MISSOURI S&T
Library and
Learning Resources

Scholars' Mine

Doctoral Dissertations

Student Theses and Dissertations

2014

Proactive search: Using outcome-based dynamic nearest-neighbor recommendation algorithms to improve search engine efficacy

Christopher Shaun Wagner

Follow this and additional works at: https://scholarsmine.mst.edu/doctoral_dissertations

Part of the Computer Sciences Commons, and the Geographic Information Sciences Commons
Department: Computer Science

Recommended Citation

PROACTIVE SEARCH: USING OUTCOME-BASED DYNAMIC

NEAREST-NEIGHBOR RECOMMENDATION ALGORITHMS TO IMPROVE

SEARCH ENGINE EFFICACY

by

CHRISTOPHER SHAUN WAGNER

A DISSERTATION

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

2014

Approved by

Dr. Ali Hurson, Co-advisor
Dr. Sahra Sedigh, Co-advisor
Dr. Jennifer Leopold
Dr. Wei Jiang
Dr. Donald Wunsch

ABSTRACT

The explosion of readily available electronic information has changed the focus of data processing from data generation to data discovery. The prevalent use of search engines has generated extensive research into improving the speed and accuracy of searches. The goal of this research is to predict user behavior as a means to proactively improve speed and accuracy of search engines. The proactive approach eliminates query entry time, improving speed. Assuming success, the user locates an electronic resource of interest, improving accuracy.

Algorithms that have been shown to predict many different aspects of user behavior exist in literature. Two common approaches are used in such prediction: statistical techniques and collaborative actions. This research extends the scope of proactive search by using search histories of users in building a predictive model. The proposed approach was compared to statistical and collaborative behavior models. The test results verified that search engine prediction is a viable approach and supports the intuitive notion that prediction is more successful when user behavior exhibits less entropy.

The benefits of the proposed approach go beyond improvement in performance and accuracy. As a result of working with search histories as sequences of resources, it is possible to predict a series of resources that a user will likely select in the immediate future. This makes it possible for search engines to return resource sequences instead of simple resources. Working with sequences allows the search engine user to more effectively locate information of interest. In the end, a proactive search engine improves speed and accuracy through prediction and sequencing of electronic resources.

ACKNOWLEDGMENTS

The completion of my dissertation has been a long journey, far longer than planned. I would not have completed this work without the tireless assistance and faith of my advisors, Dr. Sahra Sedigh and Dr. Ali Hurson. Without Dr. Hurson's guidance, I would not have found my initial path into converting information resources into sequences of information modules. Without Dr. Sedigh's in-depth help, I would not have followed the path from sequencing resources to prediction of user behavior.

I am grateful that the Computer Science department at Missouri University of Science & Technology allowed me a position as a graduate instructor during my time as a graduate student. It is well known that I am completing my PhD at this time with the plan to become a professor. The experiences that I had during my two years as an instructor will be invaluable towards my future.

I must acknowledge Dr. Brent Egan of the Care Coordination Institute. I have worked for Dr. Egan for more than ten years. During that time, he has pushed me to further my education and made great accommodations, including allowing me to maintain full-time employment while working 1,000 miles away. Without his help, it would not have been financially possible to complete this work while raising two children.

Finally, I must also thank my wife for putting up with endless hours of explaining to our children that daddy is "busy", taking them to the park, to a museum, or off to grandma's house for the weekend. She has been a great woman. It has been said that behind every great man there is a great woman. I hope the converse holds true.

TABLE OF CONTENTS

SECTION

LIST OF ILLUSTRATIONS

LIST OF TABLES

LIST OF ABBREVIATIONS

$\mathcal{O}$ Asymptotic runtime complexity.

k-**NN** k nearest neighbors.

SVD singular value decomposition.

GLOSSARY

n-**gram** An ordered sequence of n items.

polysemy The existence of more than one attributes or meanings for a single name or label, e.g., "count" could refer to many attributes for a user.

prediction A specialized form of recommendation that is sensitive to time or order.

recommendation Proactively suggesting an electronic resource that a user is assumed to find interesting.

search engine A system that allows users to locate an electronic resource by means of entering a query..

state-space model A class of probabilistic graphical models that describe the probabilistic dependence between latent state variables and observed measurements..

synonymy The existence of more than one name or label that refers to a single attribute or meaning, e.g., both gender and sex usually refer to the existence of a Y chromosome in a user.

1. INTRODUCTION

1.1. MOTIVATION

This research began with the motivation to treat school courses, normally one semester long, as a sequence of modules. The modularity of the class information is intended to make it easy to develop, share, and improve electronic resources. Many modules would be shared between classes that otherwise seem dissimilar. Modular topics supported by electronic resources would require development of electronic resources. The key to modular classroom information is the ability to locate a specific information resource at the exact instant that it is needed. A proactive tool that can predict which information resource will be used before it is requested would satisfy such a need.

The benefit of a proactive search engine is not limited to the classroom. The information age has inundated the world with vast repositories of electronic resources. While the extent of information available to users is unprecedented, identifying and locating a specific resource becomes a gargantuan task for which search engines are a necessary tool. As a result, we have become dependent on search engines [1].

Search engine research focuses on speed and accuracy [1, 2, 3]. Speed is an objective measure of the time that elapses between a user accessing the search engine and the user selecting a search result. Accuracy is a subjective measure of how well a given search results matches the query supplied by the user. A proactive search engine will improve both speed and accuracy. Because the search results are proactively displayed to the user before the user enters a search query, the time spent entering a query and waiting for search results is bypassed. The total time between accessing the search engine and selecting a search result is reduced. Assuming success, the user

will have selected a search result proactively displayed. Doing so implies that the search result would match the query that the user would have entered.

This research focuses on the use of recommendation (proactively displaying an electronic resource to a user) as a means of implementing a proactive search engine. Recommendation has proven successful in many application domains. Amazon 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 for music. However, recommendation is of limited use in applications with time constraints as it generally ignores temporal context and presents a user with a "wholesale" list of items of potential interest [4, 5, 6].

Prediction is a specialized form of recommendation that recommends which resource a user will want in a particular context [7]. Using the previous recommendation examples, prediction would identify which item an Amazon user will want to purchase next Tuesday, which movie a Netflix user will want to watch tonight, or which song a Pandora user will want to listen to directly after the current song that is playing.

A proactive search engine does not need to identify the exact time that a resource will be requested. It is enough to identify the order in which resources will be requested. Given a user's search engine history, the resources that predictably come next are recommended. Speed and accuracy are improved.

1.2. OBJECTIVES

The motivation of efficiently organizing and placing multimedia resources in a classroom environment grew beyond simply modularizing information and indexing resources by module. To make resources available when they are required, a proactive

search engine is necessary. The following areas of research were necessary to develop a viable proactive search engine design:

1. **Recommendation Systems**

   With the ability to examine a user's profile, search engines have taken on the role of proactive recommendation tools. As a foundation for studying prediction algorithms, existing recommendation systems must be well understood. In general, recommendation systems function by comparing users to one another and/or items to one another. The heart of a recommendation algorithm is in the comparison algorithm.

2. **Comparison Algorithms**

   Recommendation is built on comparison algorithms. Just as there are many types of items to compare, there are many types of comparison algorithms available to use. A proactive search engine must be sensitive to the order in which electronic resources are selected. A comparison algorithm that is sensitive to order is necessary.

3. **Grouping Algorithms**

   In recommendation, it is common to limit the collaborative process to a small group of items with high similarity to a target item. Instead of comparing an Amazon user to every Amazon user, a small neighborhood of highly similar users are used. The k nearest neighbors (k-NN) algorithm is the basis for limiting recommendation to a specialized neighborhood.

4. **Prediction Algorithms**

   Algorithms to predict user behavior have been studied and proven [7, 8, 9]. A survey of existing algorithms is necessary to identify the tasks necessary to leverage prediction for a proactive search engine.

5. **Reducing Prediction Runtime**

   Existing prediction algorithms have a runtime complexity that is at least $\mathcal{O}(n^2)$. It is necessary to identify and test methods to reduce the runtime complexity of prediction.

6. **Implementation and Testing**

   With a proactive search engine algorithm firmly in place, the algorithm must be tested. If possible, it must be tested using real-world data. The hypothesis that prediction will be an improvement over general recommendation must be tested through comparison of common recommendation algorithms to the proactive search engine algorithm.

These objectives bring this research full circle. The original problem involved treating class topics as modules of information that would be available in the classroom in the order that the information is commonly taught. A general proactive search engine treats electronic resources as ordered search results, predicting which result a user will select immediately after the user's most recently selected result. Therefore, a proactive search engine will meet the requirements of the original research problem.

1.3. CONTRIBUTIONS

This research has contributed the following:

1. Analysis of recommendation algorithms, as advanced in the literature, reveals several shortcomings for predicting search engine usage, which are detailed in Section 2. This research proposes a general methodology for proactive search in an attempt to overcome these shortcomings. In a nutshell, this research proposes use of historical search result selection information, stored for each user, as an ordered model of user behavior. A target user is compared to other users to identify the resources that follows the target user's recent search history. The

resources located in the previous step are suggested to the user in a proactive manner.

2. In addition to the proposed proactive search engine algorithm, several optimization techniques are described. With the implementation of the optimization techniques, the runtime complexity of the proposed algorithm falls from $\mathcal{O}(n^3)$ to nearly $\mathcal{O}(n)$. Further, the proposed proactive search algorithm is redefined as a background agent process that may be used when a user is not active. If the user is not waiting for a result from the algorithm, runtime complexity does not have as much of an impact on the user's experience.

3. A simulation has been developed and tested with real-world search engine data. The simulation allowed the comparison of many common recommendation algorithms to the proposed proactive search engine algorithm.

4. The simulation results validated the proposal that a proactive search engine is capable of predicting which resource a user will select based solely on the user's history of selected resources. Further speculation into the impact of a proactive search engine leads to both positive and negative results. The sequencing of electronic resources will redefine how users view resources. The web is currently viewed as a collection of web pages. When sequenced, the web will be a collection of sequences of web pages. As sequences, it will not be necessary for each independent page to be a complete consumable item. It will be preferred for a page to be a modular part of a common sequence. Sequencing is positive, but a proactive search engine will likely lead to issues of swarming. Users will tend to select resources that are shown to them. Those resources will then be more popular and displayed more often. Being displayed more often will lead to users selecting them more often. The variety of immediately available resources will shrink.

1.4. ORGANIZATION

This work is divided into five sections. Section 2 covers the background research required to develop the proactive search algorithm: modeling user behavior and comparison algorithms required for both k-NN and prediction algorithms. singular value decomposition (SVD) is briefly introduced. Vector, set, and string-based comparison algorithms are compared. With behavior and similarity algorithms in place, common methods of recommendation are described.

Section 3 details the proposed proactive search algorithm. It is derived from the recommendation algorithms described in Section 2. Issues with complexity are defined. Many methods are proposed to combat runtime complexity.

Section 4 covers a detailed testing method to compare the proposed proactive search algorithm to common recommendation algorithms. The data sets used for testing are examined. The test methods are described. The results of the testing are analyzed.

Section 5 envisions a full implementation of a proactive search engine. The interface is shown, demonstrating a negligible impact on current search engine interfaces. The effects of working with sequences are discussed.

The final section concludes this work. The major contributions of this research are reiterated. Speculation about the impact of a proactive search engine is provided, leading to further research.

2. BACKGROUND

2.1. INTRODUCTION

The proposed proactive search engine algorithm is built on user behavior modeling, recommendation algorithms, string similarity and alignment algorithms, and outcome feedback methods. This section provides a survey of each topic. Section 2.2 covers common methods used to model user behavior, which leads into a description of similarity neighborhoods in Section 2.3 and a list and description of multiple recommendation algorithms in Section 2.5. Section 2.4 begins with a brief introduction to singular value decomposition (SVD) and then walks through common similarity algorithms from vector cosine to local string alignment. Section 2.6 describes how outcome feedback may be used in the development of a neighborhood.

2.2. MODELING USER BEHAVIOR

Both modeling and prediction of human behavior are established fields that gained popularity as psychology developed alongside early computers [10]. While the possibilities of human behavior appear to be infinite, the actual behavior of humans is limited by task goals and environment. Therefore, it is possible to describe human behavior as a sequence of dynamic states, which can be captured by a state-space model, such as a Markov chain, which represents user behavior with a set of interconnected nodes [11]. Each node is a time-ordered action or observation. The weight of the directed link connecting two nodes represents the probability of transitioning from the first to the second; i.e., that the action denoted by the destination node will immediately follow that of the source node. What differentiates Markov models from

other state-space models are their memoryless feature - the next state only depends on the current state; i.e., the current state subsumes the entire history of transitions.

Figure 2.1 is a model of user behavior at a vending machine. Many of the actions and observations are omitted, leaving only the most common actions and observations. User actions are in circles. User observations are in squares. Weighted arrows designate a transition from one state (be it an action or observation) to another. The weights (percentages) are the important part of the behavior model, as they determine the likelihood that the user will follow the transition. For example, 5% of the time, a person would press the coin return directly after inserting change. After pressing a product button, the user will, 100% of the time, either observe that an item is received or nothing happens. The percentage of time that each of these observations occurs is the vending machine's behavior and is omitted from the user behavior model.

With this model, a computer can monitor user behavior and predict what the user's upcoming actions and observations will be. If a user inserts change, there is a 95% chance that the user will press a button to select an item. More complex predictions can be made, such as estimation of the probability that a user will press the coin return. After inserting change, there is a 5% chance that a user will press the coin return. If the user presses a button to select an item, there is still a chance that the user will press the coin return, which is based on the chance that "nothing happens" will be observed by the user. Because "nothing happens" is an observation and not a decision for the user, the computer can monitor the vending machine to accurately know the probability of nothing happening. Assume that it is 10%, and that the vending machine behavior is independent of that of the user - a reasonable assumption. Users who select an item will observe nothing happening 10% of the time and 85% of those users will press the coin return. Therefore, 8.5% of users who select an item will eventually press the coin return. The overall chance of the coin

return being pressed is 13.5%. Further, there is a possibility that those who press the coin return will observe that nothing happens. Of those who press the coin return and observe nothing happening, there is an 85% chance that they will press the coin return a second time. This form of Markov modeling has been successfully tested
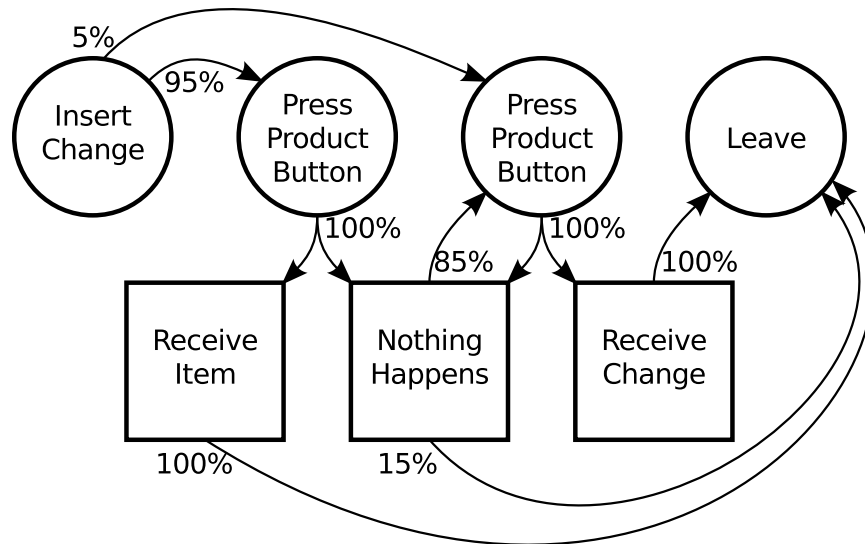
Figure 2.1: A model of typical user behavior at a vending machine.

for observation and prediction of complex human behavior. Toledo and Katz used a similar model to represent lane change behavior by automobile drivers [8]. After defining their model, lane change behavior was found to be accurately predictable. While it is rare for drivers to change lanes in the exact same order or at the exact same location, the overall behavior was predicted by observing a specific driver's actions and utilizing the Markov model that was developed by observing many other drivers.

Similarly, Pentland and Liu used Markov models to define general actions performed by automobile drivers [9]. They increased the accuracy of predictions by producing multiple Markov models. Drivers that exhibited similar behavior were clustered into similarity groups or neighborhoods. (Common methods used to construct

similarity neighborhoods are described in Section 2.3.) A separate model was developed for each neighborhood. The models contained many common attributes, but were different enough to clearly identify deviations in behavior between each similarity neighborhood. New drivers were observed without prediction to produce a short history of driving behavior. That history was used to place the driver in one of the similarity neighborhoods. Then, the model for that neighborhood was used to predict the driver's behavior. The resulting predictions proved to be 95% accurate.

2.3. GROUPING USERS INTO NEIGHBORHOODS OF SIMILARITY

When discussing prediction of user behavior, a common example is Amazon's product recommendation algorithm. Customers recognize it as the "Customers Who Bought This Item Also Bought" feature. It is a popular and somewhat effective neighborhood model for collaborative filtering [4]. The goal is to identify objects by specific attributes and then use those attributes to cluster or group those objects by similarity. Each cluster is commonly referred to as a neighborhood. For Amazon, the customer's attributes are a set of products each customer purchased. Regardless of the similarity of the products purchased by a particular customer, customers who have purchased a large number of the same products are considered similar. For search engines, metrics for search engine usage already exist. Google has patented many of their measurements of search result relationships, such as keyword identification, hand ranking, geospatial relationships, and number of inbound links [12]. Following Amazon's model, search engine users who have selected a large number of the same search results are considered similar and should be grouped into neighborhoods of similarity in developing a predictive search engine.

The k nearest neighbors (k-NN) algorithm is commonly used to group objects into neighborhoods of similarity [5, 13, 14]. Objects are characterized by a set of predefined simple attributes - often a small fraction of the attributes that could

potentially characterize an object. The choice of attributes to include in this sub-set greatly affects the usefulness of the resulting neighborhood model. Objects with similar attributes are grouped together. Once grouped, it is assumed that objects within the same neighborhood will share all attributes, including those not used in developing the neighborhood model.

As an example, the Piggly Wiggly grocery store may create neighborhoods of similarity based on the time of day a customer is most likely to make a purchase, the average amount of each purchase, and the specific store at which the customer makes a purchase. From there, a neighborhood of morning shoppers who make purchases over \$200 per trip to a beach-side store may be identified as a neighborhood. With three simple attributes in common, Piggly Wiggly assumes that other attributes are shared. If a portion of customers in that neighborhood suddenly purchase a specific product, Piggly Wiggly can target marketing for the product to everyone in that specific group instead of the general population. Obviously, Piggly Wiggly can use a more complex algorithm for grouping customers into neighborhoods, but the concept remains the same [15].

Regardless of application, the k-NN algorithm is generalized into three simple steps, detailed in algorithm 2.1. In these steps, a concept of distance is often used instead of similarity. Distance is a measure based on the attributes of two objects. The distance between two identical objects is zero. The larger the distance between two objects, the less similar the objects are. The term "distance" is derived from vector distance. Assuming that the attributes for an object are treated as a vector, the distance between the attribute vectors of two objects is the distance between the objects themselves. Common methods for measuring distance are discussed in Section 2.4. An "object" may be any entity that will be modeled. For the purpose of collaborative filtering in a user-product environment, some models cluster the users

---

Algorithm 2.1 The k-NN algorithm.

$k \leftarrow$ the number of objects in the neighborhood
$U \leftarrow$ all objects
$t \leftarrow$ target object
$N \leftarrow$ initially empty neighborhood
for all $u \in U$ **do**
  if $u = t$ **then**
    continue
  end if
  $s \leftarrow$ similarity between u and t
  Add u to N with a score of s
end for
Sort N from greatest s to lowest s
Remove all but greatest k members of N

---

together, while others cluster the products. Amazon.com is an example of a successful collaborative filtering environment in which the users are compared to one another based on purchased trends, as in "Customers Who Bought This Item Also Bought" feature [4]. Pandora.com is an example of a successful collaborative filtering environment in which the products - songs in this case, are matched by similarity across many metrics identified by the Music Genome Project [16]. Users who like one song are offered songs within the same neighborhood of similarity. Each approach (clustering users, vs. products) has its own merits [17]. Users with similar attributes will likely behave in a similar manner. Products with similar attributes will likely be purchased (or perused) in a similar manner. It is also possible to have a complex cluster model that compares both users and products.

In implementation, many variations of k-NN exist [13, 14, 15, 18, 19, 20, 21]. By definition, k refers to the number of objects in the neighborhood, the k most similar objects. The target of similarity may change from implementation to implementation. It may be the k objects that are most similar to a target object. It may be the k objects

that are most similar to each other. In a rather radical change, some implementations consider k to be a limit of difference. The neighborhood is a collection of objects that are at least k similar to the target object. In these models, increasing k may or may not alter the number of objects in the neighborhood as seen in Figure 2.2. While
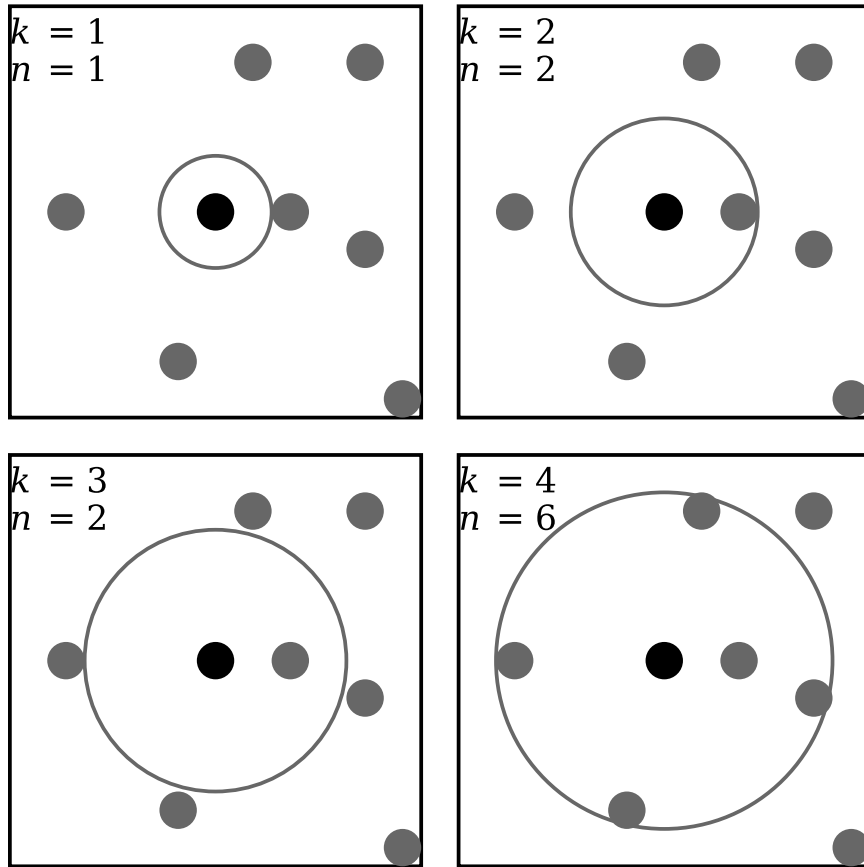
Figure 2.2: Effect of increasing the neighborhood radius (k) on the neighborhood cardinality (n).

these implementations of k-NN are very different from one another, they are all based on the concept of similarity. However, similarity is not a well-defined term. Just as there are many k-NN algorithms, there are many similarity algorithms. Section 2.4 introduces a variety of similarity algorithms.

2.4. SIMILARITY ALGORITHMS

This section is divided into two sets of subsections. The first two cover singular value decomposition (SVD) for use in identifying similarity and grouping similar items. The remaining sections cover vector-based algorithms, set-based algorithms, and string-based algorithms. Singular value decomposition (SVD) is unrelated to the similarity metrics sections, but is included because it is commonly used in recommendation [22, 23]. The last three sections are a walk from vector cosine, the most common similarity metric, to local alignment, the similarity metric used in the proposed proactive search engine algorithm.

2.4.1. Singular Value Decomposition. It is often necessary to define a relationship between two sets of objects, such as customers and products. One method of doing so is to group the objects into respective neighborhoods of similarity and then compare and contrast the various neighborhoods. The k-NN algorithm is used to create a single neighborhood for a single object, not a set of neighborhoods for all objects. Further, the k-NN algorithm is not capable of handling missing attributes - a common problem in real-world data.

Grouping and comparing objects is subject to several challenges, beyond missing attributes. There are issues of synonymy and polysemy. Synonymy occurs when two identical attributes have different names. Polysemy occurs when a single name refers to multiple attributes. To correct for missing data (sparsity), synonymy, and polysemy; Singular value decomposition (SVD) has been widely used as part of latent semantic indexing [24]. Decreasing missing data, synonymy, and polysemy with SVD in turn increases the accuracy of grouping by similarity [19].

The purpose of SVD is to decompose a matrix M into three matrices that represent its rows, columns, and the relationship between the rows and columns, respectively. Specifically, SVD will convert an $m \times n$ matrix M into a collection of three matrices: an $m \times m$ unitary matrix U that describes the rows of M, an $n \times n$

unitary matrix V that describes the columns of M, and an $m \times n$ diagonal matrix Σ that describes the relationship between the rows and columns of M [25]. In practice, a thin form of SVD is implemented, because it produces the same estimation with fewer calculations and values to store [6]. A thin SVD calculates only n columns of U and n rows of Σ. The following example computes a thin SVD. With V^T denoting the conjugate transpose of V, the SVD of matrix M is defined as in equation 2.1.

$$M \approx U \Sigma V^T \qquad (2.1)$$

Relating this to users, assume each of three users (X, Y, and Z) is characterized by four attributes (a, b, c, and d). Matrix M, in Figure 2.3, contains the attribute values for each user. Eigenvalues for each attribute over all three users produces the attribute column, U. The user row V^T is produced by taking the eigenvalues of the four attributes for each user. Before entering values into U and V^T, the values are normalized. The standard for doing so is to divide each value in a set by the square root of the sum of the square of each value in the set. Σ is the scaling factor used for U multiplied by the scaling factor for V^T. Multiplying $U \times \Sigma \times V^T$ produces an estimate

	M				U		Σ		V^T					M'		
	X	Y	Z		Att.Avg				User Avg					X	Y	Z
							Scale		X	Y	Z					
a	.41	.40	.32		-.21								a	.30	-.09	.27
b	1	-1	1	$\approx$	-.81	$\times$	1.98	$\times$	-.72	.22	-.66	$=$	b	1.15	-.35	1.06
c	.21	.24	.13		-.09								c	.13	-.04	.12
d	.95	.50	.75		-.54								d	.77	-.24	.71

Figure 2.3: Producing the first SVD values for U, Σ, and V^T.

matrix M'. While the estimated matrix, M', in Figure 2.3 is not exactly the same as the original matrix, M, the relationships between the objects are maintained. X and

Y are negatively related. X and Z are positively related. To correct for the error in the estimated matrix, the residual difference between the original and estimated matrices is used to calculate a new set of averages (U and V^T) and another scaling factor (Σ). The new matrices are shown in Figure 2.4. The original and new matrices

	$M - M'$				U'		Σ'		$V^{T\prime}$		
	X	Y	Z		Att.Avg				User Avg		
							Scale		X	Y	Z
a	.11	.49	.05		.43						
b	-.15	-.65	-.06	$\approx$	-.58	$\times$	1.17	$\times$	.23	.97	.07
c	.08	.28	.01		.25						
d	.18	.74	.04		.65						

Figure 2.4: A new U', Σ', and $V^{T\prime}$ produced from the difference between M and M'.

(from figures 2.3 and 2.4, respectively) are concatenated to produce two columns as U and two rows as V^T. The new scaling factor is placed diagonally in a new Σ. Repeating the multiplication, a new matrix M'' is produced. Comparing Figure 2.5 to Figure 2.3 illustrates that M'' is a significantly better than M' as an estimate of M. The difference between this estimated matrix M'' and the original matrix M is

U			Σ			V^T				M''		
Att.Avg						User Avg				X	Y	Z
			Scale			X	Y	Z				
-.21	.43								a	.42	.39	.31
-.81	-.58	$\times$	1.98	0	$\times$	-.72	.22	-.66	b =	1.00	-1.01	1.01
-.09	.25		0	1.17		.23	.97	.07	c	.20	.24	.14
-.54	.65								d	.95	.50	.76

Figure 2.5: M'' produced from a more complete U, Σ, and V^T.

used to create another matrix of residual values, which in turn are used to create another column of attribute averages in U, another scaling factor in Σ, and another row of user averages in V^T. The result is shown in Figure 2.6. Figure 2.6 depicts

	$M - M''$					U				Σ				V^T	
	X	Y	Z			Att.Avg								User Avg	
										Scale				X	Y	Z
a	-.01	.02	.01			-.21	.43	.69						-.72	.22	-.66
b	0	.01	-.01	$\approx$		-.81	-.58	.02	$\times$	1.98	0	0	$\times$	.23	.97	.07
c	.01	0	-.01			-.09	.25	-.70		0	1.17	0		-.65	.10	.75
d	0	0	-.01			-.54	.65	-.17		0	0	.02				

Figure 2.6: U, Σ, and V^T are completed by repeating the decomposition method on $M - M''$.

the SVD for the original matrix M. Attribute averages are represented by the U matrix. User averages are represented by the V^T matrix. Scale is represented by the Σ matrix. Multiplied together, $U\Sigma V^T = M$. Further, the U and V^T matrices are not required to identify the relationships between the users and attributes. The Σ matrix reflects composite information about the relationships between users and attributes. The U and V^T matrices contain information about specific attributes and users, not about relationships across the two sets. Since SVD is intended to store relationship information, only the diagonal values of the Σ matrix are required. For this example, from Figure 2.6, the user-attribute relationship of M is represented by the vector $\{1.98, 1.17, 0.02\}$.

Once the SVD for existing data is calculated, it is possible to predict missing attributes for users. Assume a new user, T, is introduced. Only the first three attributes, a, b, and c, are known for this user. Using these three attributes, the user average column for T is calculated to be $\{0.27, 0.72, 0.19\}$. The value of the missing attribute, d, for T is estimated in Figure 2.7. By estimating missing attributes for

$$
\begin{array}{ccccccc}
 & & \Sigma & & T & & \\
d & & & & & & T.d \\
\begin{array}{|c|c|c|}\hline -.54 & .65 & -.17 \\\hline\end{array} & \times & \begin{array}{|c|c|c|}\hline 1.98 & 0 & 0 \\\hline 0 & 1.17 & 0 \\\hline 0 & 0 & .02 \\\hline\end{array} & \times & \begin{array}{|c|}\hline .27 \\\hline .72 \\\hline .19 \\\hline\end{array} & = & \begin{array}{|c|}\hline .26 \\\hline\end{array}
\end{array}
$$

Figure 2.7: Using Σ to estimate d for a new user T.

users, it is possible to maintain accurate similarity measures among all users. Further, SVD is not affected by synonymy or polysemy. The k-NN algorithm compares each attribute separately. Synonymy and polysemy artificially alter the weight of attributes. SVD produces a relationship value, Σ, from all attributes for all users at the same time. Having a value repeated or two values combined in the attributes will result in U, Σ, and V^T matrices that produce the original matrix M with the same repeated or combined attributes.

2.4.2. Grouping with Singular Value Decomposition. Singular value decomposition (SVD) is commonly used for handling data synonymy, polysemy, and sparsity. Less commonly used is another benefit of SVD, the ability to perform efficient and reliable similarity clustering [17, 26]. If the original matrix, M, is a mapping of customers and products, the matrices U and V^T describe the customers and products with normalized values. Consider the example customer matrix V (transposed from V^T) for eight customers depicted in Figure 2.8. First, each positive value is replaced with a 1. Each negative value is replaced with a 0. To make the result easier to read, the ones and zeros are read as binary numbers, each of which is converted to a decimal number (110 becomes 6). Objects with the same decimal number are in the same group. Customers A, D, and H are in the same neighborhood of similarity. Customers C, F, and G are in another neighborhood. If desired, the U matrix (produced from the original customer-product matrix M (see Figure 2.6) could be used to easily group the products into neighborhoods of similarity. The benefit

S	.52	.19	.78		S	1	1	1		S	7
T	.14	.25	-.32		T	1	1	0		T	6
U	.48	-.34	.19		U	1	0	1		U	5
V	.49	.37	.88	$\rightarrow$	V	1	1	1	$\rightarrow$	V	7
W	.95	.18	-.18		W	1	1	0		W	6
X	.11	-.38	.48		X	1	0	1		X	5
Y	.56	-.65	.84		Y	1	0	1		Y	5
Z	.78	.74	.54		Z	1	1	1		Z	7

Figure 2.8: Producing groups of similarity for eight objects using SVD.

of having all objects grouped into neighborhoods of similarity with one function is obvious, but it comes at a cost. SVD is a complex and time-consuming function. It does not allow for limiting the size of neighborhoods. Within a neighborhood, it does not indicate which objects are more or less similar to one another. When speed, size limitation, the neighborhood of a single object, or comparative similarity is important, using the k-NN algorithm is preferred. Further, the SVD does not define what it means to be similar as it places customers into neighborhoods of similarity.

2.4.3. Vector-Based Similarity Metrics. The concept of "similarity" is very vague and often subjective. A proper metric of similarity must produce a stable and comparable result. Further, an algorithm that depends on similarity must define what attributes are being compared. Then, it is possible to state that a set of attributes for one user or resource have a specific measure of similarity to another set of attributes for another user or resource.

Within the realm of search engines, the definition of a user may vary. Some search engines store user information such as name, date of birth, and gender. To be universal, the only attributes that every search engine must possess is the logs of search engine usage per user. Therefore, each user is defined as an ordered set of resource selections. A resource selection is a tuple containing user, time, resource.

For simplicity, assume that every resource may be represented as a single letter. A user is represented as an ordered set of resources, such as K, L, A, T, U. As an ordered string, the set is simply "KLATU." The exact time of each selection is ignored, but order is maintained. Having each user identified by an ordered string requires a similarity metric that measures the similarity between two ordered strings.

The following survey of similarity algorithms purposely steps through many different similarity algorithms. Vector cosine, the dominant similarity metric, is the starting point. Each step is clearly identified as an improvement over the previous method, with the purpose of measuring similarity between ordered strings. Given
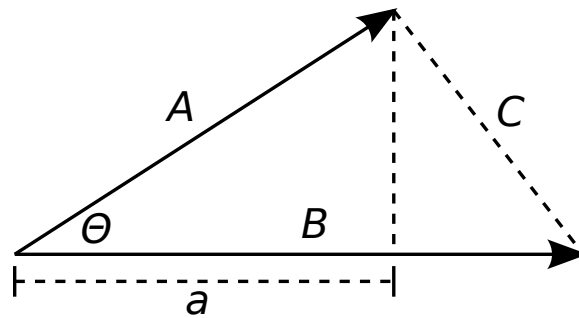
Figure 2.9: Two vectors, A and B, used to visually describe vector cosine and Tanimoto distance.

two object attribute vectors, such as A and B in Figure 2.9; vector cosine defines similarity as the cosine of the angle, Θ, between them. Equation 2.2 calculates vector cosine using "dot product" and "magnitude" vector operations. The result will be -1 when the Θ is 180°, 0 when Θ is 90°, and 1 when Θ is 0°. Therefore, it is possible to infer that -1 means A is opposite of B while 1 means that A is the same as B.

$$S(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \, \|B\|} \tag{2.2}$$

While vector cosine is easy to visually represent with two dimensions, it scales easily to many dimensions. The complexity of the dot product and magnitude calculations increases linearly as the number of dimensions increases. Further, the result is the same regardless of the order of the arguments, as it is merely a measure of the angle between the vectors.

While the magnitude of the vectors are used to normalize the result of vector cosine between -1 and 1, the difference of the magnitudes of A and B are not considered as part of the similarity. In reference to Figure 2.9, the cosine of Θ will be the same if the magnitude of A is halved or doubled. To improve on vector cosine, it is possible to use the magnitude of vector C, a vector that connects vectors A and B, in the equation.

$$||C||^2 = ||A||^2 + ||B||^2 - 2||A||||B|| \cos \Theta \tag{2.3}$$

Using the law of cosines, the magnitude of vector C is defined in Equation 2.3. To simplify, $A \cdot B = ||A||||B|| \cos \Theta$. Therefore, $||C||^2 = ||A||^2 + ||B||^2 - 2A \cdot B$. Dividing $A \cdot B$ by $||C||^2$ instead of $||A||||B||$ will result in a similarity value that mainly calculates the cosine of Θ, but increases the value of the denominator as $||C||$, the distance between A and B, increases. If the 2 is omitted, the Tanimoto difference equation [27] is formed, shown as Equation 2.4. When Θ is less than 90°, Tanimoto provides a measure of similarity that combines both the cosine of Θ and the relative magnitudes of A and B.

$$\mathrm{T}(A, B) = \frac{A \cdot B}{||A||^2 + ||B||^2 - A \cdot B} \tag{2.4}$$

2.4.4. Set-Based Similarity Metrics. Many attributes - such as the manufacturer of an automobile - are categorical. The *manufacturer* may have the value "Ford," "Toyota," or "Audi." These categorical values cannot be used in the calculation of either vector cosine or Tanimoto distance. A common solution is to assign an arbitrary index values to each categorical value, e.g. 1="Ford," 2="Toyota," and

3="Audi." As such, the values of 1, 2, and 3 may be used to calculate similarity. However, the use of index values implies a relationship between the categorical values. In this case, it implies that a Ford is twice as similar to a Toyota as it is to an Audi because the distance between 1 and 2 is half that of the distance between 1 and 3. Further, it implies that, from the reference of a Toyota, a Ford is the opposite of an Audi. These implications invalidate the similarity measures of categorical attributes.

A common solution is to use a separate binary attribute for each manufacturer. An automobile will have attributes of "Ford," "Toyota," and "Audi." A Mustang will have Ford=1, Toyota=0, and Audi=0. A Corolla will set Toyota=1. By separating each categorical attribute into a set of binary attributes, the implied similarity between the categories is removed. The result is more a comparison of sets of categorical values rather than vectors. Therefore, a set-based similarity metric is better suited to comparisons of categorical attributes.

Jaccard similarity is a common set-based similarity metric, defined in Equation 2.5 as the intersection of two sets divided by the union of the two sets [28]. This results in a much faster operation than vector cosine or Tanimoto distance (equations 2.2 and 2.4). With n attributes, vector cosine will require about $3n$ multiplications and additions, along with two square root calculations. The Jaccard similarity coefficient is a counting function that can be performed with $3n$ additions.

$$\mathrm{J}(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.5}$$

Jaccard difference is the opposite of Jaccard similarity, such that the Jaccard distance plus the Jaccard similarity of two sets equals 1. Of note, Jaccard difference produces the same result as Tanimoto difference for binary sets [27]. This provides a clear bridge between the use of vector-based metrics and set-based metrics.

Used for identification of flowers, the original Jaccard similarity algorithm was based on three counts [28]:

M_{11} is the count of attributes in which both A and B have a 1.

M_{01} is the count of attributes in which A has a 0 and B has a 1.

M_{10} is the count of attributes in which A has a 1 and B has a 0.

Not used, M_{00} is the count of attributes in which both A and B have a 0.

Using these counts, Jaccard similarity may be calculated as $M_{11}/(M_{11} + M_{01} + M_{10})$. Jaccard distance is then $(M_{01} + M_{10})/(M_{11} + M_{01} + M_{10})$. Over time, use of Jaccard similarity or difference has been generalized into many implementations that use M_{11} in the numerator, but other counts in the denominator, such as $M_{01} + M_{10}$ or the entire number of attributes.

The Sørensen-Dice coefficient [29] is an example of a set-based metric that uses the complete count of attributes in the denominator. Equation 2.6 shows that the numerator is multiplied by 2 and the complete count of attributes in sets A and B are summed in the denominator. M_{11} is still the intersection measure in the numerator. If the attributes represented in A and B are the same, $|A| = |B|$, then the denominator is $2|A|$. The 2 in the numerator cancels the 2 in the denominator, making this Jaccard similarity with the inclusion of M_{00} in the denominator. Unlike Jaccard similarity, the separation of $|A|$ and $|B|$ allows for comparison of two sets with a different number of attributes.

$$\mathrm{D}(A, B) = \frac{2|A \cap B|}{|A| + |B|} \tag{2.6}$$

While the Sørensen-Dice coefficient uses the complete count of attributes in the denominator, Hamming distance omits the denominator completely [30]. Hamming distance is simply $M_{01} + M_{10}$, i.e. the numerator of Jaccard distance. As with Jaccard similarity and distance, Hamming distance requires both sets to have the same

number of attributes. It does not lend itself to varying sets of data in the same way that the Sørensen-Dice coefficient does.

2.4.5. String-Based Similarity Metrics. To summarize, vector-based similarity metrics are very popular. Set-based similarity metrics handle categorical attributes. However, Vector-based and set-based similarity metrics are not order sensitive [31]. Order is important in many forms of recommendation. Consider the following scenario.

The *Harry Potter* books are a series of seven books. After purchasing book 5 in the series, Amazon will suggest purchasing books 1, 2, 3 and 4. However, it is highly unlikely that many people purchase book 5 in a series without having already read the previous four books. If order was taken into account, Amazon would suggest purchasing books 6 and 7 and ignore the previous books.

If events are time-ordered and can be represented symbolically, it is possible to treat a sequence of events as an ordered set, or a string. There are many order-preserving algorithms for comparing two strings [14, 31]. For the most part, these algorithms are descendant from Hamming distance, which directly relates to Jaccard distance, which in turn directly relates to Tanimoto distance, a derivative of vector cosine.

Hamming distance is a measure of the number of positions in which two strings have different symbols [30]. Two binary strings, 1011001 and 1001101, have a Hamming distance of 2 because there are two positions (three and five) in which the symbol in the first string is different than the symbol in the second string. This type of measurement is nearly identical to the Jaccard coefficient (and the related measurements).

Levenshtein distance is an extension of Hamming distance [32]. It removes the limitation of a binary alphabet, allowing for an alphabet of any arbitrary size. It

also removes the limitation that the two strings must be of equal length. In doing so, Levenshtein distance counts three types of edits between two strings being compared:

- Insertion: "cat" to "coat" is an insertion of "o".

- Deletion: "link" to "ink" is a deletion of "l".

- Substitution: "lunch" to "lurch" is a substitution of "r" for "n".

Each edit is counted. The total number of edits is the Levenshtein distance between two strings. For example, the Levenshtein distance between "Sunday" and "Saturday" is 3: an insertion of "a", an insertion of "t", and a substitution of "r" for "n". It is common to state that Levenshtein distance is an edit count, the minimum number of edits required to convert string A into string B.

As an optimization problem over two arbitrary length strings, calculating Levenshtein distance is a common example used in dynamic programming [33, 34]. The Wagner-Fischer algorithm is a dynamic programming choice for calculating Levenshtein distance. Given two strings of length m and n respectively, the run-time of the Wagner-Fischer algorithm is $\mathcal{O}(mn)$[35]. A typical recursive solution requires $\mathcal{O}(mn^2)$.

The Wagner-Fischer algorithm is a matrix solution for two strings A and B. For A of length m and B of length n, an $(m + 1) \times (n + 1)$ matrix is created. The top row of the matrix is filled with increasing integers 0, 1, 2, 3... from left to right. Similarly, the left column is filled with increasing integers from top to bottom. If comparing "SUNDAY" to "SATURDAY." With the initial matrix set, each element

of the matrix is filled in from top to bottom, left to right according to equation 2.7.

$$
M_{ij} = \begin{cases} \text{if } A_i = B_j, & M_{(i-1)(j-1)} \\[2em] \text{if } A_i \neq B_j, & \min \begin{pmatrix} M_{(i-1)(j-1)} \\ M_{(i-1)j} \\ M_{i(j-1)} \end{pmatrix} + 1 \end{cases} \tag{2.7}
$$

For example, as "S" equals "S" in the first element to fill in, $M_{1,1}$ is set to zero. "S" in "SUNDAY" does not match "A" in "SATURDAY" hence, $M_{1,2}$ gets the value min(1,2,0)+1, which is 1. After completing all elements, the matrix will contain the values shown in Figure 2.10. When completed, the value in the bottom-right

		S	A	T	U	R	D	A	Y
	0	1	2	3	4	5	6	7	8
S	1	0	1	2	3	4	5	6	7
U	2	1	1	2	2	3	4	5	6
N	3	2	2	2	3	3	4	5	6
D	4	3	3	3	3	4	3	4	5
A	5	4	3	4	4	4	4	3	4
Y	6	5	4	4	5	5	5	4	3

Figure 2.10: Completed matrix for the Wagner-Fischer algorithm.

element is the Levenshtein distance between the two strings. For "SUNDAY" and "SATURDAY", the distance of 3. Because the maximum Levenshtein distance is the length of the longest string (8 in this example), the similarity would be $(8-3)/8$, or 62.5%. Compared to set-based measures of similarity, there are five letters in common out of eight letters used. Jaccard coefficient $= 5/8 = 62.5\%$. Sørensen-Dice's coefficient $= 2 \times 5/(6+8) = 71.4\%$. Compared to vector-based measures of similarity, Tanimoto difference will be Jaccard difference $= 3/8 = 37.5\%$. Vector

cosine over the binary attributes A, D, N, R, S, T, U, Y will be 77.2%. The measure of similarity is comparable to the aforementioned measures of similarity.

String-based similarity metrics do not necessarily have a more accurate measure of similarity. String-based similarity metrics are order-sensitive. Consider changing the order of the characters in the strings. Doing so does not change the result of set or vector-based measures. Each character is an attribute without order. From a string-based comparison, "DAYSUN" and "SATURDAY" have a difference of 7, which is a similarity of $(8 - 7)/8 = 12.5\%$, distinctly different than the "SUNDAY" and "SATURDAY" comparison.

For comparison of search engine usage, Levenshtein distance accurately indicates the ordered difference between users because the order that the search results are selected is maintained. A user with a history of {A, B, C, D} will be considered very different from a user with a history of {D, C, B, A} with a string-based comparison while a set or vector-based comparison will show that the two users selected the same results.

Converting difference to similarity can produce undesirable results due to the varying length of strings being compared. Levenshtein distance is unreliable at comparing short strings to extremely long strings. What if one user has a history of {A, B, C} and another user with a search history containing over 100 items also has visited A, B, and C in the same order? Further, what if many users have visited {A, B, D, C} in that specific order? Identifying this common behavior is important to predicting overall search engine use. In order to handle real-world search engine user data, a method of aligning a short string (the recent history of one user) with a substring of a longer string (the entire history of another user) is necessary.

Given two strings, A and B, a common task that is related to testing for similarity is the task of alignment. Assuming that A is shorter in length than B, the

goal is to alter A in order to maximize similarity (minimize difference) of A compared to B. There are two forms of string alignment: global and local.

Global alignment will add gaps (a special null symbol) to A, increasing the length of A to the same length as B. While doing so, the difference between A and B is minimized [36]. That is not useful for comparing search engine histories. It will expand a short history into a long history with a lot of gaps.

Local alignment not only alters A, it also identifies a substring of B for which the substring and the altered A have the minimum distance. This is technically a global alignment between A and a substring of B [37]. With search histories, local alignment will align a short history with a substring of a long history, identifying where the two histories are the same and what follows in the longer history.

The Smith-Waterman algorithm is an adaptation of the Wagner-Fischer matrix solution used for Levenshtein discussed earlier [37]. With strings A and B of lengths m and n respectively, a matrix M of size $(m + 1) \times (n + 1)$ is created. All cells in the top row and left column of M are initialized to zero. Instead of adding 1 for a mismatch, a similarity function is used to fine-tune how to treat matches and mismatches as the matrix is filled. In Figure 2.11, the similarity function is equation 2.8.

$$\text{Sim}(a, b) = \begin{cases} \text{if } a = b, 2 \\ \text{if } a \neq b, -1 \end{cases} \tag{2.8}$$

The Smith-Waterman algorithm has a deletion (p_d) and insertion (p_i) penalty. If $p_d = p_i$, it is essentially equivalent to a single gap penalty, as commonly used in Needleman-Wunsch implementations [36]. Separating the gap penalty into two penalties allows the implementation to add extra weight to either deletions or insertions. For simplicity, the following example will use -1 for both p_d and p_i.

After the top and left cells are initialized to zero, the rest of the matrix is filled in. Similar to the Wagner-Fischer algorithm, the cells are filled in from the top left

to the bottom right using equation 2.9.

$$M_{ij} = \max \begin{pmatrix} 0 \\ M_{(i-1)(j-1)} + \mathrm{Sim}(a, b) \\ M_{(i-1)j} + p_d \\ M_{i(j-1)} + p_i \end{pmatrix} \tag{2.9}$$

It is important to note that the zero in the max function of equation 2.9 eliminates the possibility of a negative value in any cell of the matrix. Therefore, the cells that contain non-zero values will be those cells in which a match has generated an increase in value from the neighboring cells. An example that compares "BELL" to "UMBRELLA" is shown in Figure 2.11. To locate the local alignment of a completed

	U	M	B	R	E	L	L	A
	0	0	0	0	0	0	0	0	0
B	0	0	0	2	1	0	0	0	0
E	0	0	0	1	1	3	2	1	0
L	0	0	0	0	0	2	5	4	3
L	0	0	0	0	0	1	4	7	6

Figure 2.11: A completed Smith-Waterman matrix.

Smith-Waterman matrix, the cell with the greatest value is located (the cell with a value of 7 in Figure 2.11). From the current cell, the neighboring cell (up, up/left, or left) that contains the greatest value is located. This continues until all neighboring cells contain a zero. In this example, the best alignment begins at the cell containing a 7, continues up/left to a 5, up/left to a 3, then either left or up/left to a 1, and finally to the cell containing a 2. When the symbols in both strings match, write

the symbol. A gap symbol is used otherwise. The local alignment of "BELL" to "UMBRELLA" is "B-ELL."

Local alignment is a useful tool for identifying which part of a long sequence is a good match for a short sequence. For example, assume that a customer's last four purchases are known, each item identified by a letter to be "BELL." To locate trends, the complete purchase history of other customers will be searched for the same sequence of items. Instead of limiting the search specifically to "BELL," local alignment allows for a search of subsequences that are very similar, such as "BRELL." As such, the number of matching search histories will likely be larger than the number that contain "BELL" without alteration.

The primary reason that the Wagner-Fischer and related algorithms are not commonly used is the high complexity. For strings of length m and n, the complexity is bounded by $\mathcal{O}(mn)$. There are many common methods of attacking the complexity problem:

- By maintaining the values of only two rows at a time, the space required in memory is reduced from mn to $2m$. This decreases memory requirement. In the case of large values of m and n, reducing memory requirement may reduce memory swapping, which then may reduce total runtime.

- If the only interest is in detecting a difference that exceeds a threshold k, then it is only necessary to calculate a diagonal stripe of width $2k+1$. The complexity becomes $\mathcal{O}(kn)$, which is faster with the assumption that $k < m$ [31, 38].

- Using lazy evaluation on the diagonals instead of rows, the complexity becomes $\mathcal{O}(m(1+d))$ where d is the calculated Levenshtein distance. When the distance is small, this is a significant improvement [39].

Given two search engine users, local alignment is clearly an accurate method for finding an approximate match between a target user's recent history and another

user's complete history. Then, a neighborhood of users who are similar to the target user may be formed.

2.5. RECOMMENDATION ALGORITHMS

Recommendation algorithms are primarily used to limit the scope of items being presented to a user, focusing the user on items that are likely of most interest to the user. While there are many specific implementations of recommendation, most are based on a few simple strategies, such as recommending items based on frequency or sequential order. The following describes popular forms of recommendation. For consistency, the recommendation algorithms are framed in the environment of a search engine. There is a target user for whom the algorithm is providing a recommendation. The items being recommended are electronic resources, indexed by the search engine and selected by users.

The simplest form of recommendation is the rank or popularity model. Resources are ranked by the frequency of which they are selected. The resources selected most often have the highest rank. The most popular resources are suggested to the user. Due to Zipf's law, suggesting the most popular resources will be somewhat effective as the most popular item will be selected twice as often as the second most popular resource and three times as often as the third most popular resource [40].

A refinement of the rank model has become popularly known as the "people who bought X also bought Y" algorithm from Amazon [4]. Instead of identifying the most popular resources from all users, this "also" model limits the users. In a sense, it is creating a neighborhood of slightly similar users. The most popular resources from this neighborhood are suggested. Because completely dissimilar aforementioned users are omitted, results should be more accurate than the rank model. However, this model requires more computation. Instead of a single list of most popular resources, there is a list for every possible resource that may be selected. In practice, the lists

will not be calculated until required, but processing the most popular resources on demand may be very time consuming, causing a long delay for the user.

Neither models just described take time or order into account. To do so, the resources that may be suggested must come after the resources previously selected by the target user. Instead of suggesting "people who bought X also bought Y," a better suggestion would be "people who bought X **then** bought Y." Using time as context, suggesting the resource that comes later is better defined as prediction [7]. This sort of "then" algorithm should not be more complex than the "also" algorithm. It calculates the most popular resources for every resource and likely will calculate many of the lists on demand. Even if the context of order is tightened to include only resources that immediately follow the last resource selected by the target user, a "next" algorithm should not be more complex than the "also" algorithm. Both the "then" and "next" algorithms use the same neighborhood as the "also" algorithm, but refine the universe of possible resources to suggest by using order, loose ordering in the "then" algorithm and tight ordering in the "next" algorithm.

It may be possible to improve the results further by considering more than one resource that the target user and the population share. In many Asian language input editors, there is a language prediction model that use more than one previously entered character to accurately predict which characters most likely come next [41]. Similarly, the last n resources selected by the target user may be defined as an n-gram [7]. Then, any user who has that n-gram in his or her search history is used for recommendation. The most popular resources that follow the n-gram are suggested to the target user. Order is used, which should increase accuracy. Complexity in this case is increased as there will be a list of popular resources for every permutation of n resources. It is more likely that this "n-gram" model will calculate the lists on demand as it is not likely that users select n resources from a search engine in the same order very often.

These five algorithms, the "rank," "also," "then," "next," and "n-gram" models, cover nearly all of the existing recommendation algorithms. The specifics deal with how similarity is defined and how neighborhoods of similar users are formed. Using Figure 2.12 as a set of example users, it becomes very clear how the neighborhood is refined.

The "rank" model will simply suggest the most popular resource selected by all users, which is X in this example, visited by all but two users. Without a neighborhood or order context, the recommendation will rarely change as the most popular resource will likely remain the most popular resource for a long time [40].

The "also" model limits the neighborhood to users who also selected the target user's last resource, F in this example. Users 2 though 7 have selected F. Examining the histories of those users, the most popular resource (ignoring F) is E, the only other resource that appears in all histories. This clearly exposes the problem with using a recommendation algorithm for prediction. If the recommendation algorithm is not order sensitive, it will recommend a resource previously selected by the target user.

The "then" and "next" models use the same neighborhood as the "also" model, every user who has F in his or her history. Instead of suggesting the most popular of all resources in each user's history, the "then" model suggests the most popular resources that come after F in the collection of histories, I in this example. The "next" model suggests the most popular resources that immediately follow F in the collection of histories. Resource G is selected twice after F while E, H, and X are only selected once. G will be recommended.

Consider the "n-gram" model with $n = 5$. The 5-gram for user 1 is BCDEF. Only user 2 has that 5-gram. Users 3 and 4 are close, but not a perfect match. Therefore, user 2 would be the only user in the neighborhood, recommending resource H. It is important to note that a smaller n-gram will increase the size of the neighborhood.

If $n = 3$, then users 2 and 4 will be included in the neighborhood. If $n = 1$, this becomes the "next" model.

1	ABCDEF (target)
2	MXTBCDEFHIJ
3	PNYBCEFGHJK
4	RBCLDEFGIX
5	HGFED
6	XDPMEF
7	FXORELIWZAN
8	LXMBCDEGHI
9	ACXGIKM

Figure 2.12: Example search engine user histories. Users are identified by numbers. Resources are letters, shown in the order in which they were selected.

2.6. OUTCOME FEEDBACK

Outcome feedback is common in biostatistics and latent semantic indexing [24, 31, 42]. Given a process that produces a result, the output of the process may be used to refine the process itself. In the case of search engine use recommendation, the process involves building a neighborhood of similar users, producing a set of recommendations from the neighborhood, and waiting for the target user to select a resource. If the resource that the target user selects is from the recommended list, that information may be used to help refine the neighborhood.

As a simple example of an outcome feedback loop, consider a population of stock investors. In the manner that many computer algorithms are implemented, a group of people will be examined. The best stock investors, based on a test of each person, are selected. If the pre-test performs well, the investors will turn a profit.

Then, for the next investment period, everyone in the population is tested again to identify the best investors.

As an alternative, the outcome of each person's investment practices may be used. At the end of a preset time period, the profit margin of each investor is calculated. Those who turn a profit above a certain margin remain in the population of investors. All others are removed and possibly replaced with other stock investors, based on the outcome of the pre-test.

The use of outcome feedback, when effective, will reduce pre-testing of the population. Then, the likelihood that a positive outcome will lead to another positive outcome is exploited. Investors who perform well will likely continue to perform well.

When applied to search engine users, the population is the neighborhood of similar users used to make recommendations. The outcome occurs when the target user makes a resource selection. If a member of the neighborhood contributed to the suggestion of the resource that was selected, that member will be maintained in the neighborhood. All other users are candidates for replacement with more similar users. When successful, this will reduce calculations required to identify similar users while improving overall recommendation accuracy.

2.7. SUMMARY

There are many options available for comparing search engine users to one another and identifying behavior trends. This section covered the basic concepts of using state-space models to identify trends and dividing populations of users into smaller neighborhoods to refine the state-space models. Modeling and neighborhood algorithms are dependent on a means of identifying similarity.

Three basic models of similarity were discussed in this section: vector, set, and string-based algorithms (see Figure 2.13). Vector-based algorithms are popular, but lack any sense of an order of attributes. Set-based algorithms are simplified

vector-based algorithms that allow for categorical attributes instead of strictly numeric attributes. String-based algorithms also allow for categorical attributes, but maintain an order of attributes. Because they maintain order, string-based algorithms are far more complex than vector or set-based algorithms. Because search engine usage is ordered, string-based algorithms are preferred to vector-based algorithms when comparing the search engine usage of one user to another user.

A specific type of string-based comparison algorithm is required to compare a short string to a long string, aligning the short string to a substring of the long string. Local alignment algorithms identify the best alignment of a short string within a long string. The Smith-Waterman version described in this section also provides a measure of how similar the short string is to the substring with which it is aligned. As such, the Smith-Waterman algorithm will be useful when comparing historical search engine usage between users. With the concept of state-based modeling and comparison algorithms in place, five types of recommendation were defined: "rank," "also," "then," "next," and "n-gram". Each algorithm is designed to increase accuracy while also increasing complexity, as shown in Figure 2.14. Prediction is a special type of recommendation with the context of time or order. Instead of recommending an item that may be of interest to the user, a prediction algorithm recommends an item that will be of interest at a specific time or in a specific sequence.

The "rank" and "also" algorithms are simple and popular. For prediction, "rank" and "also" cannot be used. Prediction requires a concept of what comes next. Neither "rank" nor "also" take time or order into account.

The "then" and "next" algorithms may be referred to as a prediction algorithms. Both omit recommendations that only occur in the past, providing recommendations that usually occur in the future. The "n-gram" refines the prediction not by better identifying the resources being recommended, but by further refining the neighborhood of similar users.
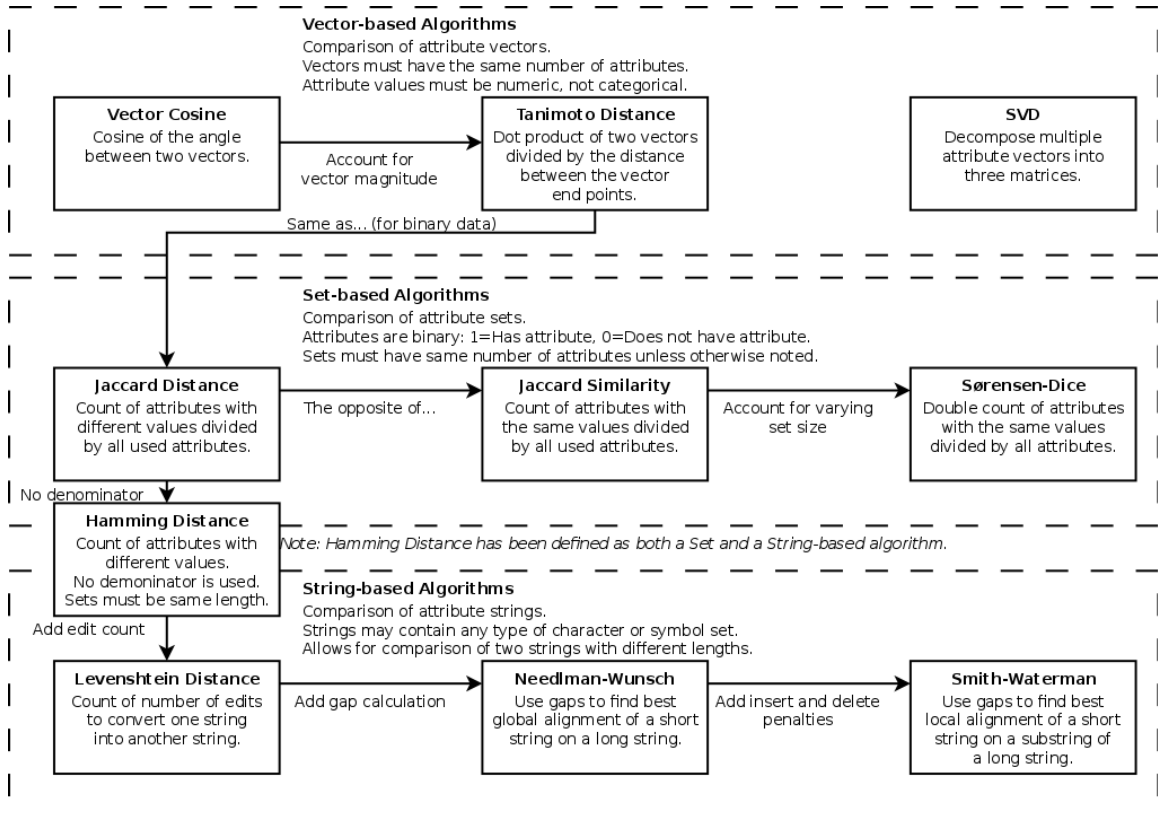
Figure 2.13: Comparison algorithms covered in Section 2.

For search engine prediction, the "n-gram" model appears to be the best. However, it makes the assumption that there will be users who select n resources in the exact same order. When that does not happen, the neighborhood of similar users shrinks to zero and no recommendation is possible.

Rank Model	Recommendation: Most popular resources. Neighborhood: Entire population.

Refine to similar users

Also Model	Recommendation: Most popular resources. Neighborhood: Users who also selected the last resource selected by the target user.

Add order to the recommendation

Then Model	Recommendation: Most popular resources that come after the last resource selected by the target user. Neighborhood: Users who also selected the last resource selected by the target user.

Confine order to what comes next

Next Model	Recommendation: Most popular resources that immediately follow the last resource selected by the target user. Neighborhood: Users who also selected the last resource selected by the target user.

Refine the neighborhood further

n-Gram Model	Recommendation: Most popular resources that follow the last n resources selected by the target user. Neighborhood: Users who also selected the last n resources selected by the target user in the same order.

Figure 2.14: Recommendation models covered in Section 2.

3. PROACTIVE SEARCH ENGINE

3.1. INTRODUCTION

With computer microminiaturaztion advances, adoption of the Internet, and proliferation of mobile networked devices, the last twenty years have witnessed an explosion of electronic information. As a result, focus has been placed on data discovery [2]. To meet this challenge, a tool is required that allows anytime, anywhere, transparent, timely, relevant, reliable and cost-effective access to the information, regardless of heterogeneity of access devices, communication medium, and autonomous nature of information sources. An intelligent search engine is required [1, 5, 43, 44].

To function, a search engine must know what information the user requires. A query is made. In Google, the user types a description of a desired web page. In Netflix, the user types the name of a movie or actor. In Tineye, the user supplies an electronic image. The search engine accepts the query from the user and uses it to locate electronic resources. A list of resources is provided to the user. If successful, the user selects the resource that contains the information required. This query-response model has become a standard for search engines [2].

As a proactive measure to augment the reactive design of search engines, a separate application has been developed that recommends resources of information to the user. Recommender systems first became apparent in large web-based shopping sites, such as Amazon's "People who bought this also bought" application. Described in Section 2.5, there are many types of recommender systems. Some recommender systems are simply statistical. Others improve recommendation based on measures of similarity between objects such as users or movies.

To refine recommendation, users are grouped into neighborhoods based on similarity. Section 2.4 covers many forms of similarity algorithms. Overall, five forms

of recommendation were introduced: "rank," "also," "then," "next," and "n-gram." The n-gram model should perform best because the neighborhood of users is limited to only those users who behaved exactly as the target user over the last n observations [7]. However, the neighborhood size shrinks, often to zero, as n in increased.

This section proposes a proactive search engine model that is as accurate as the n-gram recommendation algorithm while not prone to failure caused by small neighborhoods of similar users. Section 3.2 defines how each prediction is performed. Section 3.3 defines how the neighborhood of similar users is produced. Section 3.4 suggests multiple methods to reduce complexity.

3.2. PREDICTION ALGORITHM

The proposed proactive search engine may be described as an approximate n-gram recommendation algorithm. As with the n-gram algorithm, the resources that immediately follow an n-gram match of the target user's recent history are suggested. The difference between the n-gram algorithm and the proposed proactive algorithm is how the n-gram match is performed. The proposed algorithm implements an approximate match, not an exact match.

Local alignment, defined in Section 2.4, aligns a short string with a substring of a longer string. Figure 3.1 shows how the string KLATU aligns with three other longer strings. KLATU is not a perfect n-gram match to any of the longer strings, but an approximate match is evident. Once an alignment is made, the items that immediately follow the alignment are used for recommendation. In Figure 3.1, the recommendations are N, S, and O. If KLATU were compared to a large population, certain recommendations should be more common than others. The most popular recommendations are shown to the target user.

At this point, it is necessary to discuss what should be recommended to the user. It is common for search engines, such as Google and Bing, to suggest search

Target	...KLATU
User 1	BARADAKLAATU**N**IKTO
User 2	STKLAU **S** E
User 3	HUTTSKLAT**O**OINE

Figure 3.1: Aligning the target user's recent history with other user's complete history.

queries [1, 45]. If you type a word or two, common queries are displayed. If you enter a query, similar queries are suggested. However, users are not searching for queries. Users are searching for electronic resources. Further, the relationship between queries and electronic resources is not one-to-one. A query for "hedgehog" could refer to a small spiny mammal, an anti-submarine weapon, or a popular chocolate treat. Predicting that the user would enter "hedgehog" does not identify the resource the user desires. The proactive search engine must predict resources to the user, not queries.

It must be noted that a single prediction is not necessary. Search engines naturally display multiple results to the user. A proactive search engine can, and should, display multiple recommendations to the user. If the user does not see a recommendation that he or she wants, the query interface will still be present. Currently, search engines display the query interface while displaying search results. Therefore, displaying recommendations will not impact the interface, but will likely improve average search time and accuracy.

3.3. SIMILARITY NEIGHBORHOODS

Section 2.3 describes multiple implementations of the k nearest neighbors (k-NN) algorithm. A neighborhood of k nearest neighbors to a target user will require comparing the target user to every other user. Consider Google. With millions

of users, the time required to identify the k most similar users to a single target user will exceed any reasonably acceptable timeframe. Worse, attempting to identify the k most similar users to each separate user, in order to define each user's own neighborhood, will require comparing every user to every other user. An optimal k-NN algorithm will simply not work.

An alternative to identifying the k most similar users is to identify users who are at least k similar to a target user. If k is large enough, it should not be necessary to compare a target user to the entire population. Further, as users are identified as being at least k similar to the target user, the value of k may be refined to hunt down the most similar users as time permits.

Considering the method of beginning with a large value of k and reducing it as the neighborhood of similar users grows, it is not necessary to set a value for k. Assume that a neighborhood of at least twenty users is desired. If k begins with an extremely large value, every user will be at least k similar to the target user. Every user will be included in the neighborhood. When the neighborhood exceeds twenty users, k is changed to a value just shy of the similarity between the target user and least similar user in the neighborhood, ejecting the least similar user from the neighborhood. This refinement continues. With every user that is included, the size of the neighborhood exceeds twenty users and the least similar user is ejected from the neighborhood. Over time, the neighborhood will improve as more similar users are identified.

In this proposal, k is not a set value. It is merely the similarity between the target user and the least similar user in the neighborhood. Initially, the neighborhood is filled with twenty random users. The least similar user is identified. Then, a user is selected at random to be included in the neighborhood. If that user is more similar to the target user than the current least similar user, the two are swapped and the

least similar user in the neighborhood is recalculated. The more time that is allowed for this process, the more similar the users in the neighborhood will become.

This dynamic neighborhood process does not define similarity. Search engine users are defined as ordered sequences of search results. Comparing one ordered set to another ordered set is a string-based comparison, as defined in Section 2.4. Levenshtein distance is the standard form of string-based comparison algorithms. Using Levenshtein distance creates a problem, how to handle comparison between users with extremely short usage histories to users with extremely long usage histories.

Consider a user who has selected three resources, a history of ABC. If compared to a user with the history of DEF, the Levenshtein distance will be 3. If compared to a user with the history of ABCDEFG, the Levenshtein distance will be 4 even though ABC is a perfect substring of the longer history. One method is to divide the distance by the maximum possible distance, which is the length of the longest string. The comparison to DEF will be $3/3 = 100\%$. The comparison to ABCDEFG will be $4/7 = 57\%$.

Local alignment is used to identify the resources for recommendation. Local alignment is based on the Wagner-Fischer algorithm [37]. Levenshtein distance is calculated using the Wagner-Fischer algorithm [35]. Therefore, it should be possible to calculate similarity between users while identifying the local alignment. A method for doing so is defined in Section 3.4.

For the proposed proactive search engine, a neighborhood of similar users is dynamically constructed over time by randomly identifying users who are more similar than the least similar user currently in the neighborhood. Similarity is defined using local alignment, which is a required calculation during the recommendation process. With this design, the longer the algorithm has to process, the better the recommendation should become.

3.4. REDUCING COMPLEXITY

Local alignment was described in detail in Section 2.4. The Smith-Waterman algorithm, implemented using the Wagner-Fischer method, provides a means of calculating the best local alignment of an n-gram on a longer string. However, the Wagner-Fisher method is rarely used for large populations because the complexity of a single comparison has a runtime complexity of $\mathcal{O}(mn)$ for two strings of lengths m and n.

For the proposed algorithm, it is not necessary to consider the target user's entire history. Assume that a user has been using the search engine for multiple years, with thousands of resource selections. The most recent resource selections are more important in defining current behavior than distant history. The same does not apply to the comparison users. It may be found that the target user's recent n-gram of resource selections is found in another user's history from many months ago. Therefore, n in the measure of runtime complexity is limited by the length of the n-gram and hence it is a constant. The length of the longer string, m, is the driving factor for complexity. There are multiple methods for limiting the complexity introduced by m:

- Only use the recent history of users with extremely long histories. For example, if a user has many years worth of selection histories, only the most recent year may be of interest. This is justified by understanding that the overall nature of human behavior changes over time [10]. Ancient history may not be as effective in predicting current behavior.

- Identify how many of the n items are actually selected by the comparison user. Assume that n is five and three of the five resources are required to even consider the comparison user for the target user's neighborhood. When calculating the first row of the Wagner-Fischer matrix, every resource in the target user's history

will be calculated. If the minimum of 3 resources are not identified in that first pass, the remaining four passes to compare all five of the target user's n-gram are not necessary. Doing so will only perform $1/n$ of the calculations required for a complete local alignment when a comparison user is certainly not going to be used.

- As described in Section 2.6, assume that users who correctly contributed to the target user's prediction are similar to the target user and do not compare them to the target user again.

While the Smith-Waterman algorithm is commonly used for local alignment, it is not commonly used as a comparison of similarity. There are two expected models for prediction:

- User Levenshtein distance to identify a neighborhood of k similar users. Then, use the Smith-Waterman algorithm to identify a local alignment within each member of the neighborhood.

- Use the Smith-Waterman algorithm to identify a local alignment with each member in the population. Then, use Levenshtein distance to identify the most similar alignments, including those in the neighborhood.

Both methods perform a complex matrix calculation twice per user. Because both Levenshtein distance and the Smith-Waterman algorithm are calculated with a form of the Wagner-Fischer algorithm, it is possible to construct Levenshtein distance from the local alignment itself. The matrix calculation will only be performed once.

A local alignment has gaps when a character in the substring of the longer string does not occur in the comparison n-gram. Each gap is an insert as used in the definition of Levenshtein distance. Therefore, counting the gaps used in the local alignment is part of the calculation of Levenshtein distance.

If a character of the n-gram is missing from the local alignment, it is a deletion as used in the definition of Levenshtein distance. Therefore, the n minus the number of non-gap characters in the local alignment is the number of deletions in the local alignment. The number of gaps and number of deletions is nearly a full calculation of Levenshtein distance and all that is usually necessary. If it is necessary to identify substitutions, a substitution is any instance in which a deletion and insertion (a gap and a missing character) occur in the same location.

All three values, inserts, deletions, and substitutions, may be calculated with counters while back-tracking through the matrix in the final step of the Smith-Waterman algorithm. Using Figure 3.2 as an example, the local alignment ends at the value of 6. The row and column are both labeled with E, so the local alignment is initially E. Backtracking goes to the greatest value, which may be either 3 neighboring the 6. Moving to either 3 will move up one row and, because the G does not match either the D or the E, a gap is inserted. Moving a row and inserting a gap at the same time is a substitution and the current local alignment is -E. From either 3, the next move will be to the 4. The local alignment will be D-E. From the 4, there are two possibilities for moving to 1. In this example, a left move will be made, but the resulting local alignment and distance value would be the same if the move was up and to the left. Moving left will add a gap to the local alignment, resulting in -D-E. From the 1, the next move is to the 2. Again, this moves up a row while inserting a gap, so there are now two substitutions and the local alignment is –D-E. Finally, the local alignment ends at the 3 as H–D-E. In the end, there are three gap characters, indicating three inserts. The number of deletions is the length of the original string minus the number of non-gap characters in the resulting alignment, which is $5 - 3 = 2$. Two substitutions were recorded when performing the alignment. The distance is the number of inserts plus the number of deletions minus the number of substitutions, or $3 + 2 - 2 = 3$. With the understanding that local alignment is

	C	H	O	R	D	E	D
	0	0	0	0	0	0	0	0
H	0	0	3	2	1	0	0	0
E	0	0	2	2	1	0	3	2
D	0	0	1	1	1	4	3	2
G	0	0	0	0	0	3	3	2
E	0	0	0	0	0	2	6	5

Figure 3.2: An example of local alignment with an insert, a deletion, and a substitution.

used to identify local alignment, similarity, and the suggested resource which follows the local alignment, a large part of reducing runtime complexity involves reducing the time required to perform a local alignment. Many suggestions appear in literature, such as performing a lazy evaluation or prematurely ending a calculation once it is identified as being poor [20, 39]. The following implementation is customized to the requirements of the proposed proactive search engine, but may be applied to other uses of local alignment.

When comparing a target user's most recent n-gram to another user's history, first identify which characters exist in both histories with a +3 (using a match value of 3 in this example). As shown in Figure 3.3, a +3 is in each cell where the character on the left matches a character on the top. Of note, it is not in any way possible for the local alignment to carry over into the first or last columns. These are ignored from any further comparison. After each element of each string is compared, prefilling the matrix with values, the actual values are filled in. Figure 3.4 shows the matrix filled in. Only cells with a +3 have values. The columns that cannot be part of the alignment are ignored. The cells to the right and lower right of the +3 are automatically filled in during this process. In Figure 3.2, every cell was calculated by doing a character-to-character comparison and three cell-to-cell comparisons. A total of $4 \times 5 \times 7 = 140$

		C	H	O	R	D	E	D
	0	0	0	0	0	0	0	0
H	0		+3					
E	0						+3	
D	0					+3		
G	0							
E	0						+3	

Figure 3.3: Local alignment step one, identifying characters shared by both strings.

comparisons were performed. In Figure 3.4, every character-to-character comparison was still performed, but only 9 cell-to-cell comparisons were made (3 per each cell marked +3 where the neighboring cells are not ignored). A total of 29 comparisons are performed and the same result is achieved.

		C	H	O	R	D	E	D
	0	0	0	0	0	0	0	0
H	0		3	2	1			
E	0			2	1		3	
D	0					4	3	
G	0						3	
E	0						6	

Figure 3.4: Local alignment step two, filling in the cells identified with +3.

3.5. MAKING A PREDICTION

Sections 3.3 and 3.4 provide the basis for forming a neighborhood of similar users and then locating the best alignment of the target user's recent history with the histories in the target user's neighborhood. A prediction is then made by ranking the

resources that immediately follow each neighborhood member's alignment from most popular to least popular. Figure 3.1 is a brief example of aligning the target user's recent history with a neighborhood and identifying the resource that immediately follows the alignment per user.

This process runs in an eternal loop. When the user is not using the search engine, the neighborhood process runs. A random user is selected and compared to the target user's recent history. If the random user is more similar than the least similar member of the target user's neighborhood, the random user is admitted to the neighborhood and the least similar member of the neighborhood is ejected. This is clearly best implemented as an agent process, with a neighborhood building agent assigned to each user. The user objects need not be complex. As shown in Figure 3.5, the user object only needs to store similarity, the index of the alignment of the target user's recent history to the neighborhood user's history, and the recommended resource that follows the alignment. When the user accesses the search engine, a

User ID	Ex1234
Similarity	0.42
Alignment	37
Recommend	9669

Figure 3.5: A neighbor object retains similarity, alignment, and recommendation.

quick survey of the neighborhood is made. Each member of the neighborhood will recommend the resource that immediately follows the user's alignment to the target user. The most common resources are recommended to the target user as part of the initial search engine interface. The user has the option to select one of the recommended resources or enter a search query.

If the user selects a resource from the recommended list, it is obvious that many of the members of the neighborhood suggested that resource. As a runtime improvement and an attempt to improve accuracy, members who suggested the selected resource are automatically given a special status. Instead of recalculating the alignment between these users, the previous alignment is shifted one position, assuming that the resource that follows the one previously suggested will be the just as accurate as the current success. Further, these members should not be replaced. Each is artificially given a perfect similarity to the target user. Using the outcome of the user's selection to improve the neighborhood is a form of outcome feedback defined in Section 2.6.

3.6. SUMMARY

The proposed proactive search engine algorithm follows in the development of popular recommendation algorithms. There are two parts to the proposed algorithm: First, a neighborhood of similar users is developed. Then, each member of the neighborhood suggests a resource to the target user. The suggestions are ranked from most popular to least popular and shown to the target user.

Local alignment is used to identify which resource follows the target user's most recent n-gram in the comparison user's history. At the same time, the local alignment identifies the similarity between the two users. Therefore, local alignment provides three values: the similarity between the two users, the position in the comparison user's history of the optimal local alignment, and the resource that follows the local alignment.

To avoid comparing the target user to every user in the population, a dynamic neighborhood algorithm is used. The neighborhood is filled with random users. Each of those users is compared, using local alignment, to identify the user's similarity to the target user (as well as the suggested resource). Then, a random user is chosen

and compared using local alignment. If the new user is more similar to the target user than the least similar user in the neighborhood, the new user is added to the neighborhood and the least similar user is ejected. As long as time permits, random users are continually compared and, when necessary, swapped in.

After the target user makes a selection, the target user's recent n-gram will change. All previous measures of similarity will be obsolete. The process of using local alignment on every member of the neighborhood begins again. To avoid much of this process, outcome feedback is used.

After the target user makes a selection, each member of the neighborhood who suggested the resource that the target user selected is automatically maintained in the neighborhood. The similarity of the user is artificially marked as "perfectly similar." The local alignment marker in the neighborhood user's history is incremented by one, identifying the next resource as the new suggestion. If 10% of the neighborhood is correct in the recommendation, 10% of the local alignments required to make the next recommendation will not be necessary.

4. IMPLEMENTATION AND TESTING

4.1. INTRODUCTION

The proposed proactive search engine is intended to extend the scope of user behavior prediction by replacing existing n-gram match approaches with an approximate match. The approximate match is implemented as a local alignment of the target user's most recent search history against similar user's search histories. To avoid redundancy, the local alignment used for approximate matching is also used for calculating similarity while building a neighborhood of similar users.

To further reduce processing time, outcomes are used to maintain the neighborhood of similar users. Users who supply the correct prediction are automatically kept in the neighborhood without further calculation. Only those who fail at prediction are replaced.

This proposed approach is unique as an approximate matching approach with outcome feedback. The proposed scheme should offer a higher performance in accuracy with limited increase in processing time. To test this hypothesis, the proposed proactive search engine algorithm has been implemented and tested with real-world search engine databases. To measure the value of the proactive search engine, it is compared to many other existing recommendation algorithms.

4.2. TESTING ALGORITHMS

Due to the complexity of the proactive search engine, the proposed scheme must be thoroughly verified to justify the viability of each step, as well as the whole. The following tasks were verified, respectively:

- Approximate alignment between users

- Accuracy of dynamically-constructed neighborhoods

- Effectiveness of outcome feedback in reducing complexity, while maintaining accuracy

To demonstrate that approximate alignment is effective, many methods of recommendation are compared to one another across all data sets. For each algorithm tested, the records of the data set are parsed one at a time using algorithm 4.2, each record containing a user, a time, and a resource selected. Within the algorithm, t and u are users, defined as an ordered string of resources. The functions "position()" and "distance()" are nearly identical across all recommendation algorithms.

The position function aligns a short resource string on a longer resource string, returning the position of the best alignment. If no alignment is possible, a null is returned. If the short resource string is null, every position in the longer resource string is a perfect alignment. The last position will be returned.

The distance function measures the lack of similarity (edit distance) between a short resource string and a position in a longer resource string. If the short resource string is a perfect substring match at the given position of the longer resource string, a zero is returned. When the length of the short resource string is one resource, this is just a check to identify if the resource at the given position of the longer string is actually the lone resource in the shorter resource string.

Subtraction between resources returns the difference between the position of the two resources. If the first resource immediately follows the second resource, the difference is 1. The difference is negative when the first resource occurs before the second resource. The algorithms tested are based on popular recommendation algorithms, described in Section 2.5. Each algorithm is intended to be an improvement on the algorithm before it. The values of n, d, l_{lo}, l_{hi}, and s for each algorithm are shown in Table 4.1. The following is a description of how the recommendation algorithm is intended to function:

Algorithm 4.2 Recommendation testing algorithm.

$\{n, d, l_{lo}, l_{hi},$ and s are set by each recommendation algorithm$\}$
$t \leftarrow$ the target user
$U \leftarrow$ all users except the target user
$h \leftarrow$ last n resources in t
$R \leftarrow$ initially empty list of resource suggestions
for all $u \in U$ **do**
  $a \leftarrow$ position(h,u)
  if a is null **then**
    continue
  end if
  if distance(h,a)$> d$ **then**
    continue
  end if
  for all $r \in u$ **do**
    if $l_{lo} \leq (r - a) \leq l_{hi}$ **then**
      Add r to R
    end if
  end for
end for
Suggest s most common r in R

---

1. Rank: The twenty resources selected by the most users are recommended.

2. Also: This is the "rank" algorithm with the population of users limited to users who also selected the last resource selected by the target user.

3. Then: This is the "also" algorithm with the resources limited to resources selected any time after the last resource selected by the target user.

4. Next: This is the "then" algorithm with the resources confined to the resources selected immediately after the last resource selected by the target user.

5. n-Gram: This is the "next" algorithm with the population limited to users who selected the exact same last five resources as the target user, in the exact same order. Only resources selected immediately after the 5-gram are considered.

6. Approximate: This is the "*n*-gram" algorithm with an approximate alignment used instead of an exact 5-gram match.

Table 4.1: The n, d, l_{lo}, l_{hi}, and s for each tested recommendation algorithm.

Algorithm	n	d	l_{lo}	l_{hi}	s
Rank	0	0	$-\infty$	∞	20
Also	1	0	$-\infty$	∞	20
Then	1	0	1	∞	20
Next	1	0	1	1	20
n-Gram	5	0	1	1	20
Approximate	5	3	1	1	20

In the initial tests, the entire population was used for each algorithm. Separately, a neighborhood of the twenty most similar users was used for each algorithm. The optimal neighborhood that formed should perform better than using the entire population. Therefore, the optimal neighborhood test was compared directly to a dynamic neighborhood test.

The dynamic neighborhood test follows the algorithm defined in Section 3.3. The neighborhood is initially filled with twenty random users. Then, a total of twenty random users who are not in the neighborhood are tested for similarity. If the new user is more similar to the target user than the least similar user in the neighborhood, the new user is added to the neighborhood and the least similar user is ejected.

Dynamic neighborhoods require time to become viable. Therefore, the neighborhood tests are only performed on users with at least one hundred resource selections. All other users are used for recommendation, but are never considered a target user. (Because set Y does not have any users with at least one hundred resource selections, it is omitted from neighborhood testing.) To further demonstrate the

improvement over time, the dynamic neighborhood results of the first fifty resource selections per user were tested separate from the rest of the resource selections.

The purpose of the neighborhood tests is to demonstrate that a dynamic neighborhood algorithm will, over time, perform nearly as well as an optimal neighborhood. The tradeoff in accuracy is worth the extreme reduction in runtime complexity. The final proposal to further reduce runtime complexity is not a tradeoff with accuracy. It is intended to increase accuracy.

The final test compares the outcome-based neighborhood to the optimal and dynamic neighborhood algorithms. The outcome-based neighborhood is an addition to the dynamic neighborhood algorithm. After the target user makes a resource selection, the outcome may be compared to the recommendations from each user in the target user's neighborhood. Any user who made a correct recommendation should be kept in the neighborhood, regardless of how similar that user may be to the target user. Further, it is not necessary to perform an alignment to identify which resource comes next. For each user who made a correct recommendation, it is only necessary to identify which resource follows the resource the user recommended. It is merely incrementing a pointer to a position in the user's history. Because these users will not be replaced, it is not necessary to find a random user to replace them. For a neighborhood of twenty users, if five make a correct prediction, only fifteen random users are located and tested to see if they are more similar than the least similar users in the neighborhood.

4.3. TESTING DATABASES

Optimally, a proactive search engine should be tested in real time using a real search engine. Without a large population of users, each with a long history of resource selections, there is no data available for the proactive search engine to predict which resource a target user will select. Only large commercial search engines

have the users and history to perform a reasonable test. Therefore, a live test is not possible.

As an alternative, search log extracts from many types of search engines have been used for testing. Each data set is a set of the basic tuple user, time, resource, indicating that the user selected a specific resource at a specific time. Locating and collecting data sets is a difficult task, as most search logs are publicly unavailable. As a rule, search engine companies do not make their search logs public. The data sets used in our experiments came from the following sources. Table 4.2 summarizes the size information of these sets.

1. Set A comes from AOL. In 2006, AOL Research released a data set of three months of Internet searches with deidentified user and identified resource information.

2. Set E comes from Every Busy Woman, an online catalog of women-friendly businesses. The owner of the website authorized one year of deidentified search logs to be used for this research.

3. Set L comes from MovieLens. As a recommender system, MovieLens is not truly a search engine. The data includes user, time, movie, and rating, with the assumption that users enter ratings in the order that they watch the movies. By removing ratings, this is an approximation of user, time, and movie for a movie search engine.

4. Set M comes from the Medical University of South Carolina employee training system. This is a small data set used primarily as a quick sanity test during development before testing the larger data sets.

5. Set N comes from Netflix. There are many Netflix data sets released during the Netflix Prize competitions. This data set is nearly identical to set L. It is

also an approximation of a movie search engine. As the largest data set, the primary purpose of set N is for complexity and runtime testing.

6. Set V comes from AltaVista. AltaVista released a day of search logs for September 8, 2002 to provide researchers with a library of real-world search queries. The data set contains a deidentified user, time, and query. When a link is selected, the URL is provided.

7. Set W comes from AllTheWeb. In 2001, AllTheWeb released snapshots of their search logs, containing IP address, time, query, and resource selected. The purpose of the snapshots was to aid research in query processing. The data set contains all search queries performed over a single day.

8. Set X comes from Excite. To support research, Excite used to produce deidentified search logs, one day per release. The September 16, 1997 data set is used because it contains the time of the search, unlike other Excite releases.

9. Set Y comes from Yandex. Unlike American search engines, Yandex regularly releases short snapshots of their search logs. This data set contains one week of search engine usage.

An overwhelming problem with these data sets is that they are short in time, often a single day, and they are likely edited to remove assumed anomalies. The following describes the details of each data set in order to make a hypothesis about the outcome of proactively guessing the resource for each record in the data set.

The sheer size of each data set varies from very small to very large. Of specific interest is the ratio between users and resources. A data set with far more resources than users will likely have less overlap of resource selection between users as it is possible for each user to select a resource without any two users selecting the same resource. Set W has a high resource to user ratio, indicating that many resources

will not be selected by multiple users. Conversely, having a large user to resource ratio indicates that resources must be selected by multiple users. Sets L and N have very high user to resource ratios. The data sets were normalized such that each

Table 4.2: Data Set Size.

Set	Records	Users	Resources
A	114,494	18,526	57,018
E	168,387	12,857	10,458
L	1,000,209	68,404	3,708
M	3,168	45	255
N	23,168,232	463,616	17,755
V	7,669	1,071	4,631
W	447,435	45,617	381,521
X	97,211	17,498	74,964
Y	30,655	3,121	27,910

resource selection is an integer tuple user, time, resource. Then, the data was pruned, removing data that would not contribute to this research. Users who do not make at least three resource selections were deleted from the data set. Without a history for the user, it is not possible to make a recommendation. Repeated resource selections were combined into a single resource selection. The selection sequence 5, 6, 6, 7 became 5, 6, 7. However, repetition separated by other resources was maintained. The selection sequence 5, 6, 7, 6 was not altered.

Understanding that the data sets are limited and likely manipulated by the data source administrators, effort was placed in identifying useful attributes of the data sets. Some attributes are used to identify manipulations. Other attributes are used to identify which data sets will likely perform well in testing.

For proactive recommendation to be successful, it is necessary to have multiple users select the same resource. If each user selects a unique set of resources, with no

overlap between users, recommendation is not possible. There are two measures of how well resources are distributed among the users. A direct measure is a count of how many resources are selected by more than one user. A similar measure is the average number of users who selected each resource. However, manipulation may force the data to have multiple users per resource.

Based on the Zipf-Mandlebrot law, the frequency distribution of resources should be logarithmic if it is natural, i.e. not manipulated. Table 4.3 lists the rank distributions of each data set along with the previously mentioned metrics: percent of resources that are shared by at least two users and the average number of resources per user. A value of 0.00 indicates a flat distribution in which each resource was selected by the exact same number of users. A value of 1.00 indicates a logarithmic, also referred to as a $1/r$, distribution. Any value below 0.5 is more flat than logarithmic, indicating the likelihood that the data set was manipulated before it was released for testing. Most of the data sets reflect more of a flat distribution than a logarithmic

Table 4.3: Resource Distribution.

Set	% Shared	Avg U/R	Rank Dist
A	25.8%	1.7	0.46
E	62.3%	10.5	0.08
L	96.9%	269.9	0.29
M	33.3%	4.4	0.66
N	93.2%	25.3	0.15
V	13.8%	1.3	0.26
W	8.3%	1.1	0.43
X	47.6%	1.2	0.40
Y	5.1%	1.1	0.26

distribution. This observation became a question about order. Do the data sets exhibit a natural order? In this case, order represents a natural progression from one

set of search results to another set of search results. Further, do the users tend to move from a widely varied set of search results to a small set of popular results? If so, it is possible to state that users tend to converge on a small set of popular resources. Measuring this sense of order and convergence is a problem in itself.

Based on information theory, specifically the work of Minkowski, order and convergence were measured using comparable estimates. Order was measured as a count of distinct 5-grams divided by the total number of records. If order is high, the number of distinct 5-grams will be low, resulting in a lower result. For clarity, the result is subtracted from 1 so that a higher value indicates a higher level of order. Convergence was measured by plotting the number of unique resources selected in the first 10% of each user's history, then the next 10%, continuing to the final 10% of each user's history. If the data set converges, the number of resources selected should reduce over time. A linear trendline across the plot should have a negative slope. Table 4.4 lists the order and convergence for each data set. Figure 4.1 provides a graphical representation of four of the data sets, making the order and convergence easier to visualize. Based on an analysis of each data set, it is clear that any form

Table 4.4: Order and Convergence.

Set	Order	Convergence
A	0.50	-0.01
E	0.62	-0.36
L	0.24	0.28
M	0.70	-0.42
N	0.18	-0.01
V	0.26	-0.03
W	0.17	-0.04
X	0.14	-0.05
Y	0.29	-0.05

of recommendation or prediction will be difficult. In sets A, V, W and X, most of the resources are not shared by at least two users. Any record which selects one of the unshared resources is guaranteed to be unpredictable. Sets L and W lack order. There is little sense that a specific resource will follow another resource. It appears that set L diverges, but closer analysis of the data shows that it has a convergence of 1.78 over the first half of the data and a convergence of -1.58 over the second half of the data for an overall convergence of 0.28. With the exceptions of sets E and M, the data sets do not significantly converge.

4.4. RECOMMENDATION RESULTS

The results of testing generally support the assumptions made about order and convergence in the previous section. Data sets with higher order and higher convergence lend to higher rates of success. However, the point of these tests are not to identify which data sets are more predictable. These tests are a comparison of various algorithms from simple rank recommendation to the proposed proactive search engine described in Section 3.

Table 4.5 contains the results of testing each recommendation algorithm without a neighborhood. The first percent is the absolute success rate: number of times for which the user selected a resource that was recommended divided by the number of attempts to make a recommendation. When the user's selected resource was not selected by any other user, no attempt to make a recommendation was made. The percent in parenthesis is a weighted success rate. The absolute success rate counts a value of one for each success. The weighted success rate counts a value of $1/r$ where r is the rank of the recommendation in the recommendation list. If the resource the user selected is third in the recommendation list, the numerator of the weighted success rate is only incremented by $1/3$.

In most recommendation testing, a more complex weighted measure is used that involves the interest level of the resource. Such a complication is unnecessary here as the resource selected has a perfect 100% interest and all other resources have an absolute 0% interest. Therefore, multiplying by interest would give full weight to the resource that was selected and negate all other recommendations. The recommenda-

Table 4.5: Comparison of recommendation algorithm test results. Absolute Percent Success (Weighted Percent Success).

Set	Rank	Also	Then	Next	n-Gram	Approx
A	8% (4%)	8% (4%)	6% (4%)	6% (3%)	0% (0%)	7% (3%)
E	25% (5%)	52% (16%)	58% (22%)	74% (48%)	46% (43%)	75% (54%)
L	5% (1%)	7% (2%)	12% (5%)	32% (12%)	0% (0%)	33% (20%)
M	85% (26%)	82% (27%)	82% (41%)	83% (68%)	43% (39%)	86% (68%)
N	0% (0%)	6% (1%)	6% (1%)	11% (1%)	0% (0%)	16% (2%)
V	25% (6%)	12% (7%)	14% (7%)	18% (7%)	0% (0%)	21% (11%)
W	7% (2%)	8% (3%)	8% (5%)	8% (6%)	0% (0%)	9% (8%)
X	7% (2%)	7% (2%)	7% (3%)	7% (4%)	0% (0%)	7% (5%)
Y	4% (2%)	20% (7%)	22% (8%)	22% (8%)	0% (0%)	21% (17%)

tion algorithm test results support the claims made in forming the proposed proactive search engine. Order is important. In general, the "then" algorithm performs better than the "also" algorithm. It is also apparent that when order exists, tightening the distance between the alignment and the recommended resources improves success as "next" generally performs better than "then." The "n-gram" algorithm was expected to fail due to the lack of multiple users selecting the exact same five resources in the exact same order. The "approximate" algorithm covered that problem by allowing for variance in the alignment and, therefore, refining the group of users that provided a recommendation. As a result, the "approximate" algorithm generally performs better than all other algorithms.

Further, it is apparent that the data set attributes of order and convergence are highly correlated to the success rate of prediction. Using the weighted approximate value as a comparison, the correlation to order is 0.83 and the correlation to convergence is -0.76. Further, the two data sets that were the least manipulated and contained the longest span of data were clearly more predictable than the heavily manipulated data sets that covered merely one day of activity.

4.5. NEIGHBORHOOD RESULTS

In large populations, the processing required to compare every user to every other user is insurmountable. A proposed method of reducing the processing requirements is to create a neighborhood, initially of a random set of users, and then attempt to improve the neighborhood over time. Table 4.6 compares the results of using a dynamic neighborhood to an optimal neighborhood. No neighborhood (technically, the entire population is a neighborhood) is shown because these tests were only performed on users with at least one hundred resource selections, making a direct comparison to Table 4.5 unreliable. To further demonstrate the improvement of the neighborhood over time, the results of the first fifty resource selections, per user, are shown separately. The first fifty are poor, indicating that later recommendations must be far more accurate.

Because the dynamic neighborhood method is dependent on the results of a pseudo-random number generator, four tests were performed for each data set. The result shown in Table 4.6 is the average of the three tests with the least difference between them, ignoring the fourth result as an outlier. It is apparent that the proposed dynamic method of building a neighborhood of similar users quickly reaches (and sometimes exceeds) the accuracy of the most optimal neighborhood. Accuracy is not the goal of the dynamic method. Runtime reduction is the goal. Table 4.7 lists the average milliseconds per user-to-user comparison per data set. Then, the time

Table 4.6: Neighborhood test results. Absolute Percent Success (Weighted Percent Success).

Set	None	Optimal	Dynamic	First 50	Outcome-Based
A	1% (0%)	2% (1%)	2% (1%)	1% (0%)	4% (2%)
E	68% (36%)	94% (40%)	86% (23%)	67% (11%)	95% (44%)
L	31% (11%)	33% (19%)	31% (17%)	25% (12%)	33% (19%)
M	78% (32%)	90% (76%)	90% (72%)	63% (61%)	93% (79%)
N	9% (1%)	9% (5%)	9% (5%)	6% (2%)	9% (5%)
V	0% (0%)	6% (3%)	0% (0%)	0% (0%)	12% (5%)
W	6% (0%)	6% (1%)	5% (1%)	0% (0%)	7% (1%)
X	0% (0%)	15% (8%)	9% (6%)	0% (0%)	15% (7%)

to compare to all users, the optimal neighborhood solution, is listed alongside the time to compare forty users, the twenty used in the neighborhood for the dynamic tests plus the twenty random users checked to improve the neighborhood. Having a linear runtime complexity, the dynamic neighborhood model completes in a fraction of the time required for the optimal neighborhood calculation. Therefore, any loss in accuracy is countered by an extreme savings in time.

Table 4.7: Runtime results for optimal vs. dynamic neighborhood construction.

Set	Compare Time	Optimal	Dynamic
A	22ms	408s	0.88s
E	30ms	386s	1.20s
L	178ms	12176s	7.12s
M	1ms	0s	0.04s
N	81ms	37553s	3.24s
V	1ms	1s	0.04s
W	82ms	3741s	3.28s

4.6. OUTCOME-BASED NEIGHBORHOOD RESULTS

Constructing a neighborhood of similar users in a dynamic, yet random, method has been shown to decrease runtime to a far greater degree than it decreases accuracy. To further reduce runtime while purposely increasing accuracy, the outcomes of each round of recommendation is used. Neighborhood users who make a correct suggestion are maintained in the neighborhood, regardless of similarity. The next suggestion for each of these users is the next resource selected by the user.

Table 4.6 compares the optimal neighborhood method to the outcome-based neighborhood method. It is clear that the inclusion of outcome-feedback improves accuracy. By avoiding similarity and alignment calculations, runtime is improved.

4.7. SUMMARY

This section covered the testing methods and results used to compare the proposed proactive search engine to many existing recommendation algorithms. The tests were divided into multiple steps to test each part of the proposed method. The approximate alignment algorithm performed better than other common recommendation algorithms. The dynamic neighborhood model ran much faster than calculating an optimal neighborhood, but did not significantly decrease accuracy of recommendations. Adding outcome feedback to the dynamic neighborhood model made up for the loss of accuracy while further reducing overall computation.

The test results in this section clearly support the claim that the proactive search engine described in Section 3 will be capable of providing a viable list of resources to a search engine user before the user enters a query, assuming that the user has a history of previously selected resources.

While performing relatively well, compared to other algorithms, the absolute performance was poor. In these tests, the source data contained very little longitudinal information per user and was clearly manipulated by whomever created the data set. Without longitudinal data, it is not reasonable to expect any recommendation algorithm to perform well. Depending on how the data was manipulated, the ability to perform recommendation may have been affected. With these factors in mind, the proactive search engine algorithm should have a better performance if given access to all of the data, unfiltered, for a real-world search engine.
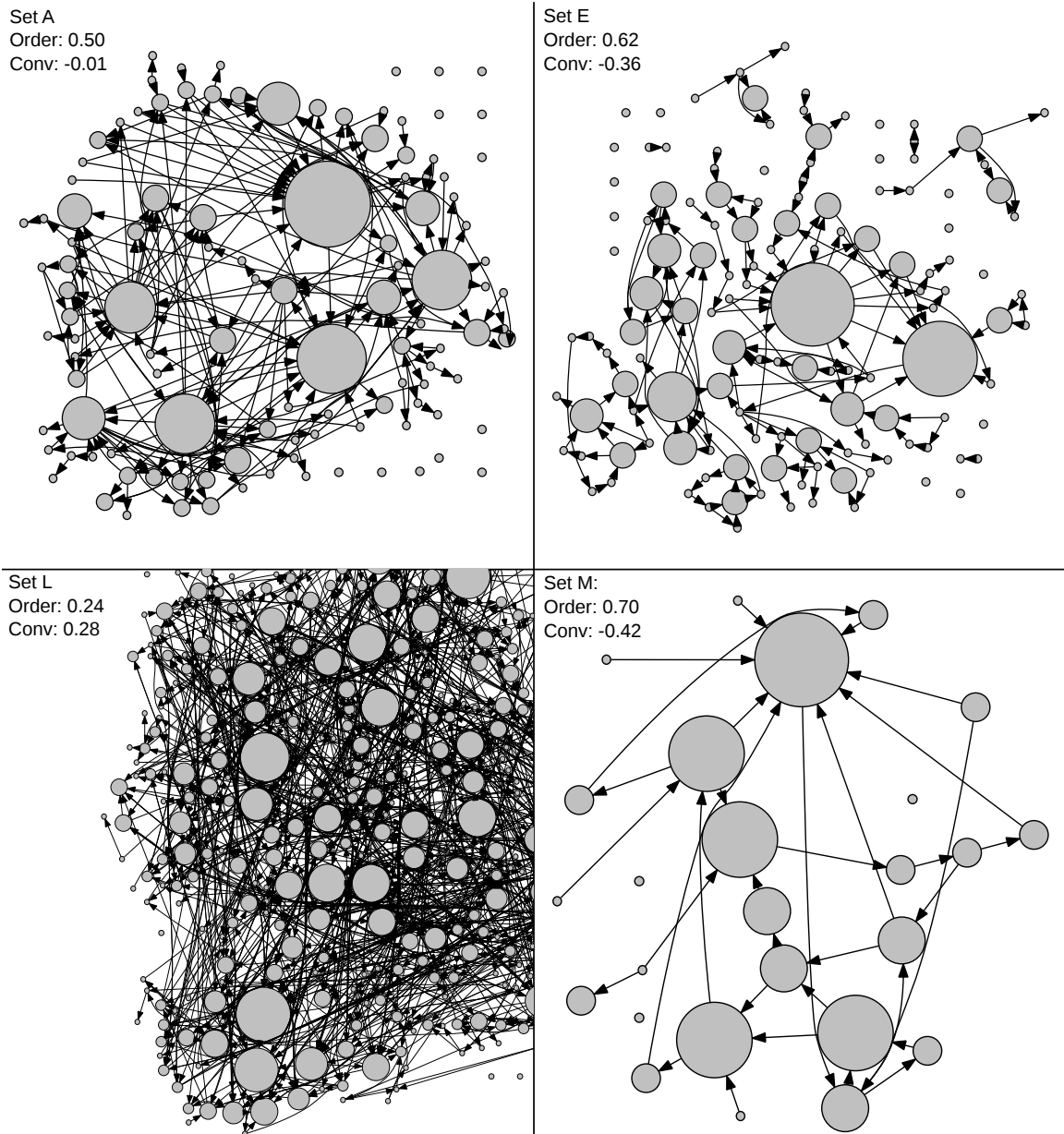
Figure 4.1: A graphical representation of the most popular resources in four test sets.

5. PREDICTIVE SEARCH ENGINE INTERFACE

5.1. INTRODUCTION

Currently, search interfaces require the user to enter some form of a query. Once a query has been entered, a set of search results are shown. There is a push to decrease the amount of time between entering a query and receiving results. Google Instant, for example, starts showing results after each word of a query is entered [45].

User histories are not ignored by search engines. Currently, user histories are used to locate and push advertisements to users, not only in the search engine, but across many affiliated websites. As shown in Figure 5.1, user histories may be used to identify what a user wants to search for before a query is entered. This reduces search time in two ways. First, the user is not required to enter a query, assuming prediction succeeds. Second, the user may be looking for a resource that normally is not found until later in the search process. Seeing a sequence of upcoming predictions, the user can skip ahead to the resource of interest.

5.2. MODULAR RESOURCES

Information resources are currently treated as complete packages. Web pages, for example, are self-contained sources of complete information. The completeness of web pages works well with modern search engines. The search engines prefer each resource to be a complete source of information. The web pages get a higher listing with more content per page.

A transition to sequences of web pages will break the cycle of padding independent pages with more and more information. Instead of being independent sources of

complete information, web pages will be modular segments of a sequence of information sources. Search engines will prefer modular sources that fit well into sequences. Web pages will be refined to get higher ranking in search engines.

5.3. INFORMATION SEQUENCES

Outside of web search engines, the modularity of information may be applied without a complete overhaul of how information is currently developed and stored. School courses are currently semester-long. The information for a course is developed and treated as a semester's worth of information. Many of the topics are covered in multiple courses. As an example, truth tables show up in computer programming, digital logic, and critical thinking.

By treating courses as a semester-long sequence of modular information resources, truth tables could be a single module that appears in three different sequences. Instead of having three separate instructors develop slides, notes, and tests on truth tables, only one good truth table module is necessary and everyone can share and improve the resource.

With course modules in mind, the development of courses also benefits from a predictive search engine. The search engine identifies common sequences between two modules. By giving the search engine a few specific modules that must be covered through the semester, the search engine can fill in the gaps between those modules with the most common sequences of other modules. In an instant, a popular sequence of modules is developed, complete with accompanying media.

5.4. SUMMARY

The proactive search engine is designed to decrease the user's time spent using a search engine while increasing the probability that a user will quickly find an electronic resource of interest. If implemented, the nature of the search engine interface will not be heavily impacted. However, the nature of the resources will change from self-contained resources to modular resources. As modules, the resources will be used as elements of ordered sequences.

While users are trained to use search engines to locate a single resource, working with sequences does provide multiple benefits. A user could search for the common modules that fill in a sequence between a starting and ending resource. This could be used to fill in the topics for a presentation that normally occur between two major points. A user could identify a starting point and view the common sequences that follow. A user may be trying to find the solution to a printer problem. Seeing the point where other users tend to end will allow the user to quickly fast-forward to the solution. In time, the use of sequences will displace the concept of independent resources.

Figure 5.1: Instead of pushing ads, user histories may be used to predict which pages a user will visit next.

6. CONCLUSION AND FUTURE RESEARCH DIRECTION

Search engines are necessary tools to navigate modern electronic repositories. A common research goal is to improve runtime and accuracy of search engines. This research demonstrates that it is possible to proactively predict which resource a specific user will select based on a collaborative comparison between that user's recent search history and the histories of all other users.

A recommendation method based on approximate alignment and similarity with respect to order has been demonstrated, using real-world search engine data, to provide a more accurate recommendation than common recommendation algorithms. Performing the proposed collaborative recommendation is not viable for large populations of users due runtime complexity. Two methods have been proposed to attack the runtime complexity problem.

Building a neighborhood in a dynamic fashion reduces the overall runtime per recommendation by a factor of about 1,000. As expected, early recommendations are poor. Quickly, recommendations improve and, within one hundred recommendations, the dynamic method is comparable to using an optimal neighborhood for each recommendation.

To further reduce overall runtime while improving accuracy, outcome-feedback has been proposed. With outcome-feedback included, the dynamic neighborhood method meets and often exceeds the use of an optimal neighborhood.

This research has demonstrated that it is often possible to accurately predict which resource a search engine user will select before the user enters a search query. A search engine may proactively display a set of recommended resources to the user, without requiring a search query. If the user select a resource from the recommended list, overall search time is reduced as the query-response tasks are bypassed.

Continued work in this area of research will be heavily limited by the absence of real-world search engine user data. Moreover, to fully demonstrate the effectiveness of the proposed proactive search technique, it must be implemented on a real-world search engine.

Implementation should not be an intrusive task as it may be used for a small set of test users. Each test user will have a neighborhood building agent that scans the population of all users as a low priority process. When a test user accesses the search engine, the neighborhood may be scanned quickly to provide a list of recommendations. Then, assuming that the real-world test is successful, more users may be included in the test.

BIBLIOGRAPHY

[1] Arvind Rangaswamy, C. Lee Giles, and Silvija Seres. A strategic perspective on search engines: Thought candies for practitioners and researchers. *Journal of Interactive Marketing*, 23(1):49–60, February 2009.

[2] Mark Levene. *An Introduction to Search Engines and Web Navigation*. John Wiley & Sons, 2 edition, 2010.

[3] David Hawking, Nick Craswell, Peter Brailey, and Kathleen Griffihs. Measuring search engine quality. *Information Retrieval*, 4(1):33–59, April 2001.

[4] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, January 2003.

[5] Sergio Cleger-Tamayo, Juan M. Fernández-Luna, Juan F. Huete, Ramiro Pérez-Vázquez, and Julio C. Rodríguez Cano. A proposal for news recommendation based on clustering techniques. In *Trends in Applied Intelligent Systems*, volume 6098 of *Lecture Notes in Computer Science*, pages 478–487. Springer Berlin/Heidelberg, 2010.

[6] Matthew Brand. Fast online svd revisions for lightweight recommender systems. In *Proceedings of the Third SIAM International Conference on Data Mining*, pages 37–46, 2003.

[7] Zhong Su, Qiang Yang, Ye Lu, and Hongjiang Zhang. Whatnext: a prediction system for web requests using n-gram sequence models. In *Proceedings of the First International Conference on Web Information Systems Engineering, 2000*, volume 1, pages 214–221, June 2000.

[8] Tomer Toledo and Romina Katz. State dependence in lane-changing models. *Transportation Research Record: Journal of the Transportation Research Board*, 2124:81–88, 2009.

[9] Alex Pentland and Andrew Liu. Modeling and prediction of human behavior. *Neural Computation*, 11(1):229–242, January 1999.

[10] B. F. Skinner. *The Behavior of Organisms*. Copley Publishing Group, 1938.

[11] Dirk Lewandowski. Search engine user behaviour: How can users be guided to quality content? *Information Services and Use*, 28(3-4):261–268, August 2008.

[12] Michael T. Jones, Brian McClendon, Amin P. Charaniya, and Michael Ashbridge. Entity display priority in a distributed geographic information system. `http://www.google.com/patents/US20070143345`, June 2007.

[13] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, January 1967.

[14] Rui Xu and Donald C. Wunsch II. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, May 2005.

[15] Michael Mukiibi and James O. Bukenya. Segmentation analysis of grocery shoppers in alabama. In *The Southern Agricultural Economics Association Annual Meeting*, February 2008.

[16] Joyce John. Pandora and the music genome project. *Scientific Computing*, 23(10):40–41, September 2006.

[17] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42:30–37, August 2009.

[18] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Scalable distributed algorithm for approximate nearest neighbor search problem in high dimensional general metric spaces. In *Proceedings of the 5th International Conference on Similarity Search and Applications*, pages 132–147, Heidelberg, 2012. Springer.

[19] Daniel Billsus and Michael J. Pazzani. Learning collaborative information filters. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 46–54, 1998.

[20] Lei Xiong, Yang Xiang, Qi Zhang, and Lili Lin. A novel nearest neighborhood algorithm for recommender systems. In *Intelligent Systems (GCIS), 2012 Third Global Congress on*, pages 156–159, November 2012.

[21] David Bremner, Erik Demaine, Jeff Erickson, John Iacono, Stefan Langeman, Pat Morin, and Godfried Toussaint. Output-sensitive algorithms for computing nearest-neighbour decision boundaries. *Journal of Discrete & Computational Geometry*, 33(4):593–604, April 2005.

[22] Robert M. Bell and Yehuda Koren. Lessons from the netflix prize challenge. *ACM SIGKDD Exlorations Newsletter*, 9:75–79, December 2007.

[23] Pan-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Pearson Addison-Wesley, 1 edition, 2005.

[24] Scott Deerwester, Susan T. Dumais, George W. Fumas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[25] Gene H. Golub and William Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics*, 2(2):205–224, 1965.

[26] Manolis Vozalis and Konstantinos G. Margaritis. Using svd and demographic data for the enhancement of generalized collaborative filtering. *Journal of Information Sciences: An International Journal*, 177(15):3017–3037, August 2007.

[27] Taffee T. Tanimoto. *An Elementary Mathematical Theory of Classification and Prediction*. IBM Internal Report, 1957.

[28] Paul Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bulletin del la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.

[29] Thorvald Sørensen. A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analysis of the vegetation on danish commons. *Biologiske Skrifter*, 5:1–34, 1948.

[30] Richard W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

[31] Dan Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1 edition, 1999.

[32] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10:707–710, 1966.

[33] Thomas H. Cormen, Charleston E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3 edition, 2009.

[34] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 3 edition, 2007.

[35] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, January 1974.

[36] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, March 1970.

[37] Waterman MS Smith TF. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[38] David J. Lipman and William R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441, March 1985.

[39] Lloyd Allison. Lazy dynamic-programming can be eager. *Information Processing Letters*, 43(4):207–212, September 1992.

[40] George K. Zipf. The psycho-biology of language. *Language*, 12:196–210, July 1936.

[41] Jianfeng Gao, Joshua T. Goodman, and Jiangbo Miao. The use of clustering techniques for language modeling – application to asian languages. *Computational Linguistics and Chinese Language Processing*, 6(1):27 – 60, February 2001.

[42] Duygu Tumer, Mohammad Ahmed Shah, and Yiltan Bitirim. An empirical evaluation on semantic search performance of keyword-based and semantic search engines: Google, yahoo, msn and hakia. In *Proceedings of the 2009 Fourth International Conference on Internet Monitoring and Protection*, pages 51–55. IEEE Computer Society, 2009.

[43] Javed Mostafa. Seeking better web searches. *Scientific American*, 292(2):66–73, February 2005.

[44] Aaron M. Cohen, Clive E. Adams, John M. Davis, Clement Yu, Philip S. Yu, Weiyi Meng, Loma Duggan, Marian McDonagh, and Neil R. Smalheiser. Evidence-based medicine, the essential role of systematic reviews, and the need for automated text mining tools. In *Proceedings of the 1st ACM International Health Informatics Symposium*, IHI '10, pages 376–380. ACM, November 2010.

[45] About google instant. `http://www.google.com/instant/`, November 2010.

[46] C. Shaun Wagner, Sahra Sedigh, Ali R. Hurson, and Behrooz Shirazi. A survey of techniques for improving search engine scalability through profi ling, prediction, and prefetching of query results. In Samee U. Khan, Albert Y. Zomaya, and Lizhe Wang, editors, *Scalable Computing and Communications: Theory and Practice*, chapter 23, pages 467–506. Wiley-IEEE Computer Society Press, New Jersey, January 2013.

[47] C. Shaun Wagner, Sahra Sedigh, and Ali R. Hurson. Accurate and efficient search prediction using fuzzy matching and outcome feedback. In Nieves Brisaboa, Oscar Pedreira, and Pavel Zezula, editors, *Similarity Search and Applications*, volume 8199 of *Lecture Notes in Computer Science*, pages 219–232. Springer Berlin Heidelberg, 2013.

[48] C. Shaun Wagner, Sahra Sedigh, and Ali R. Hurson. Outcome-based dynamic k-nearest neighbors models for search engine prediction. Journal of Information Processing (to appear), 2014.

[49] C. Shaun Wagner, Sahra Sedigh, and Ali R. Hurson. Proactive search: Using outcome-based dynamic nearest-neighbor recomendation algorithms to increase search engine efficacy. ACM Transactions on Information Systems (submitted), 2014.

VITA

C. Shaun Wagner was born in St. Louis, Missouri and raised in north Kansas City, Missouri. After graduating from Platte City High School, he served five years in the United States Marine Corps as a radar controller engineer, maintaining digital circuits and interface programs. As a top graduate of his electronics school, he was selected as an instructor for the following term.

After his military service, Shaun began taking classes at a local university, the College of Charleston (CofC). He helped publish a paper on classification of music. He taught K-5 classes at a local elementary school. He received his B.S., with honors, in Computer Science from the CofC in 1993.

After receiving his B.S., he began a career in health informatics research for the Hypertension Initiative at the Medical University of South Carolina (MUSC), where he helped author many publications. The Hypertension Initiative outgrew MUSC and became the Care Coordination Institute (CCI). Shaun is the senior data architect at CCI, responsible for identifying, organizing, and analyzing health information for many forms of research.

While working for the Hypertension Initiative (and later the CCI), Shaun entered the graduate program at the Missouri University of Science & Technology. Before identifying a specific area of research, he taught classes at the university for three semesters. In his research, he published a background survey as a book chapter [46], proposed and published his concept for a proactive search engine at a conference [47], authored an invited paper detailing the initial results of his research [48], and submitted the final results of his work for publication [49].