Journal of Information Processing Vol.22 No.3 425–434 (July 2014)

[DOI: 10.2197/ipsjjip.22.425]

Invited Paper

Algorithms and Techniques for Proactive Search

C. Shaun Wagner[1,a)] Sahra Sedigh Sarvestani[1,b)] Ali R. Hurson[1,c)]

Received: December 4, 2013, Accepted: February 8, 2014

Abstract: While search engines have demonstrated improvements in both speed and accuracy, their response time is prohibitively long for applications that require immediate and accurate responses to search queries. Examples include identification of multimedia resources related to the subject matter of a particular class, as it is in session. This paper begins with a survey of collaborative recommendation and prediction algorithms, each of which applies a different method to predict future search engine usage based on the past history of a search engine user. To address the shortcomings identified in existing techniques, we propose a proactive search approach that identifies resources likely to be of interest to the user without requiring a query. The approach is contingent on accurate determination of similarity, which we achieve with local alignment and output-based refinement of similarity neighborhoods. We demonstrate our proposed approach with trials on real-world search engine data. The results support our hypothesis that a majority of users exhibit search engine usage behavior that is predictable, allowing a proactive search engine to bypass the common query-response model and immediately deliver a list of resources of interest to the user.

Keywords: proactive search, neighborhoods of similarity, recommendation algorithms

1. Introduction

Growing interest in augmenting classroom instruction with multimedia resources, a more personalized approach to teaching, and demonstrated efficacy of active learning techniques have led to more dynamic teaching that seeks to facilitate active learning, where interaction with and among the students determines the focus and sequence of the subject matter presented. This adaptive approach necessitates that instructors have access to a vast pool from which they can rapidly select educational resources to supplement and enrich a predetermined lesson plan. Additional motivation for pooling and enabling rapid search of educational resources arises from the increasingly modular nature of course design and the overlap among modules in different courses. Learning artifacts developed for a module in one course may (and should) become useful in other courses, facilitating networked curricula. In brief, the efficacy of modern instructional techniques relies on the ability of the instructor to rapidly locate related learning artifacts, giving search engines an instrumental role.

The personalized learning that is facilitated by rapid identification of supplementary resources is one of many applications that can benefit from a proactive search engine. We differentiate *proactive* search from *predictive* search - the latter predicts the query likely to be entered by a user; the former predicts the actual resources of interest, without requiring a query. This paper describes our work on developing techniques for accurate and rapid proactive search.

Search engines are vital tools for the information age [3], [11], [15], [17]. Their development began as a study on indexing techniques - a topic as old as information storage itself [11]. From the early indexing of religious texts into books, chapters, and verses to modern indexing of library resources by type, category, and author, indexing has served as a natural method of facilitating easier and faster access to information resources. Early on, search engines used keywords to allow users to refer to an index without knowledge of how the resources were indexed. Providing a search query (of keywords) and receiving a list of resources has become the standard query-response model of search engines [11].

Improvement of search engines focuses on two main aspects of performance: accuracy and speed [7], [15], [24]. *Accuracy* refers to how well the search engine ties a search query to a relevant information resource. It is subjective, based on the user's concept of what is relevant and what is not. *Speed* refers to how quickly the search engine is able to return results. Speed is easily measured as a duration. We do not use the term *response time* to avoid ambiguity - the duration we measure transpires between invocation of the search engine by the user, and the selection of a result.

Our research focuses on the use of *recommendation*; i.e., proactively returning responses to queries that the user is expected to make, to improve both accuracy and speed of search engines. Successful recommendation can improve accuracy, as the resources recommended are likely to be of interest to the user. Speed is improved because proactive search eliminates the need for the user to enter a query and wait for a response - the results will be waiting for the user to review them. Proactive search capability can supplement the typical query-response functionality of a search engine - when recommendation is rendered infeasible due to lack of information; e.g., prior search history, the proactive search engine will perform exactly as it would without recommendation, without any degradation in accuracy or speed.

Recommendation has proven successful in many application

[1] Missouri University of Science and Technology, Rolla, MO 65409, USA
[a)] csw6g3@mst.edu
[b)] sedighs@mst.edu
[c)] hurson@mst.edu

domains where users need to be directed to resources. amazon.com uses recommendation to direct users to products that they are anticipated to purchase. Netflix uses recommendation to direct users to movies they are anticipated to find of interest. Pandora attempts the same task for music. These applications succeed in guiding a user to resources of interest to him or her; however, the form of recommendation they use is of limited use in applications with time constraints, as it generally ignores the context and presents a user with a "wholesale" list of items of potential interest. Recommendation classifies resources by interest, without considering the time or sequence in which the resources are likely to be of interest [2], [10]. In the classroom setting described earlier in this paper, such a "wholesale" list is too generic to be helpful in identifying resources relevant to a specific topic of discussion.

Prediction is a specialized form of recommendation that identifies resources that are likely to be of interest to a specific user in a specific (temporal) *context* [2]. Using the previous recommendation examples, prediction would identify the item that an amazon.com user is likely to purchase next Tuesday or the movie a Netflix user is likely to watch tonight. In the classroom, prediction can identify the resource needed "next" - immediately following the use of a specific sequence of other resources. Proactive search will transparently and pervasively anticipate the information for which the user is likely to search, based on the context and previous behavior of the user. The proactive approach, when feasible and successful, eliminates query entry time and hence reduces the overall search time for the user.

A proactive search engine does not need to identify the absolute time when a resource will be requested - it is enough to identify the order in which resources will be requested. The search history of the user, specifically, the resources selected from search results, creates a sequence of resources that is used to predict the resources likely to be of subsequent interest. Accuracy and speed are improved, facilitating applications such as real-time identification of educational resources as a class is in session.

The success of proactive search is entirely contingent on modeling and predicting specific user behavior; e.g., purchasing or perusal of movies or music, based on the history of the user. Modeling human behavior is an established science [13], [18]. The common model is state-based, similar to a Markov model. Users are defined as being in a discrete state that describes how the user relates to the environment. An event in the environment or an action by the user causes a change in the state. Behavior models have been successfully applied to very complex situations, such as predicting the actions of the driver of an automobile [13], [16], [21].

Many behavior models rely on identifying similarity among users - it is assumed that similar history begets similar actions in the future. A population is divided into groups of users, where the members of a group have exhibited similar behavior. These are often referred to as *neighborhoods of similarity*, or simply neighborhoods. Using neighborhoods of similarity refines behavior models by constraining the pool of users from which the data supporting the model is drawn. This in turn improves the accuracy of predictions made based on the behavioral model. The k-nearest neighbor (k-NN) algorithm captures the process of constructing a neighborhood. All of the many variations of k-NN aim to construct a neighborhood around a target user [20]. The parameter k may refer to the extent of similarity between the target user and other users in the neighborhood - anyone who is less than k similar is excluded from the neighborhood. In other implementations, k refers to the cardinality of the neighborhood.

In the interest of improving speed, this paper suggests a refinement to the k-NN technique. In classifier systems, the outcome of classification may be used to refine the classifier and improve accuracy. Proactive search relies on classification of resources (e.g., results returned in response to a search query) into two groups. One group is comprised of the one resource that the user will select; every other resource is in the second group. In refining the neighborhood, any member whose presence contributed to identifying the resource of interest to the user is kept in the neighborhood, without any further measurement of similarity. Those who failed at classifying the proper resource are automatically replaced with other users who may be more successful in the next prediction. Carrying out measurements of similarity between users can be a complex process, and as such, avoiding comparisons for members who failed to contribute to successful prediction reduces complexity and is likely to improve accuracy. This process is described as *outcome-based* refinement of the neighborhood, where the outcome in question is whether or not the recommended resource was of interest to the user.

This paper demonstrates the feasibility of proactive search, elaborating and expanding on our earlier work [22]. We describe the techniques that comprise the foundation of proactive search, and propose refinements that improve its accuracy and speed. The novelty of the contribution lies in the very concept of proactive search, which to our knowledge we are the first to propose. The refinements proposed and demonstrated are also novel. It is worth reiterating that the objective of our work is to eliminate altogether the need for entering a query, differentiating our proposal from techniques such as Google Instant Search, which predict the query, then continue to apply the query-response model.

The basis of proactive search is accurate prediction of user interests. To this end, Section 2 presents a survey of common prediction algorithms and describes their role in our proposed proactive search method. Prediction is in turn contingent on identification of similarity - the topic of Section 3, where common k-NN implementations are introduced Section 4 describes the specific hypothesis of the research presented in this paper - that proactive search succeeds in prediction of resources of interest while eliminating the need for a search query. The test methods used to confirm this hypothesis, the real-world search engine data used, and the scoring methods applied in validation of the proposed techniques are also described in Section 4. The results of an experimental case study are presented in Section 5. Section 6 concludes the paper by discussing the viability and effects of proactive search.

2. Recommendation and Prediction

The last decade has seen a rapid increase in electronic storage capacity. Connectivity among these devices has also increased, resulting in a world-wide web of interconnected networks. The

explosion of demand for information has resulted in a plethora of recommendation algorithms designed to limit the information that a user must examine when attempting to search the vast pool available.

General recommendation techniques are of limited use in prediction, as they are unable to give sufficient consideration to context. Consider an amazon.com user who has just purchased volume five of the *Harry Potter* series. Recommendation would identify volumes one through four as being of interest to this target user. They most likely are of interest. However, the target user is very likely to have read them before purchasing the fifth volume. Prediction would go beyond recommendation and identify volume 6 as the book the user will be interested in next. As such, prediction is context-aware recommendation. The context in question is the sequence in which the volumes are read.

In the amazon.com example, it is obvious that the products are the resources that the algorithm should recommend or predict. In a standard search engine, prediction could anticipate either the query that a user will type or the information resource that a user will select. The latter could be considered the ultimate objective of the former, as the response to a query is a list of results from which the user selects one (or more) resources. Google and Bing have produced algorithms for predicting what a user will type as a query. However, they still require the user to begin entering the query and attempt to automatically complete the query before it is finished or alter the query once it is submitted. However, the user is actually searching for an information resource; the query is only a means to an end. Further, a query is not a discrete index to a specific resource. As an example, a search for "hedgehog" may refer to a small spiny mammal, a popular chocolate treat, or an old anti-submarine mine. Prediction of the query is not as effective as predicting the exact resource that the user will select. This distinction is the advantage of the proposed work over methods such as Google Instant Search.

As mentioned earlier, prediction is context-aware recommendation and can leverage existing recommendation techniques. The simplest form of recommendation utilizes the *rank model*. Resources are ranked by the frequency with which they are selected. The resources selected most often, which have the highest rank, are suggested to the user. The justification of this method is Zipf's law, which suggests that the most popular items will be selected more than half of the time [26].

amazon.com applies a refinement of the rank model when suggesting resources to customers based on similarities in purchase history ("people who bought *X* also bought *Y*"). Instead of identifying the most popular resources over all users, this "also" model constrains the population considered in determination of popularity to a neighborhood of users who have a past purchase in common. The most popular resources from this smaller neighborhood are suggested. Omission of completely dissimilar users should increase the accuracy of such recommendations.

Neither version of the rank model takes time or sequence into account. As a further refinement, instead of suggesting "people who bought *X* also bought *Y*," a better suggestion would be "people who bought *X* **then** bought *Y*."

It may be possible to further improve the recommendation by considering more than one resource that the target user and the population share. The *n* resources most recently selected by the target user can be represented as an *n*-gram, defined as a contiguous sequence of *n* items selected from a (larger) sequence. In this method, any user who has the same *n*-gram in his or her search history is used in making recommendations. The most popular resources that "follow" the *n*-gram are suggested to the target user. This should be more accurate at predicting what comes next than a model that does not consider order.

Humans are not automata, and as such there is variation in their behavior - even among multiple occurrences of the same action. The *n*-gram model requires an exact match between subsets and does not allow for variance. Allowing a "fuzzy" (rather than exact) match between the target user's most recent *n*-gram and an *n*-gram in another user's history will identify similarities that would otherwise be missed. Fuzzy matching between sequences is popular in biostatistics and latent semantic indexing. Identifying similar subsets of two or more sequences is known as *local alignment*. When prediction is carried out using fuzzy matching, the most popular resources that follow the local alignment are suggested to the target user.

All of the methods described in this section - with the exception of the simple rank model - are essentially creating a neighborhood of similar users for the target user. The next section discusses methods for creating and refining these neighborhoods.

3. Neighborhoods of Similarity

The objective of the *k*-nearest neighbor (*k*-NN) algorithm is to identify neighborhoods of similarity [5], [6], [25]. The algorithm begins with a target user and identifies other users to place in the target user's neighborhood. In some models, the *k* most similar users are included in the neighborhood. In other models, users that are at least *k* similar are included in the neighborhood. Both models require definition of a *measure of similarity*.

Measuring similarity between electronic resources is not straightforward, however, the algorithms described in the remainder of this section attempt to quantify similarity (or difference) among the attributes of electronic entities. Some algorithms treat each entity as a vector, where each attribute is represented by a dimension in the vector space [19]. Two parallel vectors that point in the same direction (regardless of magnitude) are considered identical. When vectors point in opposite directions, they are considered opposite. The angle between two vectors becomes the measure of similarity. Often, the cosine of the angle is used, producing -1 for opposite and 1 for identical vectors.

If all of the attributes are binary, the *Jaccard similarity coefficient* is computed as a set comparison between two entities [8]. The more attributes the two entities have in common, the more similar they are. This is very similar to Hamming distance, which identifies the number of positions in which two binary strings differ. If each position of a binary string is considered an attribute, the Jaccard and Hamming measures may be used interchangeably to measure either similarity or difference.

Sequence plays an important role in prediction. Neither vector nor set comparison algorithms require attributes to maintain a specific order. Consider comparing two users based on gender,

age, and nationality. No difference results from changing the order of attributes from {*gender, age, nationality*} to {*age, gender, nationality*}. To preserve (and enable consideration of) sequence, the attributes must be stored as ordered sequences of values, and the comparison algorithm must be sensitive to the order.

Ordered sequences of values are often referred to as *strings*. String-based comparison measures are order-sensitive extensions of Hamming distance, which counts the total number of positions in which two strings differ without considering the order of the differing bits. If the first two bits of two binary strings, respectively, are swapped; the Hamming distance between them will not change. An order-sensitive string-based comparison algorithm should produce a different result when the order of values is changed.

Levenshtein distance, which is the foundation of most string-based comparison algorithms [1], [4], [12], identifies the number of changes required to turn one string of values into another string of values. It does not provide a specific solution for calculating minimal distance; i.e., aligning the two strings such that the difference between their respective elements is minimized. The latter is achieved by the Wagner-Fischer (often misattributed to Levenshtein) matrix solution [23]. Similar to Wagner-Fischer, there exist other matrix solutions that can align one string to another to produce a global alignment (Needleman-Wunsch) or local alignment (Smith-Waterman) with minimum difference.

In determining similarity between two search engine users, it is important to identify the purpose of the comparison. Our objective is proactive search, which requires that we model the behavior of selecting from among search results and predict the resource that will be selected. Therefore, the comparison should be between the search histories of the two users. Consideration of demographic attributes such as age or gender is unlikely to increase the accuracy of prediction, as these attributes only matter to the extent of their effect on search behavior. If an older male physician in Japan performs searches similar to a young female engineer in Norway, it would be imprudent to ignore this similarity.

Search histories are ordered sequences of resources, implying that string-based comparison is necessary. One method of detecting similarity is to simply count the resources selected by both users. The greater the number of resource selections that two users have in common, regardless of order, the more similar they can be considered.

Considering the order of selection should improve the measure of similarity. To this end, the target user's history is represented as an n-gram and compared to n-grams representing the history of other users. Users whose respective histories have long ordered sequences in common are considered similar - the greater the length of the common sequences, the more similar the users. While this should produce a better neighborhood than a simple test of common resources, it is prohibitively complex in practice. The best algorithm for identifying the longest common substring between two strings of data is complex - $O(n^2)$.

As described in Section 2, human behavior has natural variance. If inclusion of a given user in a neighborhood is contingent on an exact match of n-grams within the search history, users whose behavior is similar, but not identical to that of the target user will be excluded from the neighborhood. Allowing for "fuzzy" matching in local alignment will address this shortcoming without increasing the complexity of the matching algorithm over the exact case.

In this paper, we compare three forms of similarity measurement in the k-NN method of building a neighborhood of similarity around a target user. All are limited to the n most recent resource selections by the target user. The first implementation identifies users who also selected the n resources most recently selected by the target user. The second implementation treats the last n resource selections as an n-gram and identifies users who also have that n-gram in their history. The third implementation also treats the last n resource selections as an n-gram, but performs a local alignment of that n-gram with other user's histories and measures the extent of the alignment.

As all three implementations are based on k-NN, we propose to utilize outcome-based refinement to improve the accuracy of prediction. After the similarity neighborhood is produced, the search history of each member of the neighborhood is used for prediction - each member will produce a suggestion (or a set of suggestions, based on the specific implementation). In outcome-based neighborhood refinement, the suggestion from each member of the neighborhood is compared to the resource actually selected by the user. Only those members whose suggestion matches the selected resource are kept in the neighborhood. Other members are replaced with a different user. In theory, outcome-based k-NN should produce a viable neighborhood of similarity by simply replacing anyone who fails at prediction with an arbitrarily chosen user. Eventually, the neighborhood will contain only users who repeatedly make correct predictions. Therefore, checking the outcome of the prediction made by each member should help improve accuracy, while reducing the complexity of the overall algorithm by eliminating fruitless similarity comparisons.

4. Empirical Validation

This research is based on the hypothesis that given the history of resource selections from a population of search engine users, it is possible to predict a majority of their future selections. If this hypothesis is correct, it implies that a search engine can proactively identify the information resource of interest to a user - even before the user performs a query. The success of proactive search is contingent upon access to search histories, and as such, each user will need to provide a number of queries and make a number of selections to create a history.

4.1 Prediction and Neighborhood Algorithms

The previous section presented a survey of recommendation and similarity comparison algorithms. The six most common algorithms, enumerated below, were selected and implemented to test our research hypothesis and validate our proactive search technique. Testing multiple algorithms makes data available to measure how much of an improvement - if any - is achieved by using more complex algorithms, as compared to simpler algorithms. We sought to test as many algorithms as possible, as quickly as possible. One of our data sets, N has 23,168,232 records. If the

slowest algorithm took one second to perform a trial on a single record, a complete test of set *N* would require nine months of continual processing.

When used with a neighborhood, a prediction algorithm considers only the members of the neighborhood. Otherwise, the algorithm uses the entire population of users for prediction, at the cost of considerably greater runtime. Let *s* refer to the number of resources returned by the algorithm and *n* to the number of selections in the most recent history of the target user. In our tests, $s = 10$ and $n = 5$.

The prediction algorithms, respectively, operate as follows:

(1) *Random*: Suggests *s* arbitrarily selected resources. It is used to provide a worst-case baseline, as no intelligence or history is used in making predictions.
(2) *Popular*: Suggests the *s* most frequently selected resources.
(3) *Also*: Suggests the *s* most frequently selected resources from users whose last *n* resources selected are the same as those of the target user. Selection order is not considered. This algorithm becomes redundant with neighborhood models that use the selection history to construct the neighborhood.
(4) *Next*: Adds a constraint to the *Also* algorithm - the resources returned must have been selected immediately (in sequence) after the resource in common between the users.
(5) *n-Gram*: Suggests the *s* most frequently selected resources, with an exact ordered match to the last *n* resources selected by the target user.
(6) *Fuzzy*: Suggests the *s* most frequently selected resources, with a fuzzy ordered match to the last *n* resources selected by the target user. Local alignment is used as the fuzzy matching algorithm.

Multiple neighborhood models were tested, corresponding to the multiple methods of measuring similarity. The techniques used in identifying similar users are the same as those used to identify resources to suggest. The algorithms used in neighborhood construction are:

(1) *No-Neighborhood*: As a baseline, this algorithm constructs no neighborhood, forcing prediction to use the entire population.
(2) *Common*: Selects users who selected the *s* resources most recently selected by the target user. The order of selection is not considered.
(3) *n-Gram*: Selects users who selected, in the same order, the target user's *s* most recently selected resources.
(4) *Fuzzy*: Selects users who have a local alignment with the target user's *s* most recently selected resources.

Outcome-based refinement is carried out for all the of neighborhood algorithms, with the obvious exception of *No-Neighborhood*.

4.2 Data Sets

While it is possible to theorize how well recommendation or prediction may perform, tests on records from real search engines demonstrate performance in a practical setting. Numerous data sets, from different search engines, were investigated. The six sets detailed below and described in **Table 1** were chosen to illustrate the performance of the algorithm on data sets that vary in size and attributes.

Table 1 Basic statistics of data sets used in validation.

Set	Records	Users (*U*)	Resources (*R*)	*U*/*R*	*R*/*U*
A	114,494	18,526	57,018	0.32	3.08
E	168,387	12,857	10,458	1.23	0.81
L	1,000,209	68,404	3,708	18.36	0.05
M	3,168	45	255	0.18	5.67
N	23,168,232	463,616	17,755	26.11	0.04
Y	30,655	3,121	27,910	0.11	8.94
V	3,518,498	381,276	1,651,071	0.23	4.33
W	1,257,942	153,857	977,891	0.16	5.31
X	1,025,907	213,608	375,916	0.57	1.76

Set *A* comes from AOL. In 2006, AOL Research released a data set of three months of Internet search history. Users were deidentified. The attributes of interest to our validation are the user, query time, and resource selection. We do not consider the actual queries supplied by the users.

Set *E* comes from Every Busy Woman, an online catalog of women-friendly businesses. The log files from the website's search engine record the user making the query, the time of the query, and the business listing selected from the query.

Set *L* comes from MovieLens. As a recommender system, MovieLens is not truly a search engine. However, the use of data set *L* to predict the movie that a user will watch made one aspect of the MovieLens data set interesting. The data set includes the user, the rating the user gave to a movie, the movie, and the date(s) on which it was watched. From this set, we derived the user, timestamp, and movie selection.

Set *M* comes from the Medical University of South Carolina. It is comprised of log files from a search engine for employee training resources. Due to its small size, it is used primarily for very quick code checking before the larger data sets are tested.

Set *N* comes from Netflix. Most Netflix data sets are designed to predict the rating that a user will give a movie - an attribute irrelevant to our prediction. The data set we utilized included history without ratings, but containing the user, movie, and the time the movie was watched.

Set *Y* comes from Yandex, the largest search engine based in Russia. Unlike US-based search engines, Yandex regularly releases deidentified data sets for research. Similar to data set *A*, this set provides a record of real Internet searches performed by real people.

Three data sets used in validation, but not detailed in this paper, came from the popular search engines AltaVista, AllTheWeb, and Excite. These are identified as *V*, *W*, and *X* in Table 1. All three of these are website search engine logs and were produced to demonstrate a variety of queries, providing very little longitudinal data for prediction. Tests on these three sets performed better than set *A*, another website search engine log file. Overall performance was very similar to set *L* (from MovieLens).

All data sources were normalized such that each user and each resource, respectively, was identified by a unique integer. The selection time of each resource was converted to a timestamp indicating the number of seconds since 1970-01-01 00:00:00. Each database was stored in the format {*user, selection_time, resource_selected, previous_selection_time*}. The previous selection

time was necessary for linking sequences of resource selections.

When the same user selected the same resource more than once in succession, it was assumed that reload or refresh of the resource had taken place, not a true selection of a new resource. As such, successive selections of the same resource were flagged as repeats and ignored. A selection sequence of {1, 2, 2, 3} would then become {1, 2, 3}. However, a selection sequence of {1, 2, 3, 2} would not be altered, because the two selections of resource 2 are separated by selection of a different resource. Most sets had no repeats.

On rare occasion, a user selected more than one resource within one second. When sorting resource selections by selection time, allowing for more than one resource selection in the same second could cause errors, as a second was considered the atomic unit of time. If the query made on the database of selection records was limited to one resource selection per timestamp, the additional selections were omitted. If the query was simply sorted on selection time, it was impossible to determine the order in which each of several concurrent selections was made. As a solution to this problem, the data sets were stretched to increase the amount of time between each selection. After stretching, unused seconds between each selection allowed the extra selections to be shifted into the future without overlapping with existing resource selections. The final order of the resource selections was maintained, while ensuring that each selection made by a user was given a unique timestamp.

Users with only a single resource selection were ignored in testing, as their history was insufficient for use in recommendation or prediction. Further, the first two resource selections of each user are ignored, as prediction requires a history of multiple selections. Testing begins on the third resource selection of each user.

As shown in Table 1, the data sets resulting from the aforementioned pre-processing tasks are radically different. They vary considerably in size, number of resources selections recorded for each user, and the number of users for whom selection of a given resource is recorded.

4.2.1 Distribution

Zipf's law states that given a large and structured set of texts in a given (natural) language, the frequency of occurrence of any word is inversely proportional to its rank in the frequency table [26]; e.g., the most frequently occurring word will appear twice as often as the second most frequently occurring word. We expect the same property to hold for resource selections recorded in the data sets used for validation - significant departure from the Zipfian (power-law) distribution could be indicative of manipulation of the data.

For each data set, the users-per-resource and resources-per-user frequencies, respectively, were checked against the Zipfian distribution. The results are shown in **Table 2**, normalized such that 0% is a perfect $1/r$ distribution and 100% is one standard deviation away from the expected power-law value. Set *A* appears to be manipulated such that there was a cap on the maximum number of records allowed per user, which matches the description of how the data was generated. Set *N* has nearly the same number of users for each resource, which matches the description of how the data was generated. Set *L* exhibits fixed user/resource and resource/user frequency, indicating that it has been significantly manipulated and is an unnatural representation of search queries. Data that has been manipulated is unlikely to lead to acceptable predictions, indicating that proactive search based on sets *A*, *L*, and *N* is likely to have poor results.

Table 2 Deviation of data from Zipfian distribution. 0% = $1/r$ distribution. 100% = one standard deviation from $1/r$ distribution.

Set	User Deviation	Resource Deviation
A	109%	48%
E	88%	15%
L	163%	215%
M	54%	63%
N	39%	193%
Y	93%	5%

Table 3 Order of the data sets.

Set	Records	Sequences	Order
A	114,494	14,847	0.13
E	168,387	189,111	1.12
L	1,000,209	995,253	1.00
M	3,168	2,484	0.78
N	23,168,232	12,004,826	0.52
Y	30,655	1,539	0.05

4.2.2 Order

In a large population of search engine users, prediction must be dependent on the existence of common sequences of resource selections. If so, it is possible to replace the common sequences with a single resource identifier that indicates the sequence was selected. This is similar to file compression, where a common sequence of bits is replaced with a much shorter identifier.

Andrey Kolmogorov, one of the founders of algorithmic complexity theory, described this form of compression as a measure of *entropy* [9]. The greater the number of common sequences, the lower the complexity of the data. A lack of common sequences indicates entropy. In this research, lack of entropy manifests as *order*, indicating that there is a common order to the selection of search results.

Within each data set, common sequences of five resource selections were identified. For two data sets of the same size, a greater number of common sequences indicates entropy, as there is no single common sequence of resource selections. For two data sets with the same number of sequences, a larger data set indicates more order as there are more instances of the common sequences being selected. Therefore, order is measured as the total number of records divided by the number of unique sequences of five resource selections. **Table 3** shows the order of the data sets. Sets *E* and *M* have higher order, indicating that prediction should work well on those sets. Set *Y* has nearly no order. With nearly complete entropy, prediction is unlikely to succeed for set *Y*.

4.2.3 Convergence

A combination of distribution and order may be referred to as *convergence*. If a data set is naturally distributed and there is order, the sequences of data selections should converge on a subset of the total resources in the data set.

Measuring convergence begins with identifying the distribu-

tion of selections. The resources selected are ranked from most- to least-selected. A best-fit logarithmic trendline is calculated to match the frequency counts. It has the form $f(x) = a\ln(x) + c$, where a is the natural logarithmic coefficient and c is a constant. The trendlines of the data sets cannot be compared directly, due to the radical differences in population sizes. Dividing a by c will produce a measure of convergence, where a value of 0 indicates a uniform distribution. The greater the magnitude of the result, the more convergent the resource selections. Therefore, convergence is measured as $|a/c|$.

Table 4 shows the convergence of the respective data sets. Sets A, E, and M exhibit the greatest convergence, indicating that prediction should be successful. Set Y has nearly no convergence, indicating the opposite.

4.3 Quantifying Success of Prediction

In testing algorithms and data sets, the method of scoring the tests must demonstrate how effective the algorithms are in predicting the resource that the target user will select. In its simplest form, the score is the percentage of tests in which the algorithm produces a suggestion that is the resource the target user selects. We use the term *success score*, rather than *success rate*, to emphasize that our measure of success is not a simple percentage. The proactive search algorithm returns a list of suggestions, similar to what the user would receive from a conventional search engine in response to a query. In the proactively-generated list, the resources are ordered from most to least likely to be selected. In determining the success score, we use weights to differentiate between cases where the user selects resources from different positions on this list. In other words, our success score is higher if we correctly predicted the resource to be selected - the resource that is first on the list of suggestions. If the user selects a resource that is lower on the list, we still have a successful prediction, but the resulting success score will be lower.

In recommendation systems, where the objective is to find items of interest to the user, weighted scoring uses a measure of interest in determining the success score. For prediction systems, interest is binary - the user is assumed to be "interested" in the one resource that is selected and uninterested in others. Therefore, the rank, r, of each suggestion in the returned list could be selected as the corresponding weight for scoring. For each trial (prediction) of the proactive search algorithm, we add $1/r$ to the success count, where r is the rank of the resource selected in the suggestion list. The item most likely to be selected appears at the top of the list. If this item is selected, $r = 1$ and the success count will be incremented by one full unit. If the selection is in the tenth rank, $r = 10$, and only $1/10$ will be added to the success count.

Table 4 Convergence of the data sets.

Set	Ln Coef (a)	Constant (c)	Convergence ($\|a/c\|$)
A	−16,901.2	75,762.5	0.22
E	−10,278.8	48,873.6	0.21
L	−3,724.6	28,258.4	0.13
M	−477.3	1,974.8	0.24
N	−141,674.8	900,504.4	0.16
Y	−19,842.5	842,548.4	0.02

The final success score is the total success count divided by the number of trials.

In our data sets, the respective selection histories recorded for different users vary in length. As such, some users will affect the overall success score more than others - they will contribute a greater number of trials. To determine the resulting skew, the average success score was calculated for each user and compared to the overall success score. If it was significantly greater, the user was considered to heavily skew the outcome.

The duration of a complete test of each algorithm over each complete data set (one prediction for each query submitted) is also recorded, and the average number of trials per second and the average number of seconds per trial are calculated. The amount of time that the trials require is important in justifying the added complexity for prediction algorithms that exhibit a higher success score.

5. Discussion of Results

Recall that our proactive search involves two tasks: i) identifying users similar to the target user (whose search query is being examined) and constructing a neighborhood, and ii) predicting the resource to be selected by the user based on similarities between the history of the user and the respective histories of its neighbors. In this section, we discuss the results of empirical trials carried out to validate these respective tasks. Prediction is discussed first, as the proactive search was initially validated without constructing neighborhoods. Every test was scored (in weighted and unweighted form), using a suggestion limit of ten resources.

5.1 Results of Prediction Trials

The first set of trials involved execution of the proactive search algorithm, without constructing neighborhoods of similar users, using the each of the algorithms enumerated in Section 4.1 for predicting the resource selected. The results, shown in **Table 5**, were not far from expectations. The results for set M are considerably different from that of every other data set. We attribute this difference to the small size of M - approximately 3,200 records and an order of magnitude smaller than any other data set. In general, as the algorithms increase in sophistication, progressive improvement is seen in the success score. The *n-Gram* algorithm is the exception to this rule. This algorithm is very restrictive, as

Table 5 Unweighted (weighted) success scores of prediction algorithms.

Set	Random	Popular	Also
A	1% (0%)	3% (0%)	4% (1%)
E	10% (2%)	30% (5%)	54% (7%)
L	0% (0%)	12% (2%)	14% (2%)
M	56% (12%)	77% (10%)	73% (9%)
N	0% (0%)	1% (0%)	5% (1%)
Y	0% (0%)	1% (0%)	3% (1%)
Set	Next	n-Gram	Fuzzy
A	8% (3%)	1% (1%)	10% (7%)
E	58% (6%)	51% (28%)	59% (42%)
L	20% (5%)	19% (12%)	27% (16%)
M	75% (7%)	39% (30%)	45% (30%)
N	9% (3%)	6% (4%)	14% (8%)
Y	4% (1%)	0% (0%)	3% (2%)

it requires an exact match on n consecutive resource selections ($n = 5$ in this test). For sets *A* and *Y*, it was very rare for two users to have five identical, consecutive resource selections. Allowing for minor differences in the resource selections, as is done in fuzzy matching, increases accuracy. In these trials, a fuzzy match was declared if was restricted for cases where the edit distance (difference) between local alignments was at most half the length of the search history being compared. For the maximal length of five, a difference of two was accepted. With a target history of only three selections, the local alignments could have an edit distance of at most one.

The unweighted results show that there is usually little difference in success score between *Next* and *Fuzzy*. From weighted results, it is clear that *Fuzzy* places the resource actually selected near the top of the list of predicted resources, while *Next* places it near the bottom; i.e., *Fuzzy* is better at predicting the resource selected. Therefore, it is expected that if the number of resources suggested are reduced by a factor of two, *Fuzzy* would retain nearly the same success score, but *Next* would drop drastically.

5.2 Results of Neighborhood Trials

Each of the neighborhood algorithms enumerated in Section 4.1 was tested separately, with a neighborhood size of twenty users, using the *Next* prediction algorithm - chosen due to the fact that it is not similar to any of the neighborhood algorithms being tested. As shown in **Table 6**, the lowest and highest success scores, respectively, are achieved with the *No-Neighborhood* and *Fuzzy* algorithms. Once again, sets *A* and *Y* did not exhibit notable improvement (over the baseline) with the *n-Gram* algorithm, because of the rarity of users who share exact sequences of n selected resources.

Outcome-based refinement was subsequently applied to the proactive search algorithm. Every neighborhood algorithm was tested with this refinement, again using the *Next* algorithm for prediction. The success score invariably improved, as may be seen by comparing Table 6 (results without outcome-based refinement) to **Table 7** (results with outcome-based refinement).

A second trial of outcome-based refinement was performed, where the neighborhoods were formed by arbitrarily selecting members of the population of users. As before, any user who did not contribute to success of the prediction was replaced, but this time with an arbitrarily selected user - who could have been ejected from the neighborhood in an earlier trial. The success score attained in initial trials was very low, but consistent across data sets and invariably improved with time.

5.3 Timing Results

Optimization of the algorithms, such as partitioning user populations on popular resources and sequences selected, made it possible to perform many trials per second on a single processor. As expected, the number of trials per second reduces as the algorithm complexity increases; however, anomalies were noted. We attribute these anomalies to specific features of the data sets, rather than the test method. For example, set *M* was small enough that the database engine would automatically store the results of all complex queries as temporary tables. As a result, identifying the most popular resources or a set of users with a matching n-gram became abnormally fast - queries were executed on small temporary tables rather than the main database, artificially increasing the number of trials per second.

Overall, identifying the most popular resources was the fastest task, with 4 to 26 trials per second, depending on the size of the data set. *n-Gram* trials were slowest, with 0.5 to 8 trials per second. The *Fuzzy* algorithm, with 1 to 9 trials per second, was not significantly faster than the n-Gram algorithm, which appears counterintuitive. With a strict limitation of five resources from the target user's history, the fuzzy comparison was implemented as a regular expression, allowing for a linear implementation of fuzzy string matching rather than a polynomial implementation.

Using outcome-based refinement yielded measurable improvement for every neighborhood algorithm tested. The extent of improvement increased with complexity of the neighborhood algorithm - the *n-Gram* and *Fuzzy* algorithms showed the greatest improvement. However, the reduction in runtime was not enough to justify using outcome-based refinement to this end - increasing the accuracy of prediction is the primary motivation for use of this technique.

Table 6 Unweighted (weighted) success scores of neighborhood algorithms.

Set	No-Neighborhood	Common	*n*-Gram	Fuzzy
A	8% (3%)	5% (2%)	9% (7%)	10% (7%)
E	58% (6%)	55% (14%)	58% (28%)	59% (42%)
L	20% (5%)	13% (6%)	19% (12%)	27% (16%)
M	75% (7%)	81% (80%)	39% (30%)	45% (30%)
N	9% (3%)	8% (2%)	11% (5%)	16% (8%)
Y	4% (1%)	4% (2%)	0% (0%)	6% (4%)

Table 7 Unweighted (weighted) success scores of refined neighborhood algorithms.

Set	No-Neighborhood	Common	*n*-Gram	Fuzzy
A	8% (3%)	12% (6%)	14% (10%)	15% (12%)
E	58% (6%)	61% (27%)	69% (41%)	72% (51%)
L	20% (5%)	25% (14%)	31% (17%)	32% (20%)
M	75% (7%)	82% (80%)	52% (41%)	57% (41%)
N	9% (3%)	17% (10%)	21% (13%)	25% (16%)
Y	4% (1%)	6% (4%)	4% (1%)	11% (9%)

5.4 Correlation

It would be useful to identify a measurable attribute of a data set that is strongly correlated with the success score of prediction. This would enable elimination of data sets for which prediction is unlikely to succeed.

The respective correlation of success score and each of the three measures described in Section 4.2 was calculated. Distribution was found to have the weakest correlation, −0.22. Order and convergence, respectively, were determined to have correlations of 0.84 and 0.60 with the success score. Convergence is a measure of both distribution and order, and as such it is reasonable to assume that convergence alone may be used to identify data sets for which prediction is (un)likely to succeed.

6. Conclusion

In this paper, we have proposed proactive search, which eliminates the need for query entry by suggesting resources based on the search history of the user and similar users. Success of this technique is contingent on correctly identifying similarity among users. To this end, we described several algorithms that can be used to construct neighborhoods of similarity and demonstrated that prediction based on these neighborhoods is likely to identify resources of interest to a given user. The most sophisticated algorithm presented is based on fuzzy matching of ordered sequences of resources selected by different users. The proposed proactive search technique was experimentally validated using several real-world search engine logs.

Proactive search can considerably accelerate the search process. In this technique, the user accesses the search engine, is presented with a list of resources, and selects a resource from this list. There is no query, query analysis, index searching, or resource list building - all of which are tasks required in using a conventional search engine.

What cannot be determined with the methods in this paper is the effect of showing users a list of resources before they perform a query; i.e., the extent to which a user will be biased towards selection of the resources suggested. Such bias has been documented for usage of recommendation systems [14]. The likely result of this bias is an improved success rate for prediction. If a greater number of users select a suggested resource, it will be suggested more often, increasing the likelihood that an even greater number of users will select it.

In addition to improving performance, implementation of a proactive search engine creates a fundamental change in how information resources are perceived. Currently, each resource is a single entity. Instead of predicting and suggesting single resources, a proactive search engine can predict and recommend sequences of resources. As such, it is not necessary that each resource be viewed a single entity. They will function just as well as modular resources designed to fit into a sequence.

Returning to the motivation for this research, viewing educational resources as sequences of modular resources fits well with the overall goal of supporting a networked curriculum. Ideally, an instructor will have two options. In the first option, the instructor searches for a few important topics in a course. The search engine identifies common sequences that include those topics and presents the instructor with a sequence of resources. In the second option, the instructor searches for multimedia resources to support instruction. The search engine will identify the sequence of resources being selected and predict the resources likely to be useful in the immediate future. In the latter case, an instructor does not have to spend classroom time searching for multimedia. The resources will be proactively identified and waiting on the instructor.

References

[1] Bertsekas, D.P.: *Dynamic Programming and Optimal Control*, Athena Scientific, 3rd edition (2007).

[2] Cleger-Tamayo, S., Fernández-Luna, J.M., Huete, J.F., Pérez-Vázquez, R. and Cano, J.C.R.: A Proposal for News Recommendation Based on Clustering Techniques, *Trends in Applied Intelligent Systems*, Lecture Notes in Computer Science, Vol.6098, pp.478–487, Springer Berlin/Heidelberg (2010).

[3] Cohen, A.M., Adams, C.E., Davis, J.M., Yu, C., Yu, P.S., Meng, W., Duggan, L., McDonagh, M. and Smalheiser, N.R.: Evidence-Based Medicine, the Essential Role of Systematic Reviews, and the Need for Automated Text Mining Tools, *Proc. 1st ACM International Health Informatics Symposium* (*IHI*), pp.376–380, ACM (2010).

[4] Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C.: *Introduction to Algorithms*, MIT Press, 3rd edition (2009).

[5] Cover, T.M. and Hart, P.E.: Nearest Neighbor Pattern Classification, *IEEE Trans. Information Theory*, Vol.13, No.1, pp.21–27 (1967).

[6] Dasarathy, B.V.: *Nearest neighbor* (*NN*) *norms: NN pattern classification techniques*, IEEE Computer Society Press, Los Alamitos (1991).

[7] Hawking, D., Craswell, N., Brailey, P. and Griffihs, K.: Measuring Search Engine Quality, *Information Retrieval*, Vol.4, No.1, pp.33–59 (2001).

[8] Jaccard, P.: Étude comparative de la distribution florale dans une portion des Alpes et des Jura, *Bulletin del la Société Vaudoise des Sciences Naturelles*, Vol.37, pp.547–579 (1901).

[9] Kolmogorov, A.N.: On Tables of Random Numbers, *Theoretical Computer Science*, Vol.207, pp.387–395 (1963).

[10] Konstan, J.A. and Riedl, J.: Recommender systems: From algorithms to user experience, *User Modeling and User-Adapted Interaction*, Vol.22, pp.101–123 (online), DOI: 10.1007/s11257-011-9112-x (2012).

[11] Levene, M.: *An Introduction to Search Engines and Web Navigation*, John Wiley & Sons, 2nd edition (2010).

[12] Levenshtein, V.I.: Binary Codes Capable of Correcting Deletions, Insertions, and Reversals, *Soviet Physics Doklady*, Vol.10, pp.707–710 (1966).

[13] Lewandowski, D.: Search Engine User Behaviour: How Can Users Be Guided to Quality Content?, *Information Services and Use*, Vol.28, No.3-4, pp.261–268 (2008).

[14] Linden, G., Smith, B. and York, J.: amazon.com Recommendations: Item-to-Item Collaborative Filtering, *IEEE Internet Computing*, Vol.7, No.1, pp.76–80 (2003).

[15] Mostafa, J.: Seeking Better Web Searches, *Scientific American*, Vol.292, No.2, pp.66–73 (2005).

[16] Pentland, A. and Liu, A.: Modeling and prediction of human behavior, *Neural Computation*, Vol.11, No.1, pp.229–242 (1999).

[17] Rangaswamy, A., Giles, C.L. and Seres, S.: A Strategic Perspective on Search Engines: Thought Candies for Practitioners and Researchers, *Journal of Interactive Marketing*, Vol.23, No.1, pp.49–60 (2009).

[18] Skinner, B.F.: *The Behavior of Organisms*, Copley Publishing Group (1938).

[19] Tan, P.-N., Steinbach, M. and Kumar, V.: *Introduction to Data Mining*, Pearson Addison-Wesley, 1st edition (2005).

[20] Tellez, E.S., Chavez, E. and Navarro, G.: Succinct nearest neighbor search, *Proc. 4th International Conference on Similarity Search and Applications*, pp.33–40 (2011).

[21] Toledo, T. and Katz, R.: State Dependence in Lane-Changing Models, *Transportation Research Record: Journal of the Transportation Research Board*, Vol.2124, pp.81–88 (2009).

[22] Wagner, C.S., Sedigh, S. and Hurson, A.R.: Accurate and Efficient Search Prediction Using Fuzzy Matching and Outcome Feedback, *Similarity Search and Applications*, Brisaboa, N., Pedreira, O. and Zezula, P. (Eds.), Lecture Notes in Computer Science, Vol.8199, pp.219–232, Springer Berlin Heidelberg (2013).

[23] Wagner, R.A. and Fischer, M.J.: The String-to-String Correction Problem, *J. ACM*, Vol.21, No.1, pp.168–173 (1974).

[24] Wickelgren, W.A.: Speed-Accuracy Tradeoff and Information Pro-

cessing Dynamics, *Acta Psychologica*, Vol.41, No.1, pp.67–85 (1977).
[25] Xiong, L., Xiang, Y., Zhang, Q. and Lin, L.: A Novel Nearest Neighborhood Algorithm for Recommender Systems, *Proc. 3rd Global Congress on Intelligent Systems* (*GCIS*), pp.156–159 (online), DOI: 10.1109/GCIS.2012.58 (2012).
[26] Zipf, G.K.: The Psycho-Biology of Language, *Language*, Vol.12, pp.196–210 (1936).

C. Shaun Wagner served in the US Marine Corps prior to receiving his B.S. degree in Computer Science with honors from the College of Charleston in 2003. He is currently completing his Ph.D. thesis in Computer Science at the Missouri University of Science and Technology while serving as Director of IT for the Outpatient Quality Improvement Network at the Medical University of South Carolina. His research interests span the computing and medical disciplines, with focus on search prediction and hypertension, respectively.

Sahra Sedigh Sarvestani received her B.S.E.E. degree from Sharif University of Technology in 1995, and her M.S.E.E. and Ph.D. degrees from Purdue University, in 1998 and 2003, respectively. She subsequently joined the Missouri University of Science and Technology, where she is currently an Associate Professor of Electrical and Computer Engineering. Her research centers on development and modeling of dependable networks and systems, with focus on critical infrastructure. She held a Purdue Research Foundation Fellowship from 1996 to 2000, and is a member of HKN, IEEE, and ACM.

Ali R. Hurson received his B.S. degree in Physics from the University of Tehran in 1970, M.S. degree in Computer Science from the University of Iowa in 1978, and Ph.D. from the University of Central Florida in 1980. He was a Professor of Computer Science at the Pennsylvania State University until 2008, when he joined the Missouri University of Science and Technology. He has published over 300 technical papers in areas including multidatabases, global information sharing and processing, computer architecture and cache memory, and mobile and pervasive computing. He serves as an ACM distinguished speaker, area editor of the CSI Journal of Computer Science and Engineering, and Co-Editor-in-Chief of Advances in Computers. He is a member of IEEE.