queries. For example, the Implicit Timestamp query of *emnlp 2010* is *emnlp*. All other queries are No Timestamp queries because they have never been found together with a *year* indicator.

Classifying queries into the above three categories depends on the used query log. A search engine company partner provided us a query log from 08/01/2009 to 02/29/2010 for this research. We found the proportions of the three categories in this query log are 13.8% (Explicit), 17.1% (Implicit) and 69.1% (No Timestamp). These numbers

could be slightly different depending on the source of query logs. Note that 17.1% of Implicit Timestamp queries in the query log is a significant number. However, not all Implicit Timestamp queries are REQ. Many Implicit Timestamp queries have no time sense. They belong to Implicit Timestamp just because users issued the query with a *year* indicator through varied intents. For example, "google" is found to be an Implicit Timestamp query since there were many "google 2008" or "google 2009" in the query log.

The next few sections introduce our work in recognizing recurrent event time sense for Implicit Timestamp queries. We first focus on features. There are many features that were exploited in REQ classifier. We extract these features from query log, query session log, click log, search results, time series and NLP morphological analysis.

3.1 Query log analysis

The following features are extracted from query log analysis:

QueryDailyFrequency: the total counts of the query divided by the number of the days in the period.

ExplicitQueryRatio: Ratio of number of counts query was issued with year and number of counts query was issued with or without year. This feature is the method used by (Zhang et al., 2009).

UniqExplicitQueryCount: Number of uniq Explicit Timestamp queries associated with query. For example, if a query was issued with query+2009 and query+2008, this feature's value is two.

ChiSquareYearDist: this feature is the distance between two distributions: one is frequency distribution over years for all REQ queries. The other is that for single REQ query. It is calculated through following steps: (a) Aggregate the frequencies for all queries for all years. Suppose we observe all years from 2001 to 2010. So we can get vector, $E = (\frac{af_{10}}{sum1}, \frac{af_{09}}{sum1}, ..., \frac{af_{01}}{sum1})$ where af_i is the frequency sum of year $20i$ for all REQ queries. $sum1 = af_{10} + af_{09} + ... + af_{01}$, the sum of all year frequency. (b) Given a query, suppose we observe this query's yearly frequency distribution is , $O^q = (qf_{10}, qf_{09}, , ..., qf_{01})$. qf_i is this query's frequency for the year $20i$. Pad the slot with zeros if no frequency found. The expected distribution for this query is, $E^q = (\frac{sum2 * af_{10}}{sum1}, \frac{sum2 * af_{09}}{sum1}, ..., \frac{sum2 * af_{01}}{sum1})$, where $sum2 = qf_{10} + qf_{09} + ... + qf_{01}$ is sum of all year frequency for the query. (d) Calculate CHI-squared value to represent the different yearly frequency distribution between E^q and O^q according to $\chi^2 = \sum_{i=1}^{N} \frac{(O_i^q - E_i^q)^2}{E_i^q}$. Using CHI square distance as a method is widely used for statistical hypothesis test. We found it to be a useful feature for REQ classifier.

3.2 Query reformulation

If users cannot find the newest page by issuing Implicit Timestamp query, they may re-issue the query using an Explicit Timestamp query. We can detect this change in a search session (a 30 minutes period for each query). By finding this kind of behavior from users, we next extract three features.

UserSwitch: Number of unique users that switched from Implicit Timestamp queries to Explicit Timestamp queries.

YearSwitch: Number of unique year-like tokens switched by users in a query session.

NormalizedUserSwitch: Feature UserSwitch divided by QueryDailyFrequency.

3.3 Click log analysis

If a query is time sensitive, users may click a page that displays the year indicator on title or url. An example that shows year indicator on url is *www.lsi.upc.edu/events/emnlp2010/call.html*. Search engine click log saves all users' click information. We used click log to derive the following features.

YearUrlTop5CTR: Aggregated click through rate (CTR) of all top five URLs containing a year indicator. CTR of an URL is defined as the number of clicks of an URL divided by the number of page views.

YearUrlFPCTR: Aggregated click through rate (CTR) of all first page URLs containing a year indicator.

3.4 Search engine result set

For each Implicited Timestamp query, we can scrape the search engine to get search results. We count the number of titles and urls that contain year indicator. We use this number as a feature, and generate 6 features.

TitleYearTop5: the number of titles containing a year indication on the top 5 results. This value is 4 in Fig. 1.
TitleYearTop10: the number of titles containing a year indication on the top 10 results. This value is 6 in Fig. 1.
TitleYearTop30: the number of titles containing a year indication on the top 30 results.
UrlYearTop5: the number of urls containing a year indication on the top 5 results. This value is 1 in Fig. 1.
UrlYearTop10: the number of urls containing a year indication on the top 10 results.
UrlYearTop30: the number of titles containing a year indication on the top 30 results.

3.5 Time series analysis

Recurrent event query has periodic occurrence pattern in time series. Top graph of Figure 2 shows the frequency change of the query, "Oscar". The annual event usually starts from Oscar nomination as earlier as last year December to award announcement of February this year. So a small spike and a big spike are observed in the graph to indicate nomination period and ceremony period. There are a period of silence between the two periods. The frequency pattern keeps unchanged each year. We show three years (2007,8,9) in the graph. By making use of recurrent event queries' periodic properties, we calculated the query period as a new feature.

We use autocorrelation to calculate the period.

$$R(\tau) = \frac{\sum_{t=1}^{N-\tau}(x_t - \mu)(x_{t+\tau} - \mu)}{\{\sum_{t=1}^{N-\tau}(x_t - \mu)^2(x_{t+\tau} - \mu)^2\}^{1/2}}$$

where $x(t)$ is query daily frequency. N is the number of days used for this query. We can get maximum of 3 years data for some queries but only a few months for others. $R(\tau)$ is autocorrelation function. Peaks (the local biggest $R(\tau)$ given a time window) can be detected from $R(\tau)$ plot. The period T is calculated as the duration between two neighbor peaks. $T = 365$ for the query, "Oscar". The bottom graph of Fig. 2 shows the autocorrelation function plot for the query Oscar.

3.6 Recurrent event seed word list

Many recurrent event queries share some common words that have recurrent time sense. We list most frequently used recurrent seeds in Table 1. Those seeds are likely combined with other words to form new recurrent event queries. For example, the seed, "new", can be used by queries "new bmw cars", "whitney houston new songs", "apple new iphone", or "hairstyle new".

new	results	top	schedule
football	festival	movie	world
show	day	best	tax
result	calendar	honda	ford
download	exam	nfl	miss
awards	toyota	tour	sale
american	fair	list	pictures
election	game	basketball	cup

Table 1: Top recurrent event seed words

To generate the seed list, we tokenized all the queries from Implicit Timestamp queries and split all the tokens. We then sort and unique all the tokens, and submit top tokens to professional editors who are asked to pick 8,000 seeds from the top frequent tokens. Some top tokens were removed if they are not qualified to form recurrent event queries. The editors took about four days to do the judgment according to the token's time sense and examples of recurrent event queries. However, this is a one-time effort. A token will be in the seed if there are many recurrent event examples formed by this token, by editors' judgment.

Table 1 shows 32 top seeds. Some seeds connect with time such as, "new, schedule, day, best, calendar"; some relate to sports, "football, game, nfl, tour, basketball, cup"; some about cars, "honda, ford, toyota". The reason why "miss" is in the seeds is that there are many annual events about beauty contest such as "miss america, miss california, miss korea".

We use the seed list to generate the following three features:
AveNumberTokensSeeds: number of tokens that is in the seed list divided by number of tokens in the query.
AveNumberTokensNotSeeds: number of tokens that is not in the seed list divided by number of tokens in the query.
DiffNumberTokensSeeds: The difference of the above two values.

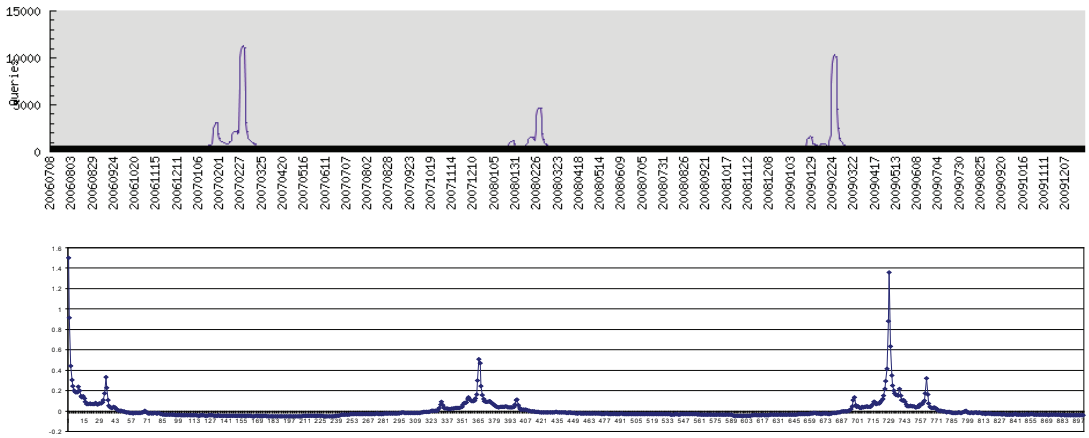

Figure 2: Frequency waveform(top) and corresponding autocorrelation curve (bottom) for query *Oscar*.

4 Learning Approach for REQ

The REQ classification is a typical machine learning task. Given M observed samples used for training data, $\{(\mathbf{x}_0, y_0), (\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_M, y_M)\}$ where $\mathbf{x}_i$ is a feature vector we developed in last section for a given query. y_i is the observation value, $\{+1, -1\}$, indicating the class of REQ and non-REQ. The task is to find the class probability given an unknown feature vector, $\mathbf{x}'$, that is,

$$p(y = c|\mathbf{x}'), \quad c = +1, -1. \tag{1}$$

There are a lot of machine learning methods applicable to implement Eq. 1. In this work, we adopted three representative methods.

The first method is Naive Bayes method. This method treats features independent. If $\mathbf{x}$ is enxtended into feature vector, $\mathbf{x} = \{x_0, x_1, \cdots, x_N\}$ then,

$$p(y = c|\mathbf{x}) = \frac{1}{Z} p(c) \prod_{i=0}^{i=N} p(x_i|c)$$

The second method is SVM. In this work we used the tool for our experiments, LIBSVM (Chang and Lin, 2001). Because SVM is a well known approach and widely used in many classification task, we skip to describe how to use this tool. Readers can turn to the reference for more details.

The third method is based on decision tree based logistic regression model. The probability is given by the formula below,

$$p(y = c|\mathbf{x}) = \frac{1}{1 + e^{-f(\mathbf{x})}} \tag{2}$$

We employ Gradient Boosted Decision Tree algorithm (Friedman, 2001) to learn the function $f(X)$. Gradient Boosted Decision Tree is an additive regression algorithm consisting of an ensemble of trees, fitted to current residuals, gradients of the loss function, in a forward step-wise manner. It iteratively fits an additive model as

$$f_t(x) = T_t(x; \Theta) + \lambda \sum_{t=1}^{T} \beta_t T_t(x; \Theta_t)$$

such that certain loss function $L(y_i, f_T(x + i))$ is minimized, where $T_t(x; \Theta_t)$ is a tree at iteration t, weighted by parameter β_t, with a finite number of parameters, Θ_t and λ is the learning rate. At iteration t, tree $T_t(x; \beta)$ is induced to fit the negative gradient by least squares.

The optimal weights of trees β_t are determined by

$$\beta_t = \text{argmin}_\beta \sum_{i}^{N} L(y_i, f_{t-1}(x_i) + \beta T(x_i, \theta))$$

Each node in the trees represents a split on a feature. The tuneable parameters in such a machine-learnt model include the number of leaf nodes in each tree, the relative contribution of score from each tree called the shrinkage, and total number of shallow decision trees.

The relative importance of a feature S_i, in such forests of decision trees, is aggregated over all the

m shallow decision trees (Breiman et al., 1984) as follows:

$$S_i^2 = \frac{1}{M}\sum_{m=1}^{M}\sum_{n=1}^{L-1}\frac{w_l * w_r}{w_l + w_r}(y_l y_r)^2 I(v_t = i) \quad (3)$$

where v_t is the feature on which a split occurs, y_l and y_r are the mean regression responses from the right, and left sub-tree, and w_l and w_r are the corresponding weights to the means, as measured by the number of training examples traversing the left and right sub-trees.

5 REQ Learner Evaluation

We collected 6,000 queries labeled as either Recurrent or Non-recurrent by professional human editors. The 6,000 queries were sampled from Implicit Timestamp queries according to frequency distribution to be representative. We split the queries into 5,000 for training and 1,000 for test. For each query, we calculated features' values as described in Section 3.

The Naive Bayes method used single Gaussian function for each independent feature. Mean and variance were calculated from the training data.

As for LIBSVM, we used C-SVC, linear function as kernel and 1.0 of shrinkage.

The parameters used in the regression model were 20 of trees, 20 of nodes and 0.8 of learning rate (shrinkage).

The test results are shown in Fig. 3, recall-precision curve. We set a series of threshold to the probability of $c = +1$ calculated by Eq. 1 so that we can get the point values of recall and precision in Fig. 3. For example, if we set a threshold of 0.6, a query with a probability larger than 0.6 is classified as REQ. Otherwise, it is non-REQ. The precision is a measure of correctly classified REQ queries divided by all classified REQ queries. The recall is a measure of correctly classified REQ queries divided by all REQ queries in test data.

In addition to the three plots, we also show the results using only one feature, ExplicitQueryRatio, for comparison with the classification method used by (Zhang et al., 2009).All the three models using all features performed better than the existing method using ExplicitQueryRatio. The highest improvement was achieved by GBDT regression tree

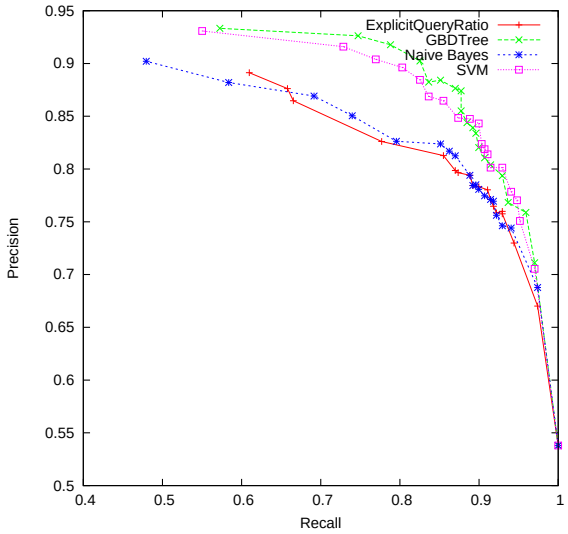

Figure 3: Comparison of precision and recall rate between our method and the existing method.

model. The results of Naive Bayes were lower than SVM and GBDTree. This model is weaker because it treats features independently. Typically SVMs and GBDT gives comparable results on a large class of problems. Since for this task we use features from different sources, the feature values are designed to have larger dynamic range, which is better handled by GBDT.

The features' importance ranked by Equation 3 is shown in Table 2. We list the top 10 features. The No.1 important feature is ExplicitQueryRatio. The second and seventh features are from search session analysis by counting users who changed queries from Implicit Timestamp to Explicit Timestamp. This is a strong source of features. The time series analysis feature is ranked No.3. Calculation of this feature needs two years query log to be much more effective, but we didn't get so large data for many queries. One of the features from recurrent event seed list is ranked No.4. This is also an important feature source. The ChiSquareYearDist feature is ranked 5th, that proves the recurrent event query frequency has a statistical distribution pattern over years. TitleYearTop30 and TitleYearTop10 that are derived from scraping results are ranked the 9th and 10th important.

Fig. 4 shows the distribution of feature values for

Feature	Rank	Score
ExplicitQueryRatio	1	100
NormalizedUserSwitch	2	71.7
AutoCorrelation	3	54.0
AveNumberTokenSeeds	4	48.7
ChiSquareYearDist	5	36.3
YearUrlFPCTR	6	19.1
UserSwitch	7	11.7
QueryDailyFreq	8	10.7
TitleYearTop30	9	10.6
TitleYearTop10	10	5.8

Table 2: Top 10 most important features: rank and importance score (100 is maximum)

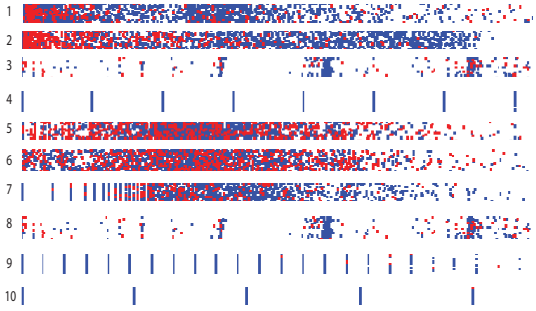

Figure 4: Feature value distribution of all data (blue=REQ, red=non-REQ)

each sample of the 6,000 data, where each point represents a query and each line represents a feature's value for all queries. One point is a query. The features are ordered according to feature importance of Table 2. The "blue" points indicate REQ queries and the "red" points, non-REQ queries. Some features are continuous like the 1st and 2nd. Some feature values are discrete like the last two indicating TitleYearTop30 and TitleYearTop10. There are "red" samples in the 4th feature but overlapped with and covered by "blue" samples visually.

In the Table 3, we show F-Measure values as we gradually added features from the feature, ExplicitQueryRatio, according to feature importance in Table 2. We listed the F-Measure values under three threshold, 0.6, 0.7 and 0.8. Higher threshold will increase classifier precision rate but reduce recall rate. F-Measure is a metric combining precision rate and recall rate. It is clearly observed that the classifier performance is improved as more features are used.

	Threshold		
Feature	0.6	0.7	0.8
ExplicitQueryRatio	0.833	0.833	0.752
+NormalizedUserSwitch	0.840	0.837	0.791
+AutoCorrelation	0.850	0.839	0.823
+AveNumberTokenSeeds	0.857	0.854	0.834
+ChiSquareYearDist	0.857	0.864	0.839
+YearUrlFPCTR	0.869	0.867	0.837
+UserSwitch	0.862	0.862	0.846
+QueryDailyFreq	0.860	0.852	0.847
+TitleYearTop30	0.854	0.853	0.843
+TitleYearTop10	0.858	0.861	0.852
+All	0.876	0.867	0.862

Table 3: F-Measures as varying thresholds by adding top features.

Query	Probability
ncaa men's basketball tournament	0.999
bmw 328i sedan reviews	0.999
new apple iphone release	0.932
sigir	0.920
new york weather in april	0.717
academy awards reviews	0.404
google ipo	0.120
adidas jp	0.082

Table 4: Probabilities of example queries by GBDT tree classifier

Some query examples, and their scores from our model are listed in Table 4. The last two examples, *google ipo* and *adidas jp*, have very low values, and are not REQs. The first four queries are typical REQs. They have higher values of features ExplicitQueryRatio,Normalized UserSwitch and YearUrlFPCTR. Although both *new apple iphone release reviews* and *academy awards reviews* are about reviews, *academy awards reviews* has lower value of NormalizedUserSwitch and ChiSquareYearDist could be the reason for a lower score.

6 Web Search Ranking

In this section, we use the approach proposed by (Zhang et al., 2009) to test the REQ classifier for Web search ranking. In their approach, search ranking is altered by boosting pages with most recent year if the query is a REQ. The year indicator

		DCG@5			DCG@1		
bucket	#(query)	Organic	Our's	% over Organic	Organic	Ours	% over Organic
[0.0,0.1]	59	6.87	6.96	1.48(-2.3)	4.08	4.19	2.69(-1.07)
[0.1,0.2]	76	5.86	6.01	2.52(0.98)	2.88	2.91	1.14(1.69)
[0.2,0.3]	85	6.33	6.41	1.24(2.12)	3.7	3.7	0.0(0.8)
[0.3,0.4]	75	5.18	5.24	1.18(-0.7)	2.92	2.95	1.14(1.37)
[0.4,0.5]	78	4.96	4.82	-2.84(-1.35)	2.5	2.42	-3.06(0)
[0.5,0.6]	84	5.4	5.37	-0.45(-0.3)	2.82	2.85	1.05(-1.5)
[0.6,0.7]	78	4.78)	5.19)	8.42(3.64)	2.56	2.83	10.75(4.1)
[0.7,0.8]	80	4.45	4.60	3.41(3.19)	2.21	2.26	1.98(2.8)
[0.8,0.9]	78	4.81	4.96	3.15(4.79)	2.32	2.33	0.55(0.65)
[0.9,1.0]	107	5.08	5.50	**8.41***(4.41)	2.64	3.09	**16.78***(1.36)
[0.0,1.0]	800	5.33	5.47	**2.74***(2.17)	2.83	2.93	**3.6***(1.26)

Table 5: REQ learner improves search engine organic results. The numbers in the brackets are by Zhang's methods. Direct comparison with Zhang's method is valid only in the last line, using all queries. A sign "$*$" indicates statistical significance (p-value<0.05)

can be detected either from title or URL of the result. For clarity, we re-write their ranking function as below,

$$F(q,d) = R(q,d) + [e(d_o,d_n) + k]e^{\lambda\alpha(q)}$$

where the ranking function, $F(q,d)$, consists of two parts: the base function $R(q,d)$ plus boosting. If the query q is not a REQ, boosting is set to zero. Otherwise, boosting is decided by $e(d_o,d_n)$, k, λ and $\alpha(q)$. $e(d_o,d_n)$ is the difference of base ranking score between the oldest page and the newest page. If the newest page has a lower ranking score than the oldest page, then the difference is added to the newest page to promote the ranking of the newest page.

$\alpha(q)$ is the confidence score of a REQ query. It is the value of Eq. 1. λ and k are two empirical parameters. (Zhang et al., 2009)'s work has experimented the effects of using different value of λ and k ($\lambda = 0$ equals to no discounts for ranking adjustment). We used $\lambda = 0.4$ and $k = 0.3$ which were the best configuration in (Zhang et al., 2009).

For evaluating our methods, we randomly extracted 800 queries from the Implicit Timestamp queries. We scraped a commercial search engine using the 800 queries. We extracted the top five search results for each query under three configures: organic search engine results, (Zhang et al., 2009)'s method and ours using REQ classifier. We asked human editors to judge all the scraped (query, url) pairs. Editors assign five grades according to relevance between query and articles: Perfect, Excellent, Good, Fair, and Bad. For example, a "Perfect" grade means the content of the url match exactly the query intent.

We use Discounted Cumulative Gain (DCG) (Jarvelin and Kekalainen, 2002) at rank k as our primary evaluation metrics to measure retrieval performance. DCG is defined as,

$$DCG@k = \sum_{i=1}^{k} \frac{2^{r(i)} - 1}{log_2(1+i)}$$

where $r(i) \in \{0 \ldots 4\}$ is the relevance grade of the ith ranked document.

The Web search ranking results are shown in Table 5. We used GBDT tree learning methods because it achieved the best results. We divided 800 test queries into 10 buckets according to the classifier probability. The bucket, [0.0,0.1], contains the query with a classifier probability greater than 0 but less than 0.1. Our results are compared with organic search results, but we also show the improvements over search organic by (Zhang et al., 2009) in the brackets. Because Zhang's approach output different classifier values from Ours for the same query, buckets of the same range in the Table contain different queries. Hence, it is inappropriate to compare

Zhang's with Ours for the same buckets except the last row where we used all the queries.

Our classifier's overall performance is much better than the organic search results. We achieved 2.74% DCG@5 gain and 3.6% DCG@1 gain over organic search for all queries. The gains are higher than (Zhang et al., 2009)'s results with regards to improvement over organic results. By direct comparison, Ours was 2.7% better than Zhangs significantly in terms of DCG@1 by Wilcoxon significant test. DCG@5 is 1.1% better, but not significant. The table also show that the higher buckets with higher probability achieved higher DCG gain than the lower buckets overall. Our approach observed 16.78% DCG@1 gain for bucket [0.9,1.0]. This shows that our methods are very effective.

7 Conclusions

We found most of REQ are long tail queries that pose a major challenge to Web search. We have demonstrated learning REQ is important for Web search. this type of queries can't be solved in traditional ranking method. We found building a REQ classifier was a good solution. Our work described using machine learning method to build REQ classifier. Our proposed methods are novel comparing with traditional query classification methods. We identified and developed features from query log, search session, click and time series analysis. We applied several ML approaches including Naive Bayes, SVM and GBDT tree to implement REQ learner. Finally, we show through ranking experiments that the methods we proposed are very effective and beneficial for search engine ranking.

Acknowledgements

We express our thanks to who have assisted us to complete this work, especially, to Fumiaki Yamaoka, Toru Shimizu, Yoshinori Kobayashi, Mitsuharu Makita, Garrett Kaminaga, Zhuoran Chen.

References

R. Baeza-Yates, F. Saint-Jean, and C. Castillo. 2002. Web dynamics, age and page qualit. *String Processing and Information Retrieval*, pages 453–461.

Steven M. Beitzel, Eric C. Jensen, Ophir Frieder, David Grossman, David D. Lewis, Abdur Chowdhury, and Aleksandr Kolcz. 2005. Automatic web query classification using labeled and unlabeled training data. In *SIGIR '05*, pages 581–582.

K. Berberich, M. Vazirgiannis, and G. Weikum. 2005. Time-aware authority rankings. *Internet Math*, 2(3):301–332.

L. Breiman, J. Friedman, R. Olshen, and C. Stone. 1984. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA.

S. Brin and L. Page. 1998. The anatomy of a large-scale hypertextual web search engine. *Proceedings of International Conference on World Wide Web*.

Andrei Z. Broder, Marcus Fontoura, Evgeniy Gabrilovich, Amruta Joshi, Vanja Josifovski, and Tong Zhang. Robust classification of rare queries using web knowledge. In *SIGIR '07*, pages 231–238.

Chih-Chung Chang and Chih-Jen Lin, 2001. *LIBSVM: a library for support vector machines*. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

J. Cho, S. Roy, and R. Adams. 2005. Page quality: In search of an unbiased web ranking. *Proc. of ACM SIGMOD Conference*.

F. Diaz. 2009. Integration of news content into web results. *Proceedings of the Second ACM International Conference on Web Search and Data Mining (WSDM)*, pages 182–191.

Anlei Dong, Yi Chang, Zhaohui Zheng, Gilad Mishne, Jing Bai, Ruiqiang Zhang, Karolina Buchner, Ciya Liao, and Fernando Diaz. 2010a. Towards recency ranking in web search. *Proceedings of the Third ACM International Conference on Web Search and Data Mining (WSDM)*, pages 11–20.

Anlei Dong, Ruiqiang Zhang, Pranam Kolari, Jing Bai, Fernando Diaz, Yi Chang, Zhaohui Zheng, and Hongyuan Zha. 2010b. Time is of the essence: improving recency ranking using twitter data. *19th International World Wide Web Conference (WWW)*, pages 331–340.

Jonathan L. Elsas and Susan T. Dumais. 2010. Leveraging temporal dynamics of document content in relevance ranking. In *WSDM*, pages 1–10.

J. H. Friedman. 2001. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232.

Kalervo Jarvelin and Jaana Kekalainen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20:2002.

K. Sparck Jones, S. Walker, and S. E. Robertson. 2000. A probabilistic model of information retrieval: development and comparative experiments. *Inf. Process. Manage.*, 36(6):779–808.

A. C. Knig, M. Gamon, and Q. Wu. 2009. Click-through prediction for news queries. *Proc. of SIGIR*, pages 347–354.

Ying Li, Zijian Zheng, and Honghua (Kathy) Dai. 2005. Kdd cup-2005 report: facing a great challenge. *SIGKDD Explor. Newsl.*, 7(2):91–99.

Xiao Li, Ye yi Wang, and Alex Acero. 2008. Learning query intent from regularized click graphs. In *In SIGIR 2008*, pages 339–346. ACM.

Donald Metzler, Rosie Jones, Fuchun Peng, and Ruiqiang Zhang. 2009. Improving search relevance for implicitly temporal queries. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 700–701.

S. Nunes. 2007. Exploring temporal evidence in web information retrieval. *BCS IRSG Symposium: Future Directions in Information Access*.

S. Pandey, S. Roy, C. Olston, J. Cho, and S. Chakrabarti. 2005. Shuffling a stacked deck: The case for partially randomized ranking of search engine results. *VLDB*.

G. Salton and M. J. McGill. 1983. *Introduction to modern information retrieval*. McGraw-Hill, NY.

Dou Shen, Rong Pan, Jian-Tao Sun, Jeffrey Junfeng Pan, Kangheng Wu, Jie Yin, and Qiang Yang. 2005. Q2c@ust: our winning solution to query classification in kddcup 2005. *SIGKDD Explor. Newsl.*, 7(2):100–110.

Dou Shen, Rong Pan, Jian-Tao Sun, Jeffrey Junfeng Pan, Kangheng Wu, Jie Yin, and Qiang Yang. 2006. Query enrichment for web-query classification. *ACM Trans. Inf. Syst.*, 24(3):320–352.

Ruiqiang Zhang, Yi Chang, Zhaohui Zheng, Donald Metzler, and Jian-yun Nie. 2009. Search result re-ranking by feedback control adjustment for time-sensitive query. In *HLT-NAACL '09*, pages 165–168.