Ranking using Multiple Document Types in Desktop Search

Jinyoung Kim and W. Bruce Croft
Center for Intelligent Information Retrieval
Department of Computer Science
University of Massachusetts Amherst
{jykim,croft}@cs.umass.edu

ABSTRACT

A typical desktop environment contains many document types (email, presentations, web pages, pdfs, etc.) each with different metadata. Predicting which types of documents a user is looking for in the context of a given query is a crucial part of providing effective desktop search. The problem is similar to selecting resources in distributed IR, but there are some important differences.

In this paper, we quantify the impact of type prediction in producing a merged ranking for desktop search and introduce a new prediction method that exploits type-specific metadata. In addition, we show that type prediction performance and search effectiveness can be further enhanced by combining existing methods of type prediction using discriminative learning models. Our experiments employ pseudo-desktop collections and a human computation game for acquiring realistic and reusable queries.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: [Information Search and Retrieval]

General Terms

Algorithms

Keywords

Information Retrieval, Desktop Search, Semi-structured Document Retrieval, Type Prediction

1. INTRODUCTION

People have many types of documents on their desktop with different sets of metadata for each type. For instance, emails have sender and receiver fields, whereas office documents have filename and author fields. Considering that personal information is now increasingly spread across various places on the web, this diversity of document types is continuing to increase.

SIGIR'10 July 19–23, 2010, Geneva, Switzerland.

Desktop search systems, which is now a standard feature of most platforms, have tried to exploit this type information by presenting search results for each document type separately (e.g. Apple Spotlight) or showing type information distinctively in a single rank list (e.g. Google Desktop).

In both scenarios, a critical aspect of the system is being able to predict which type(s) of document(s) a user is looking for given a query. Given these properties, if the system displays separate type-specific results, it can rank them by their type scores. Alternatively, the system can incorporate type scores into document ranking as a feature.

The type prediction problem bears some similarity to the vertical or resource selection problem in the areas of vertical or federated search in that the system tries to score the results from each vertical, resource, or collection based on predicted relevance for a given query. In this sense, all these problems can be put in a broad category of collection scoring. There are, however, several notable differences.

First, type-specific sub-collections in the desktop are *cooperative* in that all the documents are available to s single system. This means that sampling techniques used for federated search may not be necessary for desktop search; second, unlike typical collections used for vertical or federated search research, the sub-collections in desktop environment are small and have considerable topical overlap. This makes it challenging to apply content-based collection scoring techniques (e.g. CORI [3]) directly; third, each sub-collections in the desktop has unique metadata that has not been exploited in existing collection scoring methods.

The main goal of this paper is to show how the retrieval effectiveness of a desktop search system can be enhanced by improving type prediction performance. We focus on known-item queries, which are the most frequent type of request in the desktop environment [7], and assume that the system displays a final rank list by merging type-specific results.

Our work makes several contributions: first, we demonstrate the impact of sub-collection retrieval and type prediction performance on the quality of the final rank list; second, we suggest a type prediction method that exploits type-specific metadata and show that the new method has better performance than a state-of-the-art collection scoring method; third, we find that the combination of existing collection scoring methods can improve the performance further; fourth, we employ a game interface to collect a large quantity of known-item queries in a reasonably realistic setting.

The rest of this paper is organized as follows. We provide an overview of related work in the next section. Then

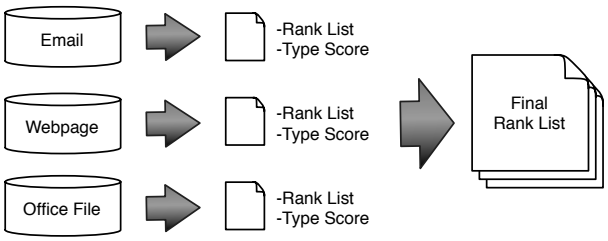

Figure 1: Suggested retrieval model for desktop search.

we introduce the retrieval model, type prediction methods and test collection generation methods we used. In the experiments, we report retrieval and type prediction performance using pseudo-desktop collections and a computer science (CS) collection where queries are collected by the game interface.

2. RELATED WORK

Related work can be found in several different areas: desktop search, vertical search and distributed IR.

For desktop search, people have studied user interface issues [6], retrieval models and evaluation methods [9]. Among these, Thomas et al. [18] views desktop search as a meta-search problem where the results from many servers are merged. They compared several server selection methods using documents collected from various sources, concluding that a selection method based on Kullback-Leibler divergence [17] performed the best. Our work extends this work by suggesting a prediction method that exploits the field structure and a combination method whose performance can be improved by interaction with the user.

To evaluation desktop search, methods for building test collections [4] [9] have been proposed. Among these, the pseudo-desktop method by Kim et al. [9] generated test collections automatically by simulation. Our work employs these pseudo-desktop collections for the retrieval experiments and improves the procedure of gathering human-generated queries by employing a game interface.

In the context of vertical search and distributed IR, researchers have proposed many methods of scoring collections against a given query. Approaches such as CORI [3] and KL-Divergence [17] treat collections as large documents and apply document scoring techniques for scoring collections. Other methods, such as ReDDE [16] model the distribution of relevant documents for each collection. Recently, Arguello et al. proposed a classification approach [1] [2] where many sources of evidences can be combined for mapping a user's query into one or more collections. Our combination approach is similar to this work but we use features and evaluation methods more suitable for our problem domain.

3. RETRIEVAL MODEL

In this section we introduce a retrieval model for desktop search. In our retrieval model, as depicted in Figure 1, type-specific results (rank list and type score) are merged into a final rank list. We first explain methods used for the retrieval of sub-collections corresponding to each file type. Then we introduce the type prediction and result merging methods we used.

The following notation will be used throughout this paper. We assume that a query $Q = (q_1, ..., q_m)$ is composed of m words and each collection C contains documents of n field types $(F_1, ..., F_n)$ where n can be different for each collection. Each document d in the collection may include fields $(f_1, ..., f_n)$, where each field is marked using lowercase letters to distinguish it from the corresponding field type in the schema. Model-specific parameters will be explained as they appear.

3.1 Type-specific Retrieval

The first step in our retrieval model is ranking documents from each sub-collection. Since our focus is on type prediction, we employ retrieval models used in the recent work by Kim et al. [9] on desktop search, which includes document query-likelihood (DLM), the probabilistic retrieval model for semistructured data (PRM-S) and the interpolation of DLM and PRM-S (PRM-D). We explain PRM-S model in the following section.

3.1.1 Probabilistic Retrieval Model for Semi-structured Data

The probabilistic retrieval model for semistructured data (PRM-S) [10] scores documents by combining field-level query-likelihood scores similarly to other field-based retrieval models [12]. The main feature of the PRM-S model is that weights for combining field-level scores are estimated based on the predicted mapping between query terms and document fields, which can be efficiently computed based on collection term statistics.

More formally, using Bayes' theorem, we can estimate the posterior probability $P_M(F_j|w)$ that a given query term w is mapped into document field F_j by combining the prior probability $P_M(F_j)$ and the probability of a term occurring in a given field type $P_M(w|F_j)$.

$$P_M(F_j|w) = \frac{P_M(w|F_j)P_M(F_j)}{\sum_{F_k \in F} P_M(w|F_k)P_M(F_k)} \quad (1)$$

Here, $P_M(w|F_j)$ is calculated by dividing the number of occurrences for term w by total term counts in the field F_j across the whole collection. Also, $P_M(F_j)$ denotes the prior probability of field F_j mapped into any query term before observing collection statistics.

With the mapping probabilities estimated as described above, the probabilistic retrieval model for semistructured data (PRM-S) can use these as weights for combining the scores from each field $P_{QL}(w|f_j)$ into a document score, as follows:

$$P(Q|d) = \prod_{i=1}^{m} \sum_{j=1}^{n} P_M(F_j|q_i)P_{QL}(q_i|f_j) \quad (2)$$

This model was shown to have a better performance than other field-based retrieval methods such as the mixture of field language models [12] and BM25F [14] for a semi-structured document retrieval task using IMDB [10] and TREC email [9] collections.

3.2 Type Prediction

As briefly mentioned in the introduction, an integral component for our retrieval model is the type prediction component, which scores each collection given a user query. The

type prediction method will produce scores for each sub-collection and these scores can be used for merging results into the final rank list or ranking sub-collection results. Section 4 deals with type prediction methods in detail.

3.3 Result Merging

With type-specific rank lists from sub-collection retrieval and collection scores from the type prediction component, we can produce the final rank list by rank-list merging algorithms. In this work, we use the well-known CORI algorithm for merging [3].

$$C_i' = (C_i - C_{min})/(C_{max} - C_{min}) \quad (3)$$

$$D' = (D - D_{min})/(D_{max} - D_{min}) \quad (4)$$

$$D'' = \frac{D' + 0.4 \cdot D' \cdot C_i'}{1.4} \quad (5)$$

Here, C_i' and D' are normalized collection and document score, computed using the maximum and minimum of collection scores (C_{max} / C_{min}) and document scores (D_{max} / D_{min}), respectively. Given C_i' and D', the final document score D'' can be computed by combining these two scores.

4. TYPE PREDICTION METHODS

In this section, we introduce our type prediction methods in detail. We first describe existing methods for type prediction which are adopted from recent works on vertical and federated search [1] [2]. Then we introduce a new type prediction method that exploits document metadata. Lastly, we explain how type prediction methods can be combined using several learning methods.

4.1 Existing Methods for Type Prediction

4.1.1 Query-likelihood of Collection

Many traditional resource selection methods (e.g. CORI) are computed from collection term statistics. Among these, we use collection query-likelihood (CQL)[17], which is a resource selection method based on the language modeling approach. The approach here is to collapse all documents in each collection into one giant 'document' and use the query-likelihood score for the document as the collection score:

$$CQL(Q, C) = \prod_{q \in Q} (\lambda P(q|C) + (1 - \lambda) P(q|G)) \quad (6)$$

Here, C is the language model of each sub-collection and G is the language model of the whole collection. The smoothing parameter λ adjusts the interpolation ratio of $P(q|C)$ and $P(q|G)$. CQL was shown to be the most effective among resource selection methods in a recent evaluation [18].

4.1.2 Query-likelihood of Query Log

Another source of evidence for the type prediction is the aggregated query terms used for finding documents that belong to each sub-collection. As done in previous work [1] [2], we use the query-likelihood score of the language model (QQL) built by queries targeted for sub-collection C as shown below:

$$QQL(Q, C) = \prod_{q \in Q} (\lambda P(q|L_C) + (1 - \lambda) P(q|L_G)) \quad (7)$$

Here, L_C is the language model of the query log corresponding to collection C. L_G is similarly defined using the query log across all collections.

4.1.3 Geometric Average

Another class of resource selection methods combine the score of top documents to evaluate each collection given the user query. Seo et al. [15] proposed using the geometric mean of the top m documents as the combination method,

$$GAVG(Q, C) = (\prod_{d \in D_{top}} P(Q|d))^{\frac{1}{m}} \quad (8)$$

where D_{top} is the set of top m documents from the collection and the score $P(Q|d)$ is padded with $P_{min}(Q|d)$ if fewer than m documents are retrieved.

4.1.4 ReDDE

ReDDE [16] [2] scores a target collection based on the expected number of documents relevant to the query. Although previous work used a centralized sample index to derive this expectation, we can estimate this directly from the target collection,

$$ReDDE(Q, C) = \sum_{d \in D_{top}} P(Q|d) \quad (9)$$

which is equivalent to using the sum of the top document scores belonging to each collection. Intuitively, this results in a higher score for the collection with more documents in higher positions.

4.1.5 Query Clarity

So far, most of our methods have been derived from resource selection techniques developed in the context of distributed IR. Query performance prediction methods can also be used for type prediction by assigning a higher score for the collection with higher predicted performance. Among such methods, we employ Query Clarity [5], which predicts performance using the KL divergence between a query language model and a collection language model.

$$Clarity(Q, C) = \sum_{w \in V} P(w|L_Q) log_2 \frac{P(w|L_Q)}{P(w|C)} \quad (10)$$

Here, query language model L_Q is estimated from the top m documents from the collection.

4.1.6 Dictionary-based Matching

In some cases, users provide direct clues about which file type they intended to search, by including terms such as 'sender'(for email), 'pdf'(for office document) or 'www'(for webpage). Although these terms may not occur in a majority of queries, they can be a strong indication of type membership for a given query. We built the dictionary for each sub-collection by using the names of the collection and metadata fields.

4.2 Using Document Metadata Fields for Type Prediction

Although some of methods introduced above use the collection term statistics, none use the field structure of documents available for desktop collection. Considering that

the retrieval effectiveness of semi-structured document collections have been improved by exploiting this structure [10], we can expect similar benefits for the type prediction problem.

Field-based collection query likelihood (FQL) – our new method for type prediction – extends the collection query likelihood model for collection scoring by combining the query-likelihood score for each field of the collection instead of using the score for the whole collection. In other words, if we borrow the view of query-term and field mapping described in Section 3.1.1, we try to infer the mapping between a user query and each collection by combining mapping probabilities for the fields of each collection.

More formally, for a collection C that contains documents of n field types $(F_1, ..., F_n)$, we can combine the language model score of each field as follows:

$$FQL(Q, C) = comb_{F_i \in C}(P(q|F_i)) \quad (11)$$

Here, F_i is a smoothed language model of the ith field of the collection and *comb* can be any function that can combine n numbers into one. We experimented with many variations of *comb* function and found that arithmetic mean gives the best performance.

4.3 Combining Type Prediction Methods

Considering that the type prediction methods introduced so far are derived from different sources, it is plausible that we can get further performance benefits by combining individual methods in a linear model where weights are found using learning methods. In this section, we describe three learning methods with different objective functions: grid search of parameter values, a multi-class classifier and a rank-learning method.

4.3.1 Grid-search of Parameter Values

Since we have only seven features to be combined, It is feasible to perform a grid search of parameter values that maximize the performance of a training set of queries. Specifically, we can find the optimal value for each parameter in turns while fixing the values of the parameters previously found, and repeating the whole procedure until we reach convergence. In searching for the optimum value of each parameter, we employed Golden Section Search[13].

4.3.2 Multi-class Classification

Given that we want to predict one of document k types for a given query, this is typical multi-class classification scenario where each type corresponds to a class. Among many choices of such methods, we used a one-vs-rest (OVR) support vector machine classifier (MultiSVM) available in Liblinear Toolkit[1]. Since the output of this classifier is not suitable to be used directly as type scores, we used a simple linear transform to convert the scores into probabilities.

4.3.3 Rank-learning Method

Alternatively, one can cast the type prediction task as a ranking problem where we try to rank the relevant collections higher than non-relevant collections. This approach can be especially beneficial for the case where the user is finding multiple documents with different types, since such situation is hard to model with typical multi-class classification methods. RankSVM [8] was used as the rank-learning method.

[1]http://www.csie.ntu.edu.tw/ cjlin/liblinear/

5. TEST COLLECTION GENERATION METHODS

Evaluation of desktop search has been considered a challenging issue due to the private nature of collections. As a solution, methods for building reusable test collections have emerged recently [4] [9]. Here we review the test collection generation methods and introduce our game interface for gathering known-item queries.

5.1 Pseudo-desktop

Kim et al. [9] introduced the pseudo-desktop method of automatically building a reusable test collection for desktop search. They collected documents with similar characteristics to a typical desktop environment and generated queries by statistically taking terms from each of the target documents. Then generated queries are validated against a set of manually written queries, which were gathered by showing people documents and asking them to write queries for finding those documents.

Although this pseudo-desktop method provides a cost-effective way to generate a large quantity of test data under perfect experimental control, there are several limitations: first, the query generation procedure is too simple in that it independently chooses words from the target document, while real queries contain phrases; second, only terms in target documents are used as queries; third, the procedure of getting manual queries is not realistic in that people were asked to formulate queries for documents they were not familiar with.

It should be possible to mitigate the problems above by refining the query generation method, but we decided instead to improve the procedure of gathering manual queries by developing a human computation game, as explained in the next section.

5.2 DocTrack Game

Human computation games [19] have recently been suggested as method for getting a large amount of human annotations in a way that motivates participants using a game-like setting. In the context of IR research, Ma et al. [11] introduced Page Hunt, which is a game designed to collect web search log data by showing each participant webpages and asking her to find them with the search interface provided.

By adapting Page Hunt to our problem domain, we developed the DocTrack game for gathering known-item queries in desktop environment, as shown in Figure 2. In addition to using documents of many types that might be found in a desktop instead of random webpages, we made several modifications to the original Page Hunt game: first, since people generally have good knowledge of their own desktops, we collected documents that participants are familiar with and let each of them browse the collection for some time before starting the game; second, to simulate a typical known-item search scenario, we showed participants multiple documents and asked them to find one of them without specifying them which one is the target document; third, we used a document viewer that can show documents of any types (e.g. pdf, doc and ppt) in the same way they are seen on the desktop.

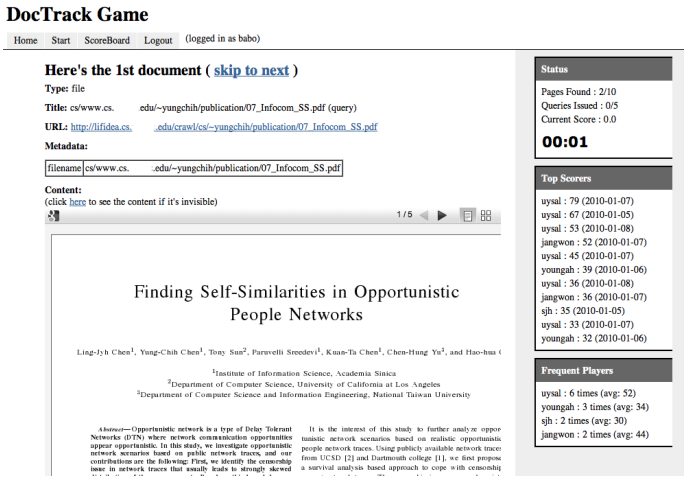

Figure 2: The screenshot of the DocTrack game. The user is being shown a document.

Compared to the method of collecting manual queries in Kim et al.[9], using the DocTrack game, we could gather many more realistic queries together with the whole session log data. This in turn allowed us to perform the training of discriminative learning models which typically requires large amounts of training data.

6. EXPERIMENTS

In this section we describe the experiments for verifying the type prediction and retrieval performance. We used three pseudo-desktop collections with generated queries for the first experiment, where we compared several type prediction methods and showed the impact of type prediction on the final ranking. We then report on experiments using a computer science (CS) collection where queries were collected by the DocTrack game.

In the indexing of both collections, each word was stemmed using the Krovetz stemmer and standard stopwords were eliminated. Indri[2] was used as a retrieval engine for all the retrieval experiments. We used prediction accuracy to evaluate type prediction performance, since we have only one *correct* collection for each query. Mean Reciprocal Rank was used as the measure of retrieval performance for all experiments, since this is a known-item task where each query has only one relevant document.

Four retrieval methods were used for each sub-collection (DLM / PRM-S / PRM-D / Best) and four methods (Uniform / CQL / FQL / Oracle) were used for type prediction. We compared only CQL and FQL for the pseudo-desktop experiment since CQL was shown to be the most effective among collection scoring method [18] and FQL is the extension of CQL for semi-structured document collection. Section 6.2 provides the comparison with other type prediction methods using the CS collection and queries from the DocTrack game.

For the Best retrieval method, we used the retrieval method with the best aggregate performance for each sub-collection, making the assumption that the best-performing retrieval method is known in advance. For Uniform and Oracle collection scoring, we considered that each collection has the same chance of containing the relevant document (Uniform) or that we have the perfect knowledge of the collection that contains the relevant document (Oracle).

[2]http://www.lemurproject.org

Table 1: Number and average length of documents for pseudo-desktop collections.

Type	Jack		Tom		Kate	
	#Docs	Len.	#Docs	Len.	#Docs	Len.
email	6067	555	6930	558	1669	935
html	953	3554	950	3098	957	3995
pdf	1025	8024	1008	8699	1004	10278
doc	938	6394	984	7374	940	7828
ppt	905	1808	911	1801	729	1859

Table 2: Query examples with corresponding target collections for pseudo-desktop collections.

Query	Target Collection
jose 03 kahan	email
presentation 19	ppt
org address	html

6.1 Pseudo-desktop Collections

Three pseudo-desktop collections described in Kim et al. [9] was used for this experiments. Each collection contained typical file types in desktop like email, webpage and office document related to three individuals, as shown in Table 1. For email, the indexed fields were *title*, *content*, *date*, *sender* and *receiver*. For other document types, *title*, *URL*, *abstract*, *date* and *text* were indexed.

For generating queries, we first chose the target document and took each query word based on the term frequency in a randomly chosen field of the document. For instance, the query 'org address' shown in Table 2 was generated by taking the term 'org' from the *URL* field and 'address' from the *title* field, respectively. Kim et al. reported that queries generated using this method have the highest similarity to a set of manual queries.

For each experiment, we generated 50 queries of average length 2 where target documents were taken from each sub-collection in proportion to the number of documents it contains. Table 2 shows several queries we used. Lastly, all the experiments were repeated three times since the query generation procedure involves some randomness.

6.1.1 Prediction Accuracy

In Table 3, we compare the accuracy of type prediction in pseudo-desktop collections for the CQL and FQL methods, where FQL shows a clear improvement over CQL method. Although this result should be interpreted with some reservations because we are using simulated queries, the same trend was found in the experiment using manual queries. We also observe that both methods show reasonable performance in the *Jack* and *Tom* collections, which contain far more email documents than other types. From this, we can conclude that both methods are relatively robust against an imbalance of sub-collection sizes.

6.1.2 Retrieval Performance

We now report the retrieval performance for the same

Table 3: Accuracy of type prediction in pseudo-desktop collections.

	Jack	Tom	Kate
CQL	0.606	0.637	0.38
FQL	0.773	0.807	0.64

Table 5: Number and average length of documents in a computer science (CS) collection.

Type	#Docs	Length
email	851	731
news article	170	352
calendar item	354	306
webpage	4727	539
office document	1887	357

queries in Table 4. The first noticeable trend is that both the choice of type-specific retrieval model and type prediction method has a big impact on the final result. Especially, Oracle type prediction was much better than the FQL method, which in turns outperformed CQL across all collections. On the other hand, the Best retrieval method was not much better than the PRM-D and PRM-S methods.

6.2 CS Collection

Next we report on experiments using a computer science (CS) collection, where the documents of various types are collected from many public sources in the Computer Science department the authors belong to. As shown in Table 5, the CS collection contained emails from the department mailing list, news articles and blog postings on technology, calendar items of department announcements, webpages and office documents crawled from the department and lab websites. The documents in the CS collection are much shorter than in the other pseudo-desktop collections.

For all document types, *title* and *content* fields were indexed. Also, there were type-specific fields such as *date*, *sender* and *receiver* for email, *tag* and *author* for news articles, *starttime* and *location* for calendar items, *URL* for webpages, *filename* for office documents.

We used the DocTrack game for collecting queries. We had 20 participants who were students, faculty members and staff in the department, all familiar with the documents in the collection. In total, 66 DocTrack games were played and 984 queries were collected using 882 target documents, some of which are shown in Table 6.

The average length of queries was 3.97, which is longer than the reported length (2) in the other desktop search studies [6]. This may be due to people paying more attention to the task in the competitive game setting compared to typical desktop search. The standard deviation (1.85) of the query length was also quite high, implying that there is a considerable variation among the querying behavior of individuals.

The participants were generally in favor of the game, saying that playing the game was fun and felt reasonably similar to their search experience in the desktop.

Since some of features required data for estimation, we used 528 queries to obtain query-log feature (QQL) values and training parameters for other features. The rest (456) of the queries were used to evaluate the type prediction performance of features and combination methods by 10-fold

Table 6: Query examples with corresponding target collections for a CS collection.

Query	Target Collection
reminder jeffrey johns	email
2010 candidate weekend	calendar item
yanlei xml dissemination	office document
cs646 homework html	html

Table 8: Significance test result for type prediction accuracy in a CS collection. Each cell shows the p-value of paired t-test between the accuracy of two methods.

Method	CQL	FQL	Grid	RankSVM	MultiSVM
CQL		0.03	0.00	0.00	0.00
FQL			0.69	0.27	0.02
Grid				0.41	0.02
RankSVM					0.07

cross-validation. For the retrieval experiments, since many queries did not return any documents, we used only queries where the relevant document was ranked in the Top 50 result set.

6.2.1 Prediction Accuracy

Table 7 summarizes the prediction accuracy result, comparing two of the best-performing single feature runs (CQL / FQL) and combination methods (Grid / RankSVM / MultiSVM). The result shows that all the combination runs improved performance over best single feature runs given by FQL, which outperformed CQL in this collection as well. MultiSVM was shown to be the most effective among combination methods. This is understandable considering that we had one target collection for each query, which is a natural setting for multi-class classification. RankSVM was slightly better than Grid yet the difference was not significant.

The result of a significance test is reported in Table 8, which shows that the performance differences between CQL and all the other methods are significant with a p-value of 0.05 and that MultiSVM outperforms all the other methods significantly with a p-value of 0.1 (using paired t-test). Overall, this means that suggested type prediction method (FQL) improves the performance of the CQL method and that the combination of features improves the performance further.

6.2.2 Retrieval Performance

Table 9 shows the retrieval performance, comparing four retrieval methods (DLM / PRM-S / PRM-D / Best) and the same set of type prediction methods as above in addition to Oracle and Uniform methods.

The result mostly shows the same trends as the pseudo-desktop collections despite the big difference in experimental conditions (note that queries were algorithmically generated for the pseudo-desktop collections). FQL was better than CQL and all the combination methods outperformed CQL and FQL significantly (with a p-value of 0.05 using paired t-test).

The only exception is that the performance of MultiSVM was slightly worse than RankSVM. Given the superior prediction accuracy of MultiSVM, it seems that the procedure of converting the SVM output into the type prediction score caused some problems. We can also see that Oracle type pre-

Table 4: Retrieval performance in three pseudo-desktop collections using different type-specific retrieval methods and type prediction methods.

	Jack				Tom				Kate				Average
	Uniform	CQL	FQL	Best	Uniform	CQL	FQL	Best	Uniform	CQL	FQL	Best	
DLM	0.129	0.159	0.27	0.331	0.104	0.123	0.192	0.224	0.126	0.12	0.237	0.294	0.194
PRM-S	0.152	0.212	0.326	0.403	0.15	0.209	0.289	0.348	0.232	0.239	0.383	0.532	0.346
PRM-D	0.148	0.219	0.335	0.403	0.155	0.204	0.289	0.346	0.25	0.245	0.387	0.538	0.355
Best	0.154	0.225	0.336	0.414	0.157	0.217	0.302	0.361	0.241	0.245	0.388	0.542	0.354
Average	0.146	0.204	0.317	0.388	0.141	0.188	0.268	0.32	0.212	0.212	0.349	0.477	

Table 7: Accuracy of type prediction for best-performing single feature runs and combination methods in a CS collection.

Method	CQL	FQL	Grid	RankSVM	MultiSVM
Accuracy	0.708	**0.743**	0.747	0.758	**0.808**

Table 11: Correlation among the prediction performance of features. Each cell shows the pearson correlation coefficient between the accuracy of two methods.

Feature	FQL	Dict	QQL	Clarity	GAVG	ReDDE
CQL	0.68	0.10	0.29	-0.03	-0.01	0.04
FQL		0.05	0.32	0.01	-0.01	0.01
Dict			0.28	0.02	-0.00	0.04
QQL				-0.02	0.04	0.05
Clarity					0.17	0.05
GAVG						0.62

Table 12: Feature weights trained using queries written by each user. The last row shows the weights trained using all queries. All the names are anonymized.

User ID	W_{CQL}	W_{FQL}	W_{Dict}	W_{QQL}	$W_{Clarity}$
Jane	1	1	0.15	1	0.09
Yao	0.38	0.56	0.15	1	0
Rajiv	0.58	1	0	0	0
Tim	0.62	1	0	1	0.18
David	1	1	0	0	0
All Users	1	1	0	1	0.09

diction method and the Best retrieval method outperform other methods, which leaves room for further improvement in both aspects.

6.2.3 Leave-one-out Prediction Accuracy

For further analysis on the effectiveness of individual features, we next report the result of single-feature and leave-one-out experiments in Table 10, where we used only one feature (single-feature) or all but one feature (leave-one-out) to measure the performance.

All in all, features with high single-feature performance had bigger impacts when they were omitted, as shown in the case of content-based features (CQL / FQL) and query log feature (QQL). On the other hand, features based on the initial retrieval result (GAVG / ReDDE) was shown to be ineffective. Presumably, this is because the documents of different types had different characteristics in this collection, which make the scores incomparable.

On the other hand, in some cases, low single-feature prediction accuracy did not necessarily translate into small difference in the leave-one-out experiment. The dictionary-based feature (Dict) and the performance prediction feature (Clarity) were shown to have a high impact on combination performance despite having the lowest single-feature results.

The result above makes more sense when considered together with the correlation among features in Table 11. Methods based on collection term statistics (CQL / FQL) and top result set (GAVG / ReDDE) have high correlations within the group, which explains why performance does not suffer much when one of high-performing features (e.g. CQL and FQL) are omitted. On the other hand, that QQL does not correlate highly with any other features explains its high impact on the leave-one-out experiment.

6.2.4 Personalization of Feature Combination

In the experiments so far we have used the queries from all participants together. Another benefit of the feature combination approach is it can adjust the result based on the querying behavior of each individual. For instance, if a user frequently includes terms that indicate a particular collection, the learning method can improve type prediction performance by assigning higher weight to the Dict or QQL features.

To explore this potential value of this personalized type prediction, we tried training feature combination methods only with queries written by each user and looked at how much variation it causes for resulting feature weights.

The result in Table 12 compares the weights trained for each user. It shows that content-based features (CQL / FQL) are not very helpful for some user (Yao), whereas query log does not make a difference for other users (Rajiv / David). Although this is a preliminary result, the result weights show considerable variation, implying that personalized training data produced different results for each user. We leave the more analysis of this benefit of personalization for future work.

7. CONCLUSION & FUTURE WORK

In this paper, we suggest a retrieval model for desktop search where type-specific retrieval results are merged into the final rank list based on type prediction scores. Using pseudo-desktop and CS collections, we demonstrated that improving the type prediction method can produce significantly better final retrieval performance.

We also introduced field-based collection query likelihood (FQL) – a new type prediction method that exploits type-specific metadata – which shows superior performance to competitive baseline methods in both collections we tested. Although FQL is used in the context of desktop search, it

Table 9: Retrieval performance in a CS collection using different type-specific retrieval methods and type prediction methods.

	Uniform	CQL	FQL	Grid	RankSVM	MultiSVM	Oracle	Average
DLM	0.343	0.507	0.53	0.552	0.563	0.556	0.674	0.526
PRM-S	0.349	0.501	0.518	0.518	0.551	0.547	0.674	0.52
PRM-D	0.36	0.518	0.536	0.536	0.567	0.564	0.694	0.537
Best	0.372	0.548	0.564	0.590	0.596	0.594	0.72	0.563
Average	0.356	0.518	0.537	0.549	0.569	0.565	0.691	

Table 10: Accuracy of type prediction for single-feature and leave-one-out experiment. Leave-one-out accuracy shows the percentage of decrease for leave-one-out experiment from the experiment using all features.

Method	CQL	FQL	Dict	QQL	Clarity	GAVG	ReDDE
Single-feature Accuracy	0.708	**0.743**	0.201	0.579	0.240	0.255	0.207
Leave-one-out Accuracy	-0.6%	-1.7%	-0.6%	**-3.1%**	-0.6%	-0.0%	-0.0%

can be used as a resource selection method for vertical or federated search, as long as each document has field structure.

Moreover, we developed a human computation game for collecting queries in a more realistic setting and used this data to train discriminative learning models that combines type prediction methods as features. Our results show that the combination method can improve type prediction performance compared to any of existing methods. We also explored the benefit of personalizing type prediction results, which may potentially improve the performance further.

Our work leaves many interesting challenges. Although this work showed the value of improving type prediction performance, better type-specific retrieval and result merging algorithms should bring further performance gains. For example, we used only textual features for sub-collection retrieval, and it should be possible to incorporate type-specific features (e.g. PageRank score for webpage).

Also, although we focused on the known-item search task in this paper, the retrieval model suggested here will be equally applicable for the ad-hoc search scenario. To gather training data for this scenario, the DocTrack game can be modified to gather ad-hoc queries using topic descriptions and corresponding sets of relevant documents.

8. ACKNOWLEDGEMENTS

This work was supported in part by the Center for Intelligent Information Retrieval and in part by NSF grant #IIS-0707801. Any opinions, findings and conclusions or recommendations expressed in this material are the authors' and do not necessarily reflect those of the sponsor.

9. REFERENCES

[1] J. Arguello, J. Callan, and F. Diaz. Classification-based resource selection. In *CIKM '09*, pages 1277–1286, New York, NY, USA, 2009. ACM.

[2] J. Arguello, F. Diaz, J. Callan, and J.-F. Crespo. Sources of evidence for vertical selection. In *SIGIR '09*, pages 315–322, New York, NY, USA, 2009. ACM.

[3] J. P. Callan, Z. Lu, and W. B. Croft. Searching distributed collections with inference networks. In *SIGIR '95*, pages 21–28, New York, NY, USA, 1995. ACM.

[4] S. Chernov, P. Serdyukov, P.-A. Chirita, G. Demartini, and W. Nejdl. Building a desktop search test-bed. In *ECIR' 07*, pages 686–690, 2007.

[5] S. Cronen-Townsend, Y. Zhou, and W. B. Croft. Predicting query performance. In *SIGIR '02*, pages 299–306, New York, NY, USA, 2002. ACM.

[6] S. Dumais, E. Cutrell, J. Cadiz, G. Jancke, R. Sarin, and D. C. Robbins. Stuff i've seen: a system for personal information retrieval and re-use. In *SIGIR '03*, pages 72–79, New York, NY, USA, 2003. ACM.

[7] D. Elsweiler and I. Ruthven. Towards task-based personal information management evaluations. In *SIGIR '07*, pages 23–30, New York, NY, USA, 2007. ACM.

[8] T. Joachims. Optimizing search engines using clickthrough data. In *KDD '02*, pages 133–142, New York, NY, USA, 2002. ACM.

[9] J. Kim and W. B. Croft. Retrieval experiments using pseudo-desktop collections. In *CIKM '09*, pages 1297–1306. ACM, 2009.

[10] J. Kim, X. Xue, and W. B. Croft. *A Probabilistic Retrieval Model for Semi-structured Data.* In Proceedings of ECIR '09. Springer, 2009.

[11] H. Ma, R. Chandrasekar, C. Quirk, and A. Gupta. Improving search engines using human computation games. In *CIKM' 09*, pages 275–284, 2009.

[12] P. Ogilvie and J. Callan. Combining document representations for known-item search. In *SIGIR '03*, pages 143–150, New York, NY, USA, 2003. ACM.

[13] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C.* Cambridge University Press, Cambridge, UK, 2nd edition, 1992.

[14] S. Robertson, H. Zaragoza, and M. Taylor. Simple bm25 extension to multiple weighted fields. In *CIKM '04*, pages 42–49, New York, NY, USA, 2004. ACM.

[15] J. Seo and W. B. Croft. Blog site search using resource selection. In *CIKM '08*, pages 1053–1062, New York, NY, USA, 2008. ACM.

[16] L. Si and J. Callan. Relevant document distribution estimation method for resource selection. In *SIGIR '03*, pages 298–305, New York, NY, USA, 2003. ACM.

[17] L. Si, R. Jin, J. Callan, and P. Ogilvie. A language modeling framework for resource selection and results merging. In *CIKM '02*, pages 391–397, New York, NY, USA, 2002. ACM.

[18] P. Thomas and D. Hawking. Server selection methods

in personal metasearch: a comparative empirical study. *Inf. Retr.*, 12(5):581–604, 2009.

[19] L. von Ahn and L. Dabbish. Designing games with a purpose. *Commun. ACM*, 51(8):58–67, 2008.