Under consideration for publication in Knowledge and Information Systems

Boosted Ranking Models: A Unifying Framework for Ranking Predictions

Kevin Dela Rosa [1], Vangelis Metsis [2], and Vassilis Athitsos [2]

[1] Language Technologies Institute, Carnegie Mellon University, Pittsburgh PA, USA
[2] Computer Science and Engineering Department, University of Texas at Arlington, Arlington TX, USA

Abstract. Ranking is an important functionality in a diverse array of applications, including web search, similarity-based multimedia retrieval, nearest neighbor classification, and recommendation systems. In this paper we propose a new method, called Boosted Ranking Model(BRM), for learning how to rank from training data. An important feature of the proposed method is that it is domain-independent, and can thus be applied to a wide range of ranking domains. The main contribution of the new method is that it reduces the problem of learning how to rank to the much more simple, and well-studied, problem of constructing an optimized binary classifier from simple, weak classifiers. Using that reduction, our method constructs an optimized ranking model using multiple simple, easy-to-define ranking models as building blocks. The new method is a unifying framework that includes, as special cases, specific methods that we have proposed in earlier publications for specific ranking applications, such as nearest neighbor retrieval and classification. In this paper we reformulate those earlier methods as special cases of the proposed BRM method, and we also illustrate a novel application of BRM, on the problem of making movie recommendations to individual users.

Keywords: Ranking models, learning, boosting, recommendation systems

1. Introduction

Ranking systems are used to automatically rank a set of items, according to specified criteria. Ranking is an integral part of a diverse array of applications, including web search, similarity-based multimedia retrieval, nearest neighbor classification, and recommendation systems for movies, books, and other shopping items. Consequently, automated methods that can learn ranking models from training

Received Jun 08, 2010
Revised Nov 04, 2010
Accepted Jan 30, 2011

data are of interest to several research communities, and can lead to benefits for the large numbers of computer users that use automated ranking systems (such as Google search, Netflix movie recommendations, or Amazon shopping recommendations) on a daily basis.

In some cases, the ranking criteria are absolute, and known in advance to the designers of the ranking system. For example, we may want to identify the most "important" web pages or the "strongest" football teams according to some strict, quantitative definitions of "important" and "strong." At the same time, in many cases, the ranking criteria are not fixed, and are not known to the designer of the ranking system in advance. This is the case, for example, for similarity-based multimedia retrieval, where the ranking of the retrieved items depends on the similarity to the query item. Another example is a movie recommendation system, that needs to provide recommendations catered specifically for each user.

In this paper we propose a novel method, called Boosted Ranking Model (BRM), for learning ranking models from training data. We address the general case where the ranking criteria are not fixed and are not known a priori. At the core of our method lies a reduction of the problem of learning how to rank to the more simple and well-studied boosting problem, i.e., the problem of combining weak binary classifiers into a strong binary classifier. The advantage of that reduction is that it allows us to use well-developed existing boosting algorithms for learning ranking models.

Following the boosting paradigm, the proposed BRM method combines multiple simple, easy-to-define "weak" models into a single, optimized model. Defining such "weak" models is a domain-specific task. However, after such weak models have been defined, our method can be applied in a domain-independent way, to construct an optimized combination of the weak models.

The proposed BRM method is a unifying framework that encapsulates, as special cases, previous methods that we have proposed in earlier work for special-purpose ranking problems such as nearest neighbor retrieval (Athitsos, Alon, Sclaroff and Kollios, 2008; Athitsos, Hadjieleftheriou, Kollios and Sclaroff, 2007) and nearest neighbor classification (Athitsos and Sclaroff, 2005). In this paper we show how those prior methods can be readily derived as instances of the BRM method. In addition, we describe a novel application of BRM, to the problem of making movie recommendations for users using the Netflix dataset (Bennett, Elkan, Liu, Smyth and Tikk, 2007).

The remainder of the paper is organized as follows: Section 2 goes over related work. Section 3 formally defines the problem we address in this paper. Section 4 describes the reduction of the problem of learning how to rank to a classical boosting problem, and describes how to adapt the AdaBoost algorithm (Schapire and Singer, 1999) to the problem of learning a ranking model. Section 5 describes a somewhat more complicated variation of our method, that allows for the construction of more informative ranking models. Section 6 shows how to derive, as special cases of the proposed framework, some earlier methods that we have proposed for nearest neighbor retrieval and classification. Section 7 shows how to apply the proposed framework to the problem of making movie recommendations. In Section 8 we provide results obtained by applying the proposed method on the Netflix dataset (Bennett et al., 2007).

2. Related Work

As ranking problems appear in a variety of research areas and applications, a large body of work is related to the topic of constructing ranking models. A popular application area for ranking methods is the area of recommendation systems, i.e., systems that make recommendations for the most appropriate actions/items for a specific user or situation. Recommendation systems can be used to recommend to users a diverse range of items, including movies (Koren, 2008), commercial products (Bridge, 2001), useful customer reviews (Zhang and Tran, 2010), images (Kim, Lee, Cho and Kim, 2004), or music (Chen and Chen, 2005).

Collaborative filtering approaches (Goldberg, Nichols, Oki and Terry, 1992) make recommendations based on previous user behavior patterns, i.e., based on ratings/rankings that users have previously assigned to various items. Collaborative filtering for recommendation systems is a topic that has attracted significant attention in recent years. Surveys of existing methods can be found in (Adomavicius and Tuzhilin, 2005; Schafer, Frankowski, Herlocker and Sen, 2007).

One common approach for collaborative filtering is neighborhood models (Bell, Koren and Volinsky, 2007; Bezerra and Carvalho, 2010; Toescher, Jahrer and Legenstein, 2008; Vucetic and Obradovic, 2005), where recommendations are based on choices made by users similar to a specific user, or choices made by people who used, rated, or purchased a specific item (Linden, Smith and York, 2003; Sarwar, Karypis, Konstan and Riedl, 2001). In (Koren, 2009), the neighborhood model is expanded to include temporal dynamics, that model how movie ratings change over time. Another common approach is the use of SVD-based or SVD-inspired factorization models (Gantner, Drumond, Freudenthaler, Rendle and Schmidt-Thieme, 2010; Paterek, 2007; Wu, 2009; Zhou, Wilkinson, Schreiber and Pan, 2008). In factorization models, each user and each item is assigned a low-dimensional vector, and the rating of a user on an item is approximated by the dot product of the vectors assigned to the user and the item. Hybrid methods have also been proposed, that combine neighborhood models and factorization models (Koren, 2008; Takács, Pilászy, Németh and Tikk, 2009). Neural networks have also been used for collaborative filtering, in the form of Restricted Boltzmann Machines (Salakhutdinov, Mnih and Hinton, 2007). Extensions of collaborative filtering algorithms have been proposed for distributed systems (Becchetti, Colesanti, Marchetti-Spaccamela and Vitaletti, 2010).

The Netflix prize (Bennett et al., 2007) has motivated a significant amount of novel research in collaborative filtering for recommendation systems, e.g., (Bell and Koren, 2007; Paterek, 2007; Toescher et al., 2008). An important difference between the problem definition in the Netflix prize and the problem we are addressing in this paper is that in the Netflix contest the goal is to predict ratings, whereas in this paper our goal is to predict rankings. Predicting rankings is a relaxed version of the rating problem: while ratings specify rankings, rankings do not specify ratings. The relaxation of the rating estimation problem into a ranking estimation problem allows us to develop a relatively simple optimization method based on boosting binary classifiers.

A key feature of the winning entry for the Netflix prize was the utilization of multiple models, including neighborhood models, factorization models, and Restricted Boltzmann Machines (Bell and Koren, 2007), based on the realization that each of those methods has its own strengths and weaknesses, and that combined together these methods can complement each other and lead to improved accuracy. The proposed BRM method is designed exactly to facilitate the task of combining multiple domain-specific methods. Our training algorithm takes as input different, previously constructed models, and combines them into an optimized hybrid model. An attractive feature of the algorithm is that it is oblivious to the structure and assumptions underlying each of the input models, thus making it easy to combine heterogeneous models together.

Another area where ranking is an important functionality is the area of similarity-based indexing, and in particular nearest neighbor retrieval. In nearest neighbor retrieval, we are typically given a database of objects and a specific distance measure for comparing two objects. At runtime, given the query object, the goal is to retrieve the most similar database objects. There are two important aspects of the nearest neighbor retrieval problem that differentiate it from the recommendation system problem. One such aspect is that, in nearest neighbor retrieval, the system can always compute the ground truth using brute-force search, i.e., by simply measuring the distance between the query object and each database object. The goal of indexing methods is to efficiently identify the nearest neighbors for each query, so as to spend significantly less time than brute-force search would require.

A second differentiating aspect of the nearest neighbor retrieval problem is the availability, in many cases, of geometric structure that can be exploited for efficient indexing. Most existing indexing methods do not view nearest neighbor indexing as a learning problem, but instead use geometric properties to speed up retrieval. Reviews of literature on geometry-based indexing methods can be found at (Böhm, Berchtold and Keim, 2001; Hjaltason and Samet, 2003*b*; Hjaltason and Samet, 2003*a*; White and Jain, 1996). In particular, indexing methods can explicitly exploit properties of Euclidean and L_p spaces, e.g., (Andoni and Indyk, 2006; Chen, Liu, Furuse, Yu and Ohbo, 2010; Gionis, Indyk and Motwani, 1999; Kanth, Agrawal and Singh, 1998; Kim, Chung, Lee and Kim, 2010; Li, Chang, Garcia-Molina and Wiederhold, 2002; Sakurai, Yoshikawa, Uemura and Kojima, 2000; Tuncel, Ferhatosmanoglu and Rose, 2002; Weber and Böhm, 2000; Weber, Schek and Blott, 1998), or the triangle inequality, e.g., (Aronovich and Spiegler, 2010; Bozkaya and Özsoyoglu, 1999; Ciaccia, Patella and Zezula, 1997; Hjaltason and Samet, 2003*a*; Traina, Traina, Seeger and Faloutsos, 2000; Uhlman, 1991; Yianilos, 1993; Zhang and Alhajj, 2010).

While the majority of existing indexing methods explicitly exploit geometric properties, it is also possible to view nearest neighbor indexing as a learning problem, where the system can learn how to produce approximate nearest neighbor rankings for each query, using some information about the query. In prior work we have proposed the BoostMap method (Athitsos et al., 2008; Athitsos et al., 2007) as a learning method for constructing an indexing structure. In (Athitsos and Sclaroff, 2005) we proposed a related ranking method, where the goal was not to build an indexing structure, but to learn a distance measure so as to optimize nearest neighbor classification accuracy. Compared to those earlier methods, that explicitly targeted the problem of nearest neighbor rankings, the BRM method described in this paper is a general, domain-independent framework, that provides a unified approach for addressing disparate, special-purpose ranking problems such as recommendation systems, nearest neighbor indexing, and nearest neighbor classification.

We should note that our previous special-purpose ranking methods (Athitsos et al., 2008; Athitsos et al., 2007; Athitsos and Sclaroff, 2005), which were designed for nearest neighbor rankings, only addressed the special case where the query is an object in some space and the database items are objects from the same space. The BRM method described here is of significantly more general scope. In BRM, the query is not constrained to be an object from the same space as the database items. Instead, the query can be any arbitrary ranking criterion, according to which database items should be ranked. For example, in the Netflix dataset, the database items are movies, whereas each query (or, synonymously, ranking criterion) corresponds to an individual user: given a specific user, we want to rank movies in order of estimated preference by the user. Thus, in the Netflix dataset, as in many other recommendation system applications, queries and database items belong to clearly distinct spaces. The BRM method can naturally handle such cases, whereas in those same cases our previous work is not applicable.

A method closely related to the proposed BRM method is RankBoost (Freund, Iyer, Schapire and Singer, 2003). In both RankBoost and BRM, boosting is used to learn how to rank a set of items. An important difference between RankBoost and BRM is that the RankBoost training algorithm learns a single ranking function, tailored to a specific set of ranking criteria. For example, consider a movie recommendation system, where the ranking criterion is simply the identity of the user for whom the system is making the recommendations. If we want to use RankBoost to produce a customized ranking function for each individual user, we would have to invoke the RankBoost training algorithm separately for each individual user. Similarly, in a nearest neighbor retrieval system, where efficiency is a key measure of performance, the RankBoost training algorithm would have to be invoked online, for each individual query.

In contrast to RankBoost, the proposed BRM method does not learn a single ranking function, but rather a global ranking model, which produces a customized ranking function for any set of ranking criteria. For example, in a movie recommendation system, after the global ranking model has been learned offline, this ranking model can readily produce a customized ranking function for any user, including users for who no data was available during training. Similarly, in a nearest neighbor retrieval system, given a previously unseen query object, the global ranking model can readily produce an approximate ranking of database objects for that query. Consequently, whereas a RankBoost-based system needs to perform training online, for every previously unseen query that is submitted to the system, a BRM-based system can perform all training off-line, thus requiring fewer computational resources during online operation.

Another method closely related to BRM is GBRank (Zheng, Chen, Sun and Zha, 2007). GBRank has a problem formulation similar to ours: given a query Q, the goal is to obtain accurate rankings of a set of database objects. At the same time, GBRank and BRM have some important differences. GBRank assumes that, given a query, each query-object pair can be represented as a vector, by extracting some features from the pair. BRM makes the more general assumption that some arbitrary scoring functions have been defined (which the BRM method combines into an optimized scoring function), and those arbitrary scoring functions do not have to rely on a vector representation of pairs of queries and database objects. Thus, BRM is general enough to encompass problems such as similarity-based indexing in non-vector spaces, as described in Sections 6.1 and 6.2, whereas GBRank is not applicable on such problems. Another difference is that our formulation, as described in Section 5, includes a domain-independent way to define query-sensitive strong classifiers, where the weight of each weak classifier varies depending on the query. In GBRank, the strong classifier assigns a fixed weight to each weak classifier.

3. Basic Definitions and Problem Formalization

The **ranking problem** can be described as follows: Let $\mathbb{U}$ be a finite set of items, that in this paper we also call a **database**. Let $\mathbb{Q}$ be a (possibly infinite) set of queries that can be submitted to the system. Every time we submit a query $Q \in \mathbb{Q}$, the response of the system is a *ranking* of all items in $\mathbb{U}$ according to the query. A **ranking model** is defined to be a function $R : \mathbb{Q} \times \mathbb{U} \to \{1, 2, \ldots, \|\mathbb{U}\|\}$, where $\|\mathbb{U}\|$ denotes the number of items in $\mathbb{U}$. This ranking model specifies that $R(Q, U)$ is the rank of item U under query Q, and $R(Q, U)$ is a number between 1 and $\|U\|$. The **highest rank** (corresponding to the most relevant item) is 1, and the **lowest rank** is $\|U\|$.

As an example, consider a similarity-based image retrieval system (e.g., QBIC (Flickner, Sawhney, Niblack, Ashley, Huang, Dom, Gorkani, Hafner, Lee, Petkovic, Steele and Yanker, 1995)), where given an image as a query we want to rank database images in order of similarity to the query. In that case, $\mathbb{U}$ is the set of database images, and $\mathbb{Q}$ is the set of possible query images. As another example, consider the Netflix domain (Bennett et al., 2007), where we want to rank movies according to the preferences of a specific user. In that case, $\mathbb{U}$ is the set of movies, and $\mathbb{Q}$ is the set of users.

We assume that there exists a single correct ranking model R_{true}, that specifies a "true" rank $R_{\text{true}}(Q, U)$ for every database item U given any query Q. Intuitively, our goal is to build a system whose rankings are as close as possible to the rankings produced by R_{true}. We also assume that, at training time, we have some partial information about R_{true}, in the following form: we are given a training set of triples $\mathbb{T}_{\text{train}} = \{(Q_1, A_1, B_1), \ldots, (Q_t, A_t, B_t)\}$ such that:

- Each Q_i is a query sampled from $\mathbb{Q}$.
- Each A_i and each B_i is an element of the set $\mathbb{U}$ whose items we want to rank.
- $R_{\text{true}}(Q_i, A_i) < R_{\text{true}}(Q_i, B_i)$. That is, given query Q_i, the "true" rank of A_i is higher (A_i is ranked as more important) than the rank for B_i.

Using such training data, we want to learn a model R that approximates R_{true}. To measure how well R approximates R_{true}, we evaluate the performance of R on some test set $\mathbb{T}_{\text{test}}$ of triples, with the constraint that if (Q, A, B) is a test triple, then Q did not appear in any of the training triples. We consider that R *fails* on triple (Q, A, B) if, given query Q, the relative ranking of A and B according to R is different than the relative ranking of A and B according to R_{true}. In other words, R fails on (Q, A, B) if either $R(Q, A) > R(Q, B)$ and $R_{\text{true}}(Q, A) < R_{\text{true}}(Q, B)$, or $R(Q, A) < R(Q, B)$ and $R_{\text{true}}(Q, A) > R_{\text{true}}(Q, B)$. Then, the **error rate** $E(R)$ of the learned ranking model R is defined as a percentage of test triples on which R fails.

Using mathematical notation, error rate $E(R)$ can be defined as follows:

$$E(R, Q, A, B) = \begin{cases} 1 & \text{if } R(Q,A) > R(Q,B) \text{ and } R_{\text{true}}(Q,A) < R_{\text{true}}(Q,B), \\ 1 & \text{if } R(Q,A) < R(Q,B) \text{ and } R_{\text{true}}(Q,A) > R_{\text{true}}(Q,B), \\ 0.5 & \text{if } R(Q,A) = R(Q,B) \text{ and } R_{\text{true}}(Q,A) < R_{\text{true}}(Q,B), \\ 0.5 & \text{if } R(Q,A) = R(Q,B) \text{ and } R_{\text{true}}(Q,A) > R_{\text{true}}(Q,B), \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

$$E(R) = \frac{\sum_{(Q,A,B)\in\mathbb{T}_{\text{test}}} E(R,Q,A,B)}{\|\mathbb{T}_{\text{test}}\|} , \tag{2}$$

where $\|\mathbb{T}_{\text{test}}\|$ is the number of elements of $\mathbb{T}_{\text{test}}$.

In the above definitions we ignore ties in the true ranks, i.e., cases where $R_{\text{true}}(Q,A) = R_{\text{true}}(Q,B)$. We should note that, in some domains, it is fairly common to encounter such cases. An example is the Netflix domain, where each user rates each movie with an integer from 1 to 5, and thus many movies receive the same rating. Ignoring ties in the definition of $E(R)$ simply means that, when $R_{\text{true}}(Q,A) = R_{\text{true}}(Q,B)$, then we do not care how R ranks A and B relative to each other for query Q. In our implementation, we exclude cases of ties both when we select training triples, used to construct a ranking model, and when we select test triples, used to evaluate a ranking model. While we ignore ties in the true ranks, we do not ignore ties in the estimated ranks, and we assign a "half error" to those cases.

The **rating problem**, or, synonymously, **scoring problem**, is closely related to the ranking problem. In the scoring problem, the goal of the system is to predict not rankings, but scores (ratings) corresponding to specific items given a query. For example, in the Netflix domain, the rating problem is the problem of predicting the rating of movies by users. Formally, a **rating model** or **scoring model** is a function $S : \mathbb{Q}\times\mathbb{U} \to \mathbb{R}$, where $S(Q,U)$ is defined as the score/rating of item U under query Q.

Clearly, any scoring model S defines an associated ranking model R^S, whereby items are ranked according to their scores. The opposite does not hold, as multiple (typically infinitely many) scoring models can produce the same ranking model. In that sense, we may consider that the ranking prediction problem is a relaxation of the scoring prediction problem.

An additional assumption that we make in our problem formulation is that we are given as input a family of scoring models, which someone (or ourselves) has already defined in a manner appropriate for the domain of interest. We use the term **weak scoring models** to refer to models in that family. Weak scoring models play in our formulation a role analogous to the role played by weak classifiers in boosting methods (Friedman, Hastie and Tibshirani, 2000; Schapire and Singer, 1999). Boosting methods assume that a family of weak classifiers has already been defined for a specific binary problem. A boosting method combines many such weak classifiers into an optimized classifier for the problem. Similarly, the proposed BRM method uses weak scoring models as building blocks for the construction of a ranking model R_{opt}, in a way that minimizes the error rate $E(R_{\text{opt}})$. As is typical in boosting methods, our method does not propose any domain-independent way of constructing a family of weak scoring models; that task is left as an implementation choice. Examples of families of weak scoring models for specific application domains are detailed in Sections 6 and 7.

Based on the above definitions, the problem we address in this paper can be defined as follows: We want to design a training algorithm that, given as input a training set of triples and a family of weak scoring models, constructs as output a scoring model S_{opt} and its associated ranking model R_{opt}, in a way that minimizes the error rate of R_{opt}. The next section describes our solution to this problem.

4. Boosted Ranking Model

In our problem definition, the goal is to combine multiple scoring models S_i into an optimized ranking model R_{opt}, in a way that minimizes the error rate $E(R_{\text{opt}})$. The main contribution of the proposed method is in reducing this problem to the well-studied problem of combining multiple weak binary classifiers into a single optimized binary classifier. The problem of binary classifier combination has been well studied in the machine learning community, and several solutions have been proposed following the "boosting" framework, such as AdaBoost (Schapire and Singer, 1999), LogitBoost (Friedman et al., 2000), or FloatBoost (Li and Zhang, 2004). In our implementation we have chosen to use AdaBoost, but this is simply an implementation choice, as any other general-purpose boosting method is also applicable.

In particular, the proposed Boosted Ranking Model (BRM) method for constructing an optimized ranking model R_{opt} from multiple scoring models S_i consists of the following steps:

1. Define, for each scoring model S_i, a corresponding binary classifier $\tilde{S}$ that is used to estimate, for any triple (Q, A, B), whether $R_{\text{true}}(Q, A) < R_{\text{true}}(Q, B)$ or $R_{\text{true}}(Q, A) > R_{\text{true}}(Q, B)$.
2. Run AdaBoost, using the provided training set of triples, to combine binary classifiers $\tilde{S}$ into an optimized binary classifier $\tilde{S}_{\text{opt}}$.
3. Define, using $\tilde{S}_{\text{opt}}$, a scoring model S_{opt}.
4. Define, using S_{opt}, the corresponding ranking model R_{opt}.

A key property in this process is the following: the error rate $E(R_{\text{opt}})$ on any set $\mathbb{T}$ of triples, measured as defined in Equation 2, is equal to the error rate of the classifier $\tilde{S}_{\text{opt}}$ constructed using AdaBoost on the same set $\mathbb{T}$. Consequently, by minimizing, using AdaBoost, the classification error of $\tilde{S}_{\text{opt}}$, we minimize the error rate $E(R_{\text{opt}})$, which is exactly the optimization criterion that we have set out to minimize.

In the remainder of this section we describe how to perform each of the above-mentioned steps.

4.1. From Scoring Models to Binary Classifiers

Let S be a scoring model mapping each (Q, U) pair of a query Q and an item U to a real number. Without loss of generality, we follow the convention that higher ratings indicate a stronger preference, and thus correspond to higher rankings, i.e., lower values of $R_{\text{true}}(Q, U)$. In the opposite case, where lower scores correspond to higher rankings, the only adjustment we need to make is to use the negation of those scores in our formulation, so that our convention still holds.

Let $\mathbb{T}$ be a set of triples (Q, A, B), defined as in the previous section, i.e., with Q being a query, and A, B being items from $\mathbb{U}$. We can define, using each scoring model S, a corresponding binary classifier $\tilde{S}$, that estimates, for every triple $(Q, A, B) \in \mathbb{T}$ whether $R_{\text{true}}(Q, A) < R_{\text{true}}(Q, B)$ or not. This classifier is defined as follows:

$$\tilde{S}(Q, A, B) = S(Q, A) - S(Q, B) \ . \tag{3}$$

The output of binary classifier $\tilde{S}$ is interpreted as follows:

– If $\tilde{S}(Q, A, B) > 0$, then $\tilde{S}$ assigns class label 1 to the triple (Q, A, B). We define **class label 1** as the class label corresponding to the case where $R_{\text{true}}(Q, A) < R_{\text{true}}(Q, B)$, meaning that, for query Q, item A is more highly ranked than item B.
– If $\tilde{S}(Q, A, B) < 0$, then $\tilde{S}$ assigns class label -1 to the triple (Q, A, B). We define **class label -1** as the class label corresponding to the case where $R_{\text{true}}(Q, A) > R_{\text{true}}(Q, B)$, meaning that, for query Q, item B is more highly ranked than item A.
– If $\tilde{S}(Q, A, B) = 0$, then $\tilde{S}$ "predicts" that $R_{\text{true}}(Q, A) = R_{\text{true}}(Q, B)$. This output can also be interpreted as a "cannot decide" output, where the classifier reports that it cannot decide whether A is ranked higher or B.

We note that these class label assignments that $\tilde{S}$ makes for triples (Q, A, B) are not necessarily correct. As any kind of classifier in any type of pattern recognition problem, a classifier of type $\tilde{S}$ is correct sometimes and incorrect other times.

We consider $\tilde{S}$ to be a binary classifier, because, as mentioned in Section 3, we ignore triples (Q, A, B) such that $R_{\text{true}}(Q, A) = R_{\text{true}}(Q, B)$. Intuitively, we can consider that for such triples we do not care whether our system ranks A higher than B or not. Mathematically, classifier $\tilde{S}$ has the form of a "confidence-rated prediction" (Schapire and Singer, 1999), where the output class label is determined by the sign of the classifier output, and the absolute value of the output is an estimate of the confidence about the predicted class label. A higher absolute value corresponds to higher confidence that the predicted class label is correct.

As long as scoring model S contains some useful information about the scoring patterns in our domain, we expect classifier $\tilde{S}$ to behave as a *weak classifier*, meaning that even if the accuracy of $\tilde{S}$ is not very high, we still expect $\tilde{S}$ to be more accurate than a random, entirely non-informative classifier. For example, a simple scoring model $S_{\text{terminator}}$ for the Netflix domain can produce estimates $S_{\text{terminator}}(Q, U)$ of how a user Q would rate a movie U, as follows:

1. Let C be the rating $S_{\text{true}}(Q, \text{Terminator})$ that user Q has given to the movie "The Terminator". Note that we can allow C to also take the value "undefined", if the user Q has not rated that particular movie.
2. Let D be the average rating that movie U received by all users who also rated "The Terminator" with a score of C.
3. Define $S_{\text{terminator}}(Q, U)$ to be D.

We use the term "weak scoring model" for a model such as $S_{\text{terminator}}$, based on our intuition that such a scoring model would not be highly accurate. At the same time, such a weak scoring model still captures some useful information about the domain, and thus, if $\tilde{S}$ is the binary classifier corresponding to a weak scoring model S, we do expect the accuracy of $\tilde{S}$ to be better than the expected 0.5 accuracy of a random classifier.

As described in the above paragraphs, we use weak scoring models S to define weak binary classifiers $\tilde{S}$. Each such classifier $\tilde{S}$ predicts, for any triple (Q, A, B), if $R_{\text{true}}(Q, A) < R_{\text{true}}(Q, B)$ or not. The next step in our method involves combining multiple such classifiers $\tilde{S}_i$ into a single, optimized classifier $\tilde{S}_{\text{opt}}$ that is expected to be more accurate than the individual classifiers. Since

Algorithm 1: The AdaBoost algorithm. This description is largely copied from (Schapire and Singer, 1999).

```
**input** : $(o_1, y_1), \ldots, (o_\beta, y_\beta)$; $o_i \in \mathcal{G}, y_i \in \{-1, 1\}$.
**output**: Strong classifier $H : \mathcal{G} \to \mathbb{R}$.
Initialize $w_{i,1} = \frac{1}{\beta}$, for $i = 1, \ldots, \beta$.
**for** training round $j = 1, \ldots, J$: **do**
    1. Train weak learner using training weights $w_{i,j}$, and obtain weak
    classifier $h_j : \mathcal{G} \to \mathbb{R}$, and a corresponding weight $\alpha_j \in \mathbb{R}$.

    2. Set training weights $w_{i,j+1}$ for the next round as follows:

        $w_{i,j+1} = \frac{w_{i,j} \exp(-\alpha_j y_i h_j(o_i))}{z_j}$ .    (4)

    where $z_j$ is a normalization factor (chosen so that $\sum_{i=1}^{\beta} w_{i,j+1} = 1$).
**end**
Output the final classifier:

    $H(x) = \sum_{j=1}^{J} \alpha_j h_j(x)$.    (5)
```

each individual $\tilde{S}_i$ is a weak binary classifier, our problem is an instance of the boosting problem, and we can apply any boosting method to obtain $\tilde{S}_{\text{opt}}$. In our implementation we use the AdaBoost algorithm, as described next.

4.2. Applying AdaBoost

The AdaBoost algorithm is shown in Algorithm 1. The inputs to AdaBoost are a set of training objects o_i, together with their corresponding class labels y_i, which are equal either to -1 or to 1. The training algorithm that we use in our method is an instance of the AdaBoost algorithm. In order to fully specify how we apply AdaBoost in our method, we need to specify how the symbols of Algorithm 1 translate to our setting.

In our problem, each training object o_i corresponds to a triple (Q_i, A_i, B_i) such that Q_i is a sample query and A_i, B_i are database items. Each training label y_i is equal to 1 or -1, depending on whether $R_{\text{true}}(Q_i, A_i) < R_{\text{true}}(Q_i, B_i)$ or not. Consequently, the inputs to the training algorithm are the following:

- A set $\mathbb{T}_{\text{train}}$ of triples (Q_i, A_i, B_i), where Q_i is a query and $A_i, B_i \in \mathbb{U}$.
- A set of labels y_i that specify, for each triple (Q_i, A_i, B_i), whether $R(Q_i, A_i) < R(Q_i, B_i)$ (in which case $y_i = 1$) or $R(Q_i, A_i) > R(Q_i, B_i)$ (in which case $y_i = -1$).
- A set $\mathbb{S}$ of scoring models. By applying Equation 3 to every scoring model in $\mathbb{S}$, we obtain a corresponding set $\tilde{\mathbb{S}}$ of weak classifiers.

An important step in any AdaBoost implementation, that is left unspecified in the AdaBoost algorithm, is how to choose weak classifier h_j and weight α_j at training round j. In our method, at training round j, the simplest approach is to evaluate all the classifiers in the family $\tilde{\mathbb{S}}$ of weak classifiers given as input, and to choose the best classifier and best weight for that classifier. If the family $\tilde{\mathbb{S}}$ is too large, to reduce training time, an alternative is to evaluate only a subset of the classifiers in $\tilde{\mathbb{S}}$ at each training round. In such cases, the subset of classifiers to be evaluated at a certain training round is chosen randomly for that training round, and a new subset is chosen at each training round.

As described in (Schapire and Singer, 1999), the function $Z_j(h, \alpha)$ gives a measure of how useful it would be to choose $h_j = h$ and $\alpha_j = \alpha$ at training round j:

$$Z_j(h, \alpha) = \sum_{i=1}^{\beta} (w_{i,j} \exp(-\alpha y_i h(Q_i, A_i, B_i))) \ . \tag{6}$$

The full details of the significance of Z_j can be found in (Schapire and Singer, 1999). Here it suffices to say that if $Z_j(\tilde{F}, \alpha) < 1$ then choosing $h_j = h$ and $\alpha_j = \alpha$ is overall beneficial, and is expected to reduce the classification error. Overall, lower values of $Z_j(\tilde{F}, \alpha)$ are preferable to higher values.

Finding the optimal α for a given classifier h (following the algorithm specified in (Schapire and Singer, 1999)) and computing the Z_j value attained using that optimal α are very common operations in our algorithm, so we define specific notation:

$$\alpha_{\min}(h, j) = \text{argmin}_{\alpha \in \mathbb{R}} Z_j(h, \alpha) \ . \tag{7}$$

$$Z_{\min}(h, j) = \min_{\alpha \in \mathbb{R}} Z_j(h, \alpha) \ . \tag{8}$$

In the above equations j specifies the training round. Function $\alpha_{\min}(h, j)$ returns the weight α that minimizes $Z_j(h, \alpha)$

The number J of training rounds is a user-specified parameter. We typically run AdaBoost multiple times, with increasing values of J, until the accuracy of the resulting strong classifier stops improving. Note that if we have already run AdaBoost with $J = 50$, and we want to run AdaBoost with $J = 100$, the first 50 training rounds do not have to be performed again. If we have saved the weak classifiers $h_1, \ldots, h_{50}$, weights $\alpha_1, \ldots, \alpha_{50}$, and training weights $w_{i,51}$ calculated in the invocation of AdaBoost with $J = 50$, then we can resume training whenever we want, so as to choose additional weak classifiers.

It is important to note that AdaBoost, and boosting methods in general, are greedy optimization methods that produce only locally optimal solutions. There is no guarantee (nor empirical evidence) that the obtained strong classifiers are globally optimal. At the same time, there is ample empirical evidence that the locally optimal classifiers constructed by AdaBoost offer competitive performance in a variety of applications (Schapire and Singer, 1999; Viola and Jones, 2001; Viola, Jones and Snow, 2003)

4.3. From Strong Classifiers to Ranking Models

The output of AdaBoost is a strong classifier $H = \sum_{j=1}^{J} \alpha_j h_j$. Classifier H has been trained to estimate, for triples of type (Q, A, B), if $R_{\text{true}}(Q, A) <$

Algorithm 2: The steps of the BRM training algorithm.

```
1  Initialize training weights $w_{i,1} \leftarrow \frac{1}{\beta}$, for $i = 1, \ldots, \beta$.
2  Define classifier $H_0 = 0$ (i.e., $H_0(Q, A, B) = 0$ for all $(Q, A, B)$).
3  for training round $j = 1, \ldots, J$: do
4      $h_j \leftarrow \text{argmin}_{h \in \tilde{\mathbb{S}}} Z_{\min}(h, j)$.
5      $\alpha_j \leftarrow \alpha_{\min}(h_j, j)$.
6      if $Z_j(h_j, \alpha_j) \geq 1$ then
7          return $H_{j-1}$
8      end
9      $z_j \leftarrow Z_j(h_j, \alpha_j)$.
10     Define classifier $H_j = \sum_{i=1}^{j} \alpha_i h_i$ (in other words, $H_j = H_{j-1} + \alpha_j h_j$.)
11     Set weights $w_{i,j+1}$ for training round $j + 1$ using Eq. 4.
12 end
```

$R_{\text{true}}(Q, B)$ or not. However, our final goal is to construct not such a binary classifier, but a ranking model R_{opt}. We construct this ranking model by defining, using H, a scoring model S_{opt}, such that R_{opt} is the unique ranking model corresponding to S_{opt}.

As discussed in Section 4.2, each weak classifier h_j chosen during training is one of the classifiers $\tilde{S}$ in $\tilde{\mathbb{S}}$. In other words, each h_j is obtained by applying Equation 3 to some scoring model S. Let's use notation S_j for the scoring model that was used in defining h_j. In other words, S_j is the scoring model whose associated binary classifier $\tilde{S_j}$ is equal to h_j. Then, we can define our **optimized scoring model** $S_{\text{opt}} : \mathbb{Q} \times \mathbb{U} \rightarrow \mathbb{R}$ as follows:

$$S_{\text{opt}}(Q, U) = \sum_{j=1}^{J} (\alpha_j S_j(Q, U)) \ . \tag{9}$$

In other words, the optimized scoring model is simply the weighted sum of the individual scoring model corresponding to weak classifiers h_j, and the weights are the α_j's that were chosen by the AdaBoost algorithm.

The final product of our method, i.e., the **optimized ranking model** R_{opt}, is simply the ranking model that is defined based on S_{opt}. In other words, given a query Q, R_{opt} ranks items $U \in \mathbb{U}$ in descending order of their scores $S_{\text{opt}}(Q, U)$.

As stated in our problem formulation in Section 3, our goal is to design a method that constructs R_{opt} in a way that minimizes the error rate $E(R_{\text{opt}})$. We need to establish that the training algorithm we have described in this section indeed minimizes $E(R_{\text{opt}})$. We observe that our training algorithm is an application of AdaBoost, and thus minimizes the error rate of the constructed strong classifier H. Consequently, it suffices to demonstrate that classifier H and R_{opt} have the same error rate, so that minimizing the error rate of H is equivalent to minimizing the error rate of R_{opt}.

Given a set of test triples $\mathbb{T}_{\text{test}}$, the error rate of binary classifier H is measured as follows:

$$E(H, Q, A, B) = \begin{cases} 1 & \text{if } H(Q, A, B) < 0 \text{ and } R_{\text{true}}(Q, A) < R_{\text{true}}(Q, B), \\ 1 & \text{if } H(Q, A, B) > 0 \text{ and } R_{\text{true}}(Q, A) > R_{\text{true}}(Q, B), \\ 0.5 & \text{if } H(Q, A, B) = 0 \text{ and } R_{\text{true}}(Q, A) < R_{\text{true}}(Q, B), \\ 0.5 & \text{if } H(Q, A, B) = 0 \text{ and } R_{\text{true}}(Q, A) > R_{\text{true}}(Q, B), \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

$$E(H) = \frac{\sum_{(Q,A,B)\in\mathbb{T}_{\text{test}}} E(H,Q,A,B)}{\|\mathbb{T}_{\text{test}}\|} \tag{11}$$

Clearly, the definition of the error rate of H is analogous to the definition of the error rate $E(R_{\text{opt}})$ in Equation 2. In order to show that H and R_{opt} have the same error rate we will actually prove a stronger statement: that H and R_{opt} make errors on the same set of triples.

Proposition 1. For any triple (Q, A, B), such that Q is a query object and A, B are database items, it holds that $E(H, Q, A, B) = E(R, Q, A, B)$. In words, it holds that H classifies correctly, misclassifies, or makes a "half error" on triple (Q, A, B) if and only if R also respectively classifies correctly, misclassifies, or makes a "half error" on that triple.

Proof: We recall that R_{opt} is defined based on scoring model S_{opt}, defined as in Equation 9. First, we show that $H(Q, A, B) = S_{\text{opt}}(Q, A) - S_{\text{opt}}(Q, B)$:

$$\begin{aligned} H(Q,A,B) &= \sum_{j=1}^{J}(\alpha_j h_j(Q,A,B)) \\ &= \sum_{j=1}^{J}(\alpha_j \tilde{S}_j(Q,A,B)) \\ &= \sum_{j=1}^{J}(\alpha_j (S_j(Q,A) - S_j(Q,B))) \\ &= \sum_{j=1}^{J}(\alpha_j S_j(Q,A)) - \sum_{j=1}^{J}(\alpha_j S_j(Q,B)) \\ &= S_{\text{opt}}(Q,A) - S_{\text{opt}}(Q,B) \end{aligned}$$

In the above derivation, we made use of the following definitions and facts:

– We defined S_j as the scoring model such that $\tilde{S}_j = h_j$.
– Following Equation 3, $\tilde{S}_j(Q, A, B) = S_j(Q, A) - S_j(Q, B)$.
– Following Equation 9, $S_{\text{opt}}(Q, U) = \sum_{j=1}^{J}(\alpha_j S_j(Q, U))$.

We now proceed by showing that $H(Q, A, B)$ always agrees with the relative rankings $R_{\text{opt}}(Q, A)$ and $R_{\text{opt}}(Q, B)$. In other words, we must show that:

– $H(Q, A, B) > 0 \Leftrightarrow R_{\text{opt}}(Q, A) < R_{\text{opt}}(Q, B)$.
– $H(Q, A, B) = 0 \Leftrightarrow R_{\text{opt}}(Q, A) = R_{\text{opt}}(Q, B)$.
– $H(Q, A, B) < 0 \Leftrightarrow R_{\text{opt}}(Q, A) > R_{\text{opt}}(Q, B)$.

We can prove each of the above three cases separately, using the already proven fact that $H(Q, A, B) = S_{\text{opt}}(Q, A) - S_{\text{opt}}(Q, B)$. As a reminder, we should bear in mind our convention that higher scores correspond to higher rankings, i.e., to lower values of R_{true}, and rank 1 is defined as the highest rank. Then, for the case where $H(Q, A, B) > 0$, we have:

$$\begin{aligned} H(Q,A,B) > 0 &\Leftrightarrow S_{\text{opt}}(Q,A) - S_{\text{opt}}(Q,B) > 0 \\ &\Leftrightarrow S_{\text{opt}}(Q,A) > S_{\text{opt}}(Q,B) \end{aligned}$$

$$\Leftrightarrow R_{\text{opt}}(Q, A) < R_{\text{opt}}(Q, B)$$

The cases where $H(Q, A, B) = 0$ and $H(Q, A, B) < 0$ can be handled exactly the same way.

Consequently, we have shown that the predictions of H and R_{opt} on the relative rankings of any two items A and B under any query Q are equivalent. It readily follows that H makes an error (or a "half error") on a triple (Q, A, B) if and only if R_{opt} also makes an error (or a "half error", respectively) on that triple.

□

We conclude that, if AdaBoost has been successful in constructing a highly accurate classifier H, then R_{opt} inherits that accuracy. It is interesting to note that, in the theoretically ideal (and not likely to be encountered in practice) case where AdaBoost constructed a perfect classifier H with 100% accuracy, it follows that $R_{\text{opt}} = R_{\text{true}}$.

4.4. Choosing a Set of Training Triples

As mentioned in Section 4.2, one of the inputs to the training algorithm is a set $\mathbb{T}_{\text{train}}$ of training triples (Q_i, A_i, B_i). In this section we discuss the topic of how to form a set of training triples. Overall, the manner of selecting training triples is an implementation choice. One obvious alternative is to choose those triples entirely randomly, as long as we have enough information to evaluate whether $R_{\text{true}}(Q_i, A_i) < R_{\text{true}}(Q_i, B_i)$ or not. At the same time, while randomly chosen triples can be a reasonable starting method, other, more selective approaches may lead to more meaningful choices, given domain-specific measures of accuracy.

As an example, consider the problem of estimating K-nearest neighbor rankings, where K is a domain-specific parameter. In that case, given a query object, our goal is to identify the K nearest neighbors of the query in the database. Typically, a database can contain a large number of objects (ranging in the thousands, or millions), whereas the number of nearest neighbors we are interested in identifying is typically orders of magnitude smaller (oftentimes $K \leq 10$). In such a case, if let's say $K = 9$, we really do not mind if the estimated ranking reverses the relative order between, for example, the 50,000-th nearest neighbor and the 60,000-th nearest neighbor. However, we would mind if the estimated ranking reverses the relative order between the 50,000-th nearest neighbor and the 7-th nearest neighbor.

A similar scenario can be encountered in recommendation systems. Users usually have a vast array of choices (e.g., movies, or books), and they tend to be interested in only a small fraction of those choices, that represents the few *best* choices. Users use recommendation systems to identify, for example, a movie or book they will really enjoy, and thus users are typically interested in the items that the scoring model rates as the best. Under that scenario, the most important rankings are rankings where either the true or the estimated ranks are very high. The kind of mistake that a user would object to the most is rating a mediocre or bad choice as a good choice (which may cause the user to choose an item that turns out to be unsatisfactory), or rating a good choice too low (which may cause the user to not choose an item that would have turned out to be satisfactory).

Under both those scenarios (nearest neighbor retrieval and movie/book/product recommendation), if we choose training triples entirely randomly, then for most triples (Q_i, A_i, B_i), neither A_i nor B_i will be among the highest rated items for Q_i. In that case, the error rate on that set of triples will not truly capture our intuitive measure of accuracy, which only depends on whether the highest-rated items have been identified successfully. If we are only interested in identifying, let's say, the K best items for each query (or K nearest neighbors for the nearest neighbor problem), then we can explicitly choose training triples (Q_i, A_i, B_i) so that one among A_i and B_i is indeed one of the K highest-rated items for Q_i, and the other one among A_i and B_i is outside those K highest-rated items for Q_i. In that case, the training algorithm will optimize an error rate that captures the fact that the highest-rated items are what we are truly interested in identifying.

We note that, in a training set of $\|U\|$ items, the total number of triples that can be defined is $O(\|U\|^3)$. The constraint that A_i or B_i should be one of the K highest-rated items for Q_i reduces the number of triples that can be defined to $O(K\|U\|^2)$, where typically $K \ll \|U\|$. Still, for training sets containing thousands of items or more, $K\|U\|^2$ can still be too large a number. This presents the implementer with a dilemma: a larger set of training triples, that may lead to better accuracy, can require too much training time.

In (Athitsos et al., 2007), where we describe a special case of the proposed BRM method, we have proposed a solution that allows using large numbers of training triples (e.g., tens of millions of training triples) while keeping training time manageable. In a straightforward implementation of the training algorithm, most of the training time is spent in computing, at each training round, for each candidate weak classifier h, quantities $\alpha_{\min}(h, j)$ and $Z_{\min}(h, j)$ as described in Equations 7 and 8. Quantities $\alpha_{\min}(h, j)$ and $Z_{\min}(h, j)$ are simply used to identify the best weak classifier (and associated weight) for that round, and computing those quantities takes time linear to the number of triples. However, quantities $\alpha_{\min}(h, j)$ and $Z_{\min}(h, j)$ can be computed *approximately*, with satisfactory precision, using only a random sample of training triples at each training round. In (Athitsos et al., 2007) we demonstrated that using a random sample of 30,000 training triples at each round, out of a total of 10 million training triples, made the training algorithm run over 300 times faster, without loss in accuracy for the resulting strong classifier, as long as a new random sample is chosen at each training round.

4.4.1. Cases of Ties in Ranking

As mentioned in Section 3, when we choose training and test triples for our algorithm, we exclude triples (Q_i, A_i, B_i) where, for query Q_i, the true ranks of A_i and B_i are equal. Our choice corresponds to an optimization criterion in which, when the *true* ranks of A_i and B_i given Q_i are equal, we do not care how the *estimated* ranks of A_i and B_i compare to each other.

An alternative, that we have not experimented with, but that may be desirable in certain cases, is to include ties in the training set, and modify the optimization criterion so that if $R_{\text{true}}(Q_i, A_i) = R_{\text{true}}(Q_i, B_i)$, we penalize for any differences in value between $S_{\text{opt}}(Q_i, A_i)$ and $S_{\text{opt}}(Q_i, B_i)$. One such possible modification is to replace Equation 6 with the following alternative:

$$Z_j(h, \alpha) = \left\{ \begin{array}{ll} \sum_{i=1}^{\beta}(w_{i,j} e^{-\alpha y_i h(Q_i, A_i, B_i)}) & \text{if } R_{\text{true}}(Q_i, A_i) \neq R_{\text{true}}(Q_i, B_i), \\ \sum_{i=1}^{\beta}(w_{i,j} e^{\alpha |h(Q_i, A_i, B_i)|}) & \text{if } R_{\text{true}}(Q_i, A_i) = R_{\text{true}}(Q_i, B_i). \end{array} \right. \quad (12)$$

In this modified definition of $Z_j(h, \alpha)$, the contribution of training triples where there is no tie is exactly the same as in Equation 6. If S is the scoring model such that $h = \tilde{S}$, then a training triple with a tie contributes to $Z_j(h, \alpha)$ a loss exponential to the absolute difference between $S(Q_i, A_i)$ and $S(Q_i, B_i)$. This way, the final scoring model S_{opt} would be rewarded for trying to make $S(Q_i, A_i)$ and $S(Q_i, B_i)$ as similar as possible.

5. An Interesting Special Case: Query-Sensitive Weak Scoring Models

In the training algorithm described in Section 4, the optimized scoring model S_{opt} is constructed as a weighted linear combination of individual scoring models S_j. Consequently, the importance of each one of those individual scoring models is fixed in a global way, and is equal for all queries. However, a richer model would allow us to assign query-specific weights to each of the individual scoring models, so as to capture the fact that some individual scoring models may be predictably more accurate for certain queries and less accurate for other queries.

In this section we describe a special case of the proposed method that allows us to construct such a richer model. Let S be any individual scoring model, mapping query-item pairs to scores. Let $\mathbb{W} \subset \mathbb{Q}$ be a subset of the set of queries. Suppose we want a scoring model $S^{\mathbb{W}}$ that imitates S on queries from $\mathbb{W}$, and gives neutral outputs for all other queries. We use the term **query-sensitive scoring model** for such a scoring model $S^{\mathbb{W}}$, and we define $S^{\mathbb{W}}$ as follows:

$$S^{\mathbb{W}}(Q, A) = \begin{cases} S(Q, A) & \text{if } Q \in \mathbb{W}, \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

We call $\mathbb{W}$ the **area of influence** for query-sensitive scoring model $S^{\mathbb{W}}$.

We note that $S^{\mathbb{W}}(Q, A)$ is a legitimate scoring model, mapping query-item pairs to scores. Consequently, we can include such query-sensitive scoring models in the family of weak scoring models that is given as input to the training algorithm. If the output scoring model S_{opt}, defined in Equation 9, includes only query-sensitive scoring models $S_i^{\mathbb{W}_i}$ in its definition, with weights α_i, then S_{opt} has the following behavior: given any query Q and item A, the score $S_{\text{opt}}(Q, A)$ is obtained by the following two steps:

1. Finding all scoring models $S_i^{\mathbb{W}_i}$'s appearing in the definition of S_{opt} such that $Q \in \mathbb{W}_i$, i.e., such that Q belongs to the area of influence for scoring model $S_i^{\mathbb{W}_i}$.
2. Computing the linear combination (with weights α_i) of the outputs $S_i(Q, A)$ of the scoring models identified in the previous step.

In other words, the output $S_{\text{opt}}(Q, A)$ can be interpreted in two mathematically equivalent ways. We can say that $S_{\text{opt}}(Q, A)$ is computed by simply taking the linear combination of the query-sensitive weak scoring models $S_i^{\mathbb{W}_i}$constituting S_{opt}. Alternatively, we can say that $S_{\text{opt}}(Q, A)$ is a linear combination of query-insensitive weak scoring models S_i, in which some of those weak models (the ones such that Q does not belong to their area of influence) are ignored. Consequently, using query-sensitive weak scoring models gives the training algorithm the ability to decide that, for some queries, some weak scoring models should be ignored.

Intuitively, an advantage of query-sensitive weak classifiers is that they provide us with a richer model, and more parameters to optimize. At the same time, a possible pitfall with using query-sensitive weak classifiers is overfitting, as we have more parameters to optimize. However, if we have access to a sufficiently large training set, the richer model that query-sensitive classifiers provide is more likely to lead to more accurate results. The experimental evaluation illustrates cases were query-sensitive weak classifiers lead to improved performance.

6. Derivation of Existing Methods as Instances of the BRM Method

In the previous sections we have described the training algorithm for constructing boosted ranking models (BRMs). Each BRM is constructed as a weighted sum of individual weak scoring models. So far, we have not specified how to obtain such individual models. Defining individual weak scoring models is an implementation choice, and the formulation of the training algorithm is oblivious as to how this choice is made. In this section, we reformulate some ranking methods we have previously published, to demonstrate that those methods can be readily obtained as instances of the BRM method. In the process, we specify the problem that each of those ranking methods addresses, and the weak scoring models that were used for each of those methods.

6.1. BoostMap

In the BoostMap method (Athitsos et al., 2008), the setup is as follows: database objects and query objects belong to the same space $\mathbb{X}$. We are given a distance measure D that can be used to evaluate the distance between any two objects in $\mathbb{X}$. Given a query object Q, our goal is to identify the nearest neighbors of Q accurately and efficiently. Furthermore, we assume that the underlying distance measure D is computationally expensive. Examples of computationally expensive distance measures include the edit distance (Levenshtein, 1966) for strings, dynamic time warping (Kruskal and Liberman, 1983) for time series, or shape context matching (Belongie, Malik and Puzicha, 2002) for edge images.

In that setting, the true rank $R_{\text{true}}(Q, U)$ of a database object U given a query object Q can be computed by measuring the exact distance D between Q and every single database object. However, the time it takes to measure all those distances can become a system bottleneck. The goal of BoostMap is to produce an indexing structure, so as to speed up the process of identifying the nearest neighbors of any query Q. Essentially, the goal is to produce an approximate ranking model R_{opt}, that is as close as possible to the true ranking model R_{true}, while at the same time the approximate rankings defined by R_{opt} should be computable significantly faster than the exact rankings defined by R_{true}.

A weak scoring model S_R for this problem can be defined by choosing any database object R, and defining the scoring model as:

$$S_R(Q, U) = -|D(Q, R) - D(U, R)| \ . \tag{14}$$

The intuition behind the definition is that any two objects Q and U that are close to each other tend to have similar distances to any third object R. Note that, in the equation, the minus sign preceding the absolute value symbol enforces our convention that higher scores correspond to higher rankings. The object R that is used to define the weak scoring model is called a **reference object**. We note that, given a query Q, producing approximate nearest neighbor rankings according to scoring model S_R requires computing the exact distance D only between Q and the reference object R, assuming that distances from R to every single database object have been precomputed off-line. Therefore, producing approximate rankings according to S_R is typically several orders of magnitude more efficient than computing the exact distance D between Q and every single database object.

By choosing thousands of different reference objects we can define thousands of different weak scoring models. Then, the BRM training algorithm can be used to build an optimized ranking model R_{opt} using those weak scoring models as building blocks. The BoostMap method (Athitsos et al., 2008) can thus be specified as an instance of the BRM method, where the goal is to estimate nearest neighbor rankings, and the weak scoring models that are used in the training algorithm are of the form S_R, as defined in Equation 14.

6.2. Query-Sensitive Embeddings

Query-sensitive embeddings (QSE) were described in (Athitsos et al., 2007). QSE can be seen as an improvement of the BoostMap method described above: in both QSE and BoostMap, the goal is to estimate nearest neighbor rankings. Using the BRM framework described in this paper, what differentiates QSE from BoostMap is that in QSE the weak scoring models that are used are query-sensitive. In particular, given a reference object R, a query-sensitive weak scoring model can be defined as follows:

$$S_R^{\mathbb{W}}(Q,U) = \begin{cases} -|D(Q,R) - D(U,R)| & \text{if } Q \in \mathbb{W}, \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

The area of influence $\mathbb{W}$ can be any interval of the positive real numbers $\mathbb{R}^+$, or any complement of such an interval. In other words, an area of influence can be of the form $[a,b]$ where a and b are real numbers, or it can be of the form $[0,a) \cup (b,\infty)$.

6.3. Boost-NN

The Boost-NN method (Athitsos and Sclaroff, 2005) aims at constructing a distance measure that is optimized for nearest neighbor classification. As in BoostMap and query-sensitive embeddings (QSE), query objects and database objects belong to the same space. However, in BoostMap and QSE we are given a distance measure and the goal is to approximate the distance measure, whereas in Boost-NN we are not given a distance measure to use. Instead, in Boost-NN the goal is to construct such a distance measure, in a way that maximizes nearest neighbor classification accuracy.

In nearest neighbor classification, a query object is classified based on the class labels of its nearest neighbors. If we use R_{true} to denote the ranking model that describes the desired nearest neighbor rankings, then we would like the following to hold: $R_{\mathrm{true}}(Q,A) < R_{\mathrm{true}}(Q,B)$ if Q and A belong to the same class and B belong to a different class. In other words, we would like the database objects of the same class as Q to be the nearest neighbors of Q, and we would like all objects belonging to a different class than Q to be ranked lower than that.

In the implementation of Boost-NN described in (Athitsos and Sclaroff, 2005), each test object and each database object is a d-dimensional vector. If we use notation $X = (X_1, X_2, \ldots, X_d)$ to denote the values of the individual dimensions of an object X, then a weak scoring model for this domain can be defined as:

$$S_i(Q, X) = -|Q_i - X_i| \ . \tag{16}$$

In other words, S_i simply considers the absolute difference between Q and X in the i-th dimension. Using this definition, Boost-NN can be seen simply as an application of the BRM method that uses the weak scoring models of the form S_i defined in the above equation.

7. Application to Movie Recommendations

In addition to reformulating our previously published ranking methods as instances of the proposed BRM method, we have also applied BRM to the problem of ranking movie preferences of individual users. Our experimental domain is the Netflix dataset (Bennett et al., 2007). In that dataset, users rate movies that they have watched with an integer between 1 and 5, where rating 1 is the worst rating and rating 5 is the best rating. We denote by $S_{\text{true}}(Q, U)$ the rating that user Q gives to movie U.

The typical user only rates a relatively small fraction of the tens of thousands of movies in the dataset. The goal in the Netflix contest is to predict ratings for movies that a user has not yet seen, so as to be able to make useful recommendations to the user. The goal of our method is related, but not identical, to the goal in the Netflix contest: in our method we are not interested in predicting ratings, but rather in predicting rankings. In other words, we do not mind if our constructed scoring model S_{opt} does not agree numerically with user ratings, but we do mind if the ranking model R_{opt} ranks, for any user Q, any pair of movies U_1, U_2 in a way that conflicts with the ratings that the user gave on those movies.

A simple scoring model that can be used in predicting movie ratings is the "average" score S_{avg}:

$$S_{\text{avg}}(Q, U) = \text{average rating of U among all users who have rated U.} \tag{17}$$

Note that the user Q is not used in defining the value of $S_{\text{avg}}(Q, U)$, which always gives the same score for U, regardless of the user. We have confirmed in our experiments that S_{avg} is a fairly informative scoring model, that can be a useful building block for constructing BRMs.

A family of scoring models can be defined using what we call "reference movies". Let's pick a specific movie M. We define $\mathbb{Q}(M, n)$ to be the set of users who have rated M with rating n. Then, if we use M as a reference movie, we can define a scoring model S^M as follows:

$$S^M(Q, U) = \begin{cases} 0, \text{ if the user has not rated } M, \\ \text{mean rating of U by users in } \mathbb{Q}(M, S_{\text{true}}(Q, M)), \text{ otherwise.} \end{cases} \tag{18}$$

In more detail, to compute $S^M(Q, U)$ we perform the following steps:

– If user Q has not rated the reference movie M, then $S^M(Q, U) = 0$. Note that, in this case, scoring model S^M does not give us any useful information for user Q, and assigns a score of 0 to all movies.

– If user Q has rated reference movie M with some score $n = S_{\text{true}}(Q, M)$, then we find the set $\mathbb{Q}(M, n)$ of all users who agreed with user Q on the rating of M. Then, we find all users in $\mathbb{Q}(M, n)$ who have rated movie U, and we return the average of those ratings.

Finally, another family of scoring models can be obtained by applying an SVD-like factorization process (Paterek, 2007; Zhou et al., 2008). In those methods, given a desired dimensionality, every user Q is assigned a d-dimensional vector $V(Q)$, and every movie is assigned a d-dimensional vector $V(U)$. These vector assignments are optimized so as to minimize the error of using the dot product between $V(Q)$ and $V(U)$ as an approximation for $S_{\text{true}}(Q, U)$. In our implementation we used Simon Funk's psuedo-SVD algorithm, FunkSVD, to assign a vector to each query and each user (Funk, 2006).

We can also define query-sensitive versions of the weak scoring models described above. One way of defining regions of influence for the query-sensitive weak models is to do k-means clustering on the set of queries. We should note here that k-means clustering is applicable on sets of vectors, where the "mean" operation is applicable. In the Netflix dataset, we vectorize all queries by applying the FunkSVD factorization process (Funk, 2006). Then, given a new query Q that we have not seen before, Q can be assigned to the cluster of its nearest mean, among the selected k means. This way, we define k regions of influence, such that the i-th region corresponds to the set of queries that are closer to the i-th mean than to any of the other means. If, using k-means clustering, or any other approach, we define k possible regions of influence, then every weak scoring model S can be used to define k different query-sensitive weak scoring models.

8. Experimental Evaluation

The proposed BRM method is a general framework for learning ranking models. As explained in Section 6, several methods we have previously designed for nearest neighbor retrieval and classification can be re-derived as special cases of this unifying framework. In that sense, a large body of experimental results that we have previously published for BoostMap (Athitsos et al., 2008), query-sensitive embeddings (Athitsos et al., 2007), and Boost-NN (Athitsos and Sclaroff, 2005) can serve as experimental validation for the BRM method, and highlight the competitive results that can be obtained using BRM. In this paper we provide some additional experiments, in which the BRM method is applied on the problem of making movie recommendations. The experimental evaluation presented here serves two goals:

– To present a new application of the BRM method, on the problem of making movie recommendations, so as to illustrate the generality of the BRM method, and its ability to treat in a domain-independent manner rather disparate ranking problems such as nearest neighbor retrieval and movie recommendations.

– To illustrate some of the common behaviors, and also some of the possible pitfalls of the BRM method.

Diff Setting	**Best Individual Accuracy**	**Accuracy of Average Score**	**BRM Accuracy**
Diff1	62.91%	68.11%	70.40%
Diff2	66.70%	73.93%	77.28%
Diff3	67.95%	76.39%	80.65%
Diff4	68.27%	76.52%	81.45%

Table 1. For each Diff setting, this table shows: the accuracy of the best individual weak scoring model, the accuracy of the average of all weak scoring models, and the accuracy of the BRM result. The results shown for BRM are obtained using the query-sensitive version of BRM.

8.1. Data, Parameters, and Implementation Choices

Our experimental domain in this evaluation is the Netflix dataset (Bennett et al., 2007). That dataset contains a total of 100,480,507 ratings (each rating corresponding to a user/movie pair), from a total of 480,189 users and for a total of 17,770 movies. From this dataset, we chose a training set of 80,425 users. For all training triples (Q, A, B) used in our training algorithm, Q was constrained to belong to this training set of users, whereas A and B could be chosen among the entire set of movies. To measure the error rate of the constructed ranking models, we used a test set of 80,425 users, meaning that for every triple (Q, A, B) used in evaluating the error rate of ranking model, Q was constrained to belong to the set of test users. We explicitly enforced that the set of training users and the set of test users be disjoint from each other. In each invocation of our training algorithm, we used 200,000 training triples. To evaluate the accuracy of each constructed ranking model, we used 200,000 test triples.

In order to obtain a more detailed quantitative evaluation, every method and variation that we have tested has been evaluated in four different settings, each of which used a different set of training and test triples. In particular, in choosing training and test triples (Q, A, B), we enforced the constraint that $|S_{\text{true}}(Q, A) - S_{\text{true}}(Q, B)| \geq n$. We used four different values for n: $n = 1, 2, 3$, or 4. Each of those four cases is denoted respectively as **Diff1**, **Diff2**, **Diff3**, and **Diff4**. The Diff1 setting was the least constrained, as it could include any triple (Q, A, B) as long as user Q did not give the same rating to both A and B. The Diff1 setting was also the hardest to classify correctly. The Diff4 setting was the most constrained, as it only included triples (Q, A, B) where user Q rated one of the two movies with a 1 and the other one with a 5. The Diff4 setting was the easiest to classify correctly.

For every experiment conducted with a particular family of weak scoring models, we also conducted a parallel experiment using query-sensitive versions of the same scoring models. In all such cases, the regions of influence $\mathbb{W}_j$ were chosen by applying k-means clustering, with $k = 10$, on the set of training users. To obtain a vector representation of each user, each user was mapped to a 128-dimensional vector, comprised of a user's weights in the decomposed user feature matrix produced by FunkSVD (Funk, 2006).

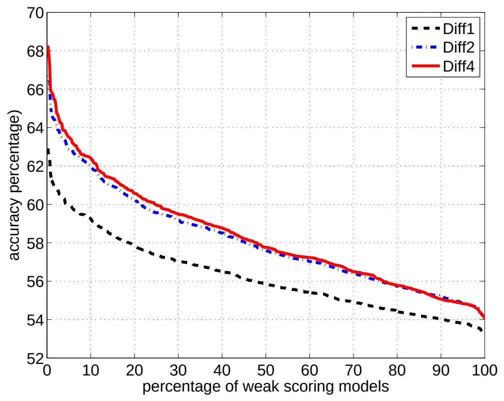


Fig. 1. Accuracy of individual weak scoring models for the Diff1, Diff2, and Diff4 settings. A point (x, y) on the plot means that x% of the weak scoring models attained an accuracy of at least y%. The plot for the Diff3 setting was very similar to that of the Diff2 and Diff4 settings, and was omitted to avoid cluttering the figure.

8.2. Experimental Results

In our first set of experiments, we applied the proposed training algorithm to construct a ranking model using individual weak scoring models defined based on reference movies, as specified in Equation 18 . We used a total of 500 candidate reference movies, which were selected as the 500 movies in the Netflix dataset that were rated by the most users. This means that the set $\mathbb{S}$ of weak scoring models, which was passed as input to the training algorithm, contained 500 scoring models.

Figure 1 shows the accuracy of the individual weak scoring models, as measured on test triples of type Diff1, Diff2, and Diff4 (the Diff3 plot was very similar to the plots for Diff2 and Diff4, and was omitted to improve the readability of the figure). We note that the accuracies of the individual weak scoring models range between 53.22% and 62.91% for Diff1, between 54.12% and 66.70% for Diff2, between 54.36% and 67.95% for Diff3, and between 54.04% and 68.27% for Diff4.

Table 1 displays the results of this set of experiments. In this table, we compare the results obtained using BRM with the results obtained using two alternatives: the first alternative is to simply use the best individual weak scoring model. The second alternative is to combine all scoring models by simply averaging them. We note that both the average of all scoring models and the boosted ranking model are linear combinations of individual weak scoring models. The key difference is that the average is an unweighted linear combination, whereas the boosted ranking model is a weighted linear combination where the weights have been explicitly chosen by the training algorithm so as to maximize accuracy.

The results in Table 1 are an example of a case where we consider our method to be successful. First of all, in all settings, the resulting boosted ranking model obtains clearly better accuracy compared to any of its individual components, and also compared to the average of all scoring models. We should note that the experimental results for BoostMap (Athitsos et al., 2008) demonstrated similar behavior for boosted ranking models used for efficient nearest neighbor retrieval:

	5-dim	**32-dim**	**64-dim**	**96-dim**	**128-dim**
Diff1	73.68%	80.60%	83.42%	85.25%	86.37%
Diff2	81.46%	89.57%	92.35%	93.98%	94.88%
Diff3	85.74%	94.05%	96.32%	97.40%	97.91%
Diff4	87.42%	95.90%	97.85%	98.68%	98.94%

Table 2. Accuracy of each of the individual SVD-based scoring models for each of the four Diff settings.

Diff Setting	**Best Individual Accuracy (128-dim SVD)**	**BRM Accuracy**
Diff1	86.37%	86.90%
Diff2	94.88%	95.01%
Diff3	97.91%	98.19%
Diff4	98.94%	99.06%

Table 3. For each Diff setting, this table shows results obtained using, as a family of weak scoring models, a set comprising all the various SVD-based scoring models, the average of the reference movie-based scoring models, and individual reference movie-based scoring models selected from the top 500 popular movies. The middle column shows the accuracy obtained using the best among all individual weak scoring models (which, in all settings, was the scoring model based on the 128-dimensional SVD representation). The right column shows the accuracy obtained using the BRM method. The results shown for BRM are obtained using the query-sensitive version of BRM.

better accuracy both compared to each individual weak scoring model, and compared to the average of all scoring models.

In a second set of experiments, we included in our pool of weak scoring models, in addition to the 500 reference movie-based scoring models, the average S_{avg} of all reference movie-based scoring models, and five scoring models obtained by using FunkSVD (Funk, 2006), with an individual scoring model corresponding to each of the following dimensionalities of the SVD representation: 5, 32, 64, 96 and 128. Table 2 shows the accuracy achieved by each of the individual SVD-based scoring models. Table 3 shows the results we obtained with the boosted ranking model.

As Table 3 indicates, in all Diff settings the BRM method obtained accuracies that were slightly better than those of the 128-dimension SVD-based scoring model, which was in all cases the most accurate weak scoring model. At the same time, the results for the Diff1 and Diff2 settings represent cases where the BRM method does not work as well, in the sense that the performance gains are not significant, compared to not using BRM and just using the best weak scoring model.

Comparing the Diff1 and Diff2 results shown in Table 3 with the Diff1 and Diff2 results shown in Table 1, we notice some differences that may help explain the difference in accuracy gains obtained by BRM. In the first set of experiments (Table 1), the family of weak scoring models included multiple complementary models, and no individual model was significantly better than every single other model. On the other hand, in the second set of experiments (Table 3), for the Diff1

	Reference Movie-Based Weak Models		**Reference Movie-Based + Avg. + SVD Weak Models**	
Diff Setting	**Non-Query-Sensitive**	**Query-Sensitive**	**Non-Query-Sensitive**	**Query-Sensitive**
Diff1	69.98%	70.40%	85.86%	86.90%
Diff2	76.72%	77.28%	94.85%	95.01%
Diff3	80.03%	80.65%	98.09%	98.20%
Diff4	80.80%	81.45%	98.79%	99.06%

Table 4. Comparing the non-query-sensitive and the query-sensitive versions of the BRM method, as applied to two families of weak scoring models. The first family contained 500 reference movie-based scoring models. The second family was a superset of the first family, that also included the average of all movie-based scoring models, and FunkSVD-based scoring models.

and Diff2 settings, the family of weak scoring models contained a few individual models (the ones based on 64, 96, and 128-dimensional SVD representations) that were largely similar to each other as well as significantly more accurate than any other model. In that case, the training algorithm did not manage to identify a way to use the remaining 503 scoring models in a way that significantly complements the best individual model.

On a more positive note, for the Diff3, and Diff4 settings, the accuracy gains may seem small in absolute terms, but they represent significant reductions in the frequency of misclassified triples. In general, it is well known that accuracy gains are much harder to attain when the baseline accuracy is already high. For example, for the Diff3 setting, the accuracy gain from the 97.91% accuracy of the 128-dimensional SVD model to the 98.19% accuracy for the BRM model, although numerically small, can be considered significant: out of 200,000 test triples, 3,620 test triples were misclassified by the BRM model, which was 13.4% fewer than the 4,180 test triples misclassified by the 128-dimensional SVD model. Similarly, for the Diff4 setting, the number of test triples misclassified by the BRM model was 11.3% smaller than the number test triples misclassified by the 128-dimensional SVD model.

Furthermore, these improvements attained by BRM for the Diff3 and Diff4 settings were attained despite the fact that these learning problems presented the same challenge that was also present for the Diff1 and Diff2 settings: i.e., three weak scoring models that are very similar to each other and significantly better than the vast majority of the remaining weak scoring models. The learning algorithm still managed to combine these very accurate models with other, significantly less accurate models, so as to achieve a measurable decrease in the frequency of misclassified triples.

In a third set of experiments, we have evaluated the difference in accuracy between query-sensitive BRM models and non-query-sensitive BRM models. Table 4 shows the results obtained using each of the two weak scoring model families that we used in the previous experiments: the family of 500 reference movie-based models, and the extended family where, in addition to those 500 models, we include the unweighted average of all reference movie-based models, and the five scoring models obtained using Funk-SVD with respective dimensionalities of

5, 32, 64, 96, and 128. We note that, for both families of weak scoring models, and in all Diff settings, the query-sensitive models are somewhat more accurate than the non-query-sensitive models. The accuracy gains were not dramatic in any case, the gain being always less than 1%. At the same time, the accuracy increase from 98.79% to 99.06% for the extended weak scoring model family and the Diff4 setting can be seen as significant, as it was measured on $200,000$ test triples, and it represents a 22% decrease in the number of test triples that were misclassified. With respect to previously published results on instances of the BRM method, we should note that query-sensitive BRM models were shown to lead to significant accuracy gains on the problem of nearest neighbor indexing (Athitsos et al., 2007).

In summary, our experimental results illustrate the ability of our method to improve ranking accuracy, sometimes significantly, compared to the individual weak scoring models given as input. The results also illustrate the potential pitfall whereby some individual scoring models are both similar to each other and at the same time significantly more accurate than all other models. In that case, our method may not offer significant accuracy gains compared to the alternative of simply using the best weak model. Finally, the results demonstrate that using the query-sensitive variation of the BRM method led in all cases to gains, albeit modest, in accuracy.

9. Discussion and Conclusions

In this paper we have addressed the problem of learning a ranking model from training data. We have assumed that we are given as input a large family of weak scoring models, which may or may not be very accurate by themselves. The main contribution of the proposed BRM method is that it reduces the problem of learning a ranking model (i.e., learning a function mapping query/item pairs to an integer rank) to the significantly more simple problem of learning a binary classifier (i.e., mapping a pattern to a 0/1 label). More specifically, we have shown how to convert the problem of learning a ranking model to the well-studied boosting problem of combining multiple weak binary classifiers into a single, optimized classifier.

The reduction of the ranking problem into the binary-classifier boosting problem allows us to formulate a training algorithm that is simply an instance of the AdaBoost training algorithm. An important feature of this reduction, and the resulting training algorithm, is that they are formulated in domain-independent terms, and can readily be applied to a variety of ranking problems, such as nearest neighbor retrieval and classification, or recommendation systems. We have shown how to derive some already existing methods addressing specific ranking problems, such as BoostMap (Athitsos et al., 2008), query-sensitive embeddings (Athitsos et al., 2007), and Boost-NN (Athitsos and Sclaroff, 2005), as instances of the general framework proposed in this paper. We have also applied this new framework to the problem of movie recommendations.

Like any boosting method, our method does not guarantee that it will actually indeed improve performance in any particular domain. At the same time, we have seen in our experiments cases where our method led to significant improvements in classification accuracy compared to the individual scoring models that it used as building blocks. Additional evidence for the usefulness of the proposed method can be obtained from prior publications, where individual instances of the BRM method were evaluated (Athitsos and Sclaroff, 2005; Athitsos et al., 2008; Athitsos et al., 2007). These results, and the domain-independent nature of our formulation, underscore the potential of the BRM method to improve performance in a variety of ranking applications.

Acknlowledgements

The research reported here has been partially funded by grants from the National Science Foundation: IIS-0705749, IIS-0812601, CNS-0923494. This research has also been supported by a UTA startup grant to Professor Athitsos, and UTA STARS awards to Professors Chris Ding and Fillia Makedon.

References

Adomavicius, G. and Tuzhilin, A. (2005), 'Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions', *IEEE Transactions on Knowledge and Data Engineering (TKDE)* **17**(6), 734–749.

Andoni, A. and Indyk, P. (2006), Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions, *in* 'IEEE Symposium on Foundations of Computer Science (FOCS)', pp. 459–468.

Aronovich, L. and Spiegler, I. (2010), 'Bulk construction of dynamic clustered metric trees', *Knowledge and Information Systems (KAIS)* **22**(2), 211–244.

Athitsos, V., Alon, J., Sclaroff, S. and Kollios, G. (2008), 'Boostmap: An embedding method for efficient nearest neighbor retrieval', *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* **30**(1), 89–104.

Athitsos, V., Hadjieleftheriou, M., Kollios, G. and Sclaroff, S. (2007), 'Query-sensitive embeddings', *ACM Transactions on Database Systems (TODS)* **32**(2).

Athitsos, V. and Sclaroff, S. (2005), Boosting nearest neighbor classifiers for multiclass recognition, *in* 'IEEE Workshop on Learning in Computer Vision and Pattern Recognition'.

Becchetti, L., Colesanti, U., Marchetti-Spaccamela, A. and Vitaletti, A. (2010), 'Recommending items in pervasive scenarios: models and experimental analysis', *Knowledge and Information Systems (KAIS)* **(published online)**.
URL: *http://dx.doi.org/10.1007/s10115-010-0338-4*

Bell, R. and Koren, Y. (2007), 'Lessons from the Netflix prize challenge', *SIGKDD Explorations* **9**(2), 75–79.

Bell, R., Koren, Y. and Volinsky, C. (2007), Modeling relationships at multiple scales to improve accuracy of large recommender systems, *in* 'ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', pp. 95–104.

Belongie, S., Malik, J. and Puzicha, J. (2002), 'Shape matching and object recognition using shape contexts', *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* **24**(4), 509–522.

Bennett, J., Elkan, C., Liu, B., Smyth, P. and Tikk, D. (2007), 'Kdd cup and workshop 2007', *SIGKDD Explorations* **9**(2), 51–52.

Bezerra, B. L. D. and Carvalho, F. A. T. (2010), 'Symbolic data analysis tools for recommendation systems', *Knowledge and Information Systems (KAIS)* **(published online)**.
URL: *http://dx.doi.org/10.1007/s10115-009-0282-3*

Böhm, C., Berchtold, S. and Keim, D. A. (2001), 'Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases', *ACM Computing Surveys* **33**(3), 322–373.

Bozkaya, T. and Özsoyoglu, Z. (1999), 'Indexing large metric spaces for similarity search queries', *ACM Transactions on Database Systems (TODS)* **24**(3), 361–404.

Bridge, D. (2001), Product recommendation systems: A new direction, *in* 'Proceedings of the Workshop Programme at the Fourth International Conference on Case-Based Reasoning', pp. 79–86.

Chen, H.-C. and Chen, A. L. P. (2005), 'A music recommendation system based on music and user grouping', *Journal of Intelligent Information Systems* **24**(2), 113–132.

Chen, H., Liu, J., Furuse, K., Yu, J. and Ohbo, N. (2010), 'Indexing expensive functions for efficient multi-dimensional similarity search', *Knowledge and Information Systems (KAIS)* **(published online)**.
URL: *http://dx.doi.org/10.1007/s10115-010-0303-2*

Ciaccia, P., Patella, M. and Zezula, P. (1997), M-tree: An efficient access method for similarity search in metric spaces, *in* 'International Conference on Very Large Databases (VLDB)', pp. 426–435.

Flickner, M., Sawhney, H., Niblack, W., Ashley, J., Huang, Q., Dom, B., Gorkani, M., Hafner, J., Lee, D., Petkovic, D., Steele, D. and Yanker, P. (1995), 'Query by image and video content: The QBIC system', *IEEE Computer* **28**(9).

Freund, Y., Iyer, R., Schapire, R. E. and Singer, Y. (2003), 'An efficient boosting algorithm for combining preferences', *Journal of Machine Learning Research (JMLR)* **4**, 933–969.

Friedman, J., Hastie, T. and Tibshirani, R. (2000), 'Additive logistic regression: a statistical view of boosting', *Annals of Statistics* **28**(2), 337–374.

Funk, S. (2006), 'Netflix update: Try this at home', http://sifter.org/ simon/journal/20061211.html.

Gantner, Z., Drumond, L., Freudenthaler, C., Rendle, S. and Schmidt-Thieme, L. (2010), Learning attribute-to-feature mappings for cold-start recommendations, *in* 'IEEE International Conference on Data Mining (ICDM)'.

Gionis, A., Indyk, P. and Motwani, R. (1999), Similarity search in high dimensions via hashing, *in* 'International Conference on Very Large Databases (VLDB)', pp. 518–529.

Goldberg, D., Nichols, D., Oki, B. and Terry, D. (1992), 'Using collaborative filtering to weave an information tapestry', *Communications of the ACM* **35**(12), 61–70.

Hjaltason, G. R. and Samet, H. (2003*a*), 'Index-driven similarity search in metric spaces', *ACM Transactions on Database Systems (TODS)* **28**(4), 517–580.

Hjaltason, G. and Samet, H. (2003*b*), 'Properties of embedding methods for similarity searching in metric spaces', *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* **25**(5), 530–549.

Kanth, K. V. R., Agrawal, D. and Singh, A. (1998), Dimensionality reduction for similarity searching in dynamic databases, *in* 'ACM International Conference on Management of Data (SIGMOD)', pp. 166–176.

Kim, C. Y., Lee, J. K., Cho, Y. H. and Kim, D. H. (2004), 'VISCORS: A visual-content recommender for the mobile web', *IEEE Intelligent Systems* **19**, 32–39.

Kim, Y., Chung, C.-W., Lee, S.-L. and Kim, D.-H. (2010), 'Distance approximation techniques to reduce the dimensionality for multimedia databases', *Knowledge and Information Systems (KAIS)* **(published online)**.
URL: *http://dx.doi.org/10.1007/s10115-010-0322-z*

Koren, Y. (2008), Factorization meets the neighborhood: a multifaceted collaborative filtering model, *in* 'ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', pp. 426–434.

Koren, Y. (2009), Collaborative filtering with temporal dynamics, *in* 'ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', pp. 447–456.

Kruskal, J. B. and Liberman, M. (1983), The symmetric time warping algorithm: From continuous to discrete, *in* 'Time Warps', Addison-Wesley.

Levenshtein, V. I. (1966), 'Binary codes capable of correcting deletions, insertions, and reversals', *Soviet Physics* **10**(8), 707–710.

Li, C., Chang, E., Garcia-Molina, H. and Wiederhold, G. (2002), 'Clustering for approximate similarity search in high-dimensional spaces', *IEEE Transactions on Knowledge and Data Engineering (TKDE)* **14**(4), 792–808.

Li, S. Z. and Zhang, Z. Q. (2004), 'FloatBoost learning and statistical face detection', *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* **26**(9), 1112–1123.

Linden, G., Smith, B. and York, J. (2003), 'Amazon.com recommendations: Item-to-item collaborative filtering', *IEEE Internet Computing* **7**(1), 76–80.

Paterek, A. (2007), Improving regularized singular value decomposition for collaborative filtering, *in* 'Proceedings of KDD Cup and Workshop'.

Sakurai, Y., Yoshikawa, M., Uemura, S. and Kojima, H. (2000), The A-tree: An index structure for high-dimensional spaces using relative approximation, *in* 'International Conference on Very Large Databases (VLDB)', pp. 516–526.

Salakhutdinov, R., Mnih, A. and Hinton, G. E. (2007), Restricted Boltzmann machines for collaborative filtering, *in* 'International Conference on Machine Learning (ICML)', pp. 791–798.

Sarwar, B., Karypis, G., Konstan, J. and Riedl, J. (2001), Item-based collaborative filtering recommendation algorithms, *in* 'International World Wide Web Conference (WWW)', pp. 285–295.

Schafer, J. B., Frankowski, D., Herlocker, J. L. and Sen, S. (2007), Collaborative filtering recommender systems, *in* P. Brusilovsky, A. Kobsa and W. Nejdl, eds, 'The Adaptive Web: Methods and Strategies of Web Personalization', Springer, pp. 291–324.

Schapire, R. and Singer, Y. (1999), 'Improved boosting algorithms using confidence-rated predictions', *Machine Learning* **37**(3), 297–336.

Takács, G., Pilászy, I., Németh, B. and Tikk, D. (2009), 'Scalable collaborative filtering approaches for large recommender systems', *Journal of Machine Learning Research (JMLR)* **10**, 623–656.

Toescher, A., Jahrer, M. and Legenstein, R. (2008), Improved neighborhood-based algorithms for large-scale recommender systems, *in* 'Proceedings of KDD Cup and Workshop'.

Traina, Jr., C., Traina, A., Seeger, B. and Faloutsos, C. (2000), Slim-trees: High performance metric trees minimizing overlap between nodes, *in* '7th International Conference on Extending Database Technology (EDBT)', pp. 51–65.

Tuncel, E., Ferhatosmanoglu, H. and Rose, K. (2002), VQ-index: An index structure for similarity searching in multimedia databases, *in* 'Proc. of ACM Multimedia', pp. 543–552.

Uhlman, J. (1991), 'Satisfying general proximity/similarity queries with metric trees', *Information Processing Letters* **40**(4), 175–179.

Viola, P. A., Jones, M. J. and Snow, D. (2003), Detecting pedestrians using patterns of motion and appearance., *in* 'IEEE International Conference on Computer Vision (ICCV)', pp. 734–741.

Viola, P. and Jones, M. (2001), Rapid object detection using a boosted cascade of simple features, *in* 'IEEE Conference on Computer Vision and Pattern Recognition (CVPR)', Vol. 1, pp. 511–518.

Vucetic, S. and Obradovic, Z. (2005), 'Collaborative filtering using a regression-based approach', *Knowledge and Information Systems (KAIS)* **7**(1), 1–22.

Weber, R. and Böhm, K. (2000), Trading quality for time with nearest-neighbor search, *in* 'International Conference on Extending Database Technology: Advances in Database Technology', pp. 21–35.

Weber, R., Schek, H.-J. and Blott, S. (1998), A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces, *in* 'International Conference on Very Large Databases (VLDB)', pp. 194–205.

White, D. A. and Jain, R. (1996), Similarity indexing: Algorithms and performance, *in* 'Storage and Retrieval for Image and Video Databases (SPIE)', pp. 62–73.

Wu, J. (2009), Binomial matrix factorization for discrete collaborative filtering, *in* 'IEEE International Conference on Data Mining (ICDM)', pp. 1046–1051.

Yianilos, P. (1993), Data structures and algorithms for nearest neighbor search in general metric spaces, *in* 'ACM-SIAM Symposium on Discrete Algorithms', pp. 311–321.

Zhang, M. and Alhajj, R. (2010), 'Effectiveness of NAQ-tree as index structure for similarity search in high-dimensional metric space', *Knowledge and Information Systems (KAIS)* **22**(1), 1–26.

Zhang, R. and Tran, T. (2010), 'An information gain-based approach for recommending useful product reviews', *Knowledge and Information Systems (KAIS)* **(published online)**.
URL: *http://dx.doi.org/10.1007/s10115-010-0287-y*

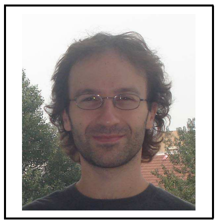
Zheng, Z., Chen, K., Sun, G. and Zha, H. (2007), A regression framework for learning ranking functions using relative relevance judgments, *in* 'ACM SIGIR Conference on Research and Development in Information (SIGIR)', pp. 287–294.

Zhou, Y., Wilkinson, D., Schreiber, R. and Pan, R. (2008), Large-scale parallel collaborative filtering for the Netflix prize, *in* 'Algorithmic Aspects in Information and Management (AAIM)', pp. 337–348.

Author Biographies

Kevin Dela Rosa is currently pursing a Computer Science Masters degree at Carnegie Mellon Universitys Language Technologies Institute. He received his Bachelors degree in Software Engineering and Physics from the University of Texas at Arlington. In the past, Mr. Dela Rosa has worked at the National Institute of Standards and Technology (NIST) and Space and Naval Warfare Systems Command (SPAWAR), and has conducted research in the domains of machine learning, natural language processing, software engineering, computer assisted language learning, and astrophysics. Mr. Dela Rosa's current research interests include applied machine learning, social media, information retrieval, and question answering.

Vangelis Metsis is currently a Ph.D. student at the Department of Computer Science and Engineering of the University of Texas at Arlington (UTA). He received his Computer Science B.Sc. degree from the Department of Informatics of Athens University of Economics and Business in 2005. During 2006- 2007 he worked as a research associate at the National Center for Scientific Research Demokritos in Athens, Greece before joining the Heracleia Human-Centered Computing Laboratory at UTA to work as a research assistant towards his Ph.D. degree. His research interests include Machine Learning, Bioinformatics and Pervasive Computing. Mr. Metsis has co-authored several peer reviewed papers published in technical conferences and journals and has served as a committee member and reviewer in many others. Mr. Metsis is currently a member of Upsilon Pi Epsilon Texas Gamma Chapter, and Golden Key honor societies.



Vassilis Athitsos received the BS degree in mathematics from the University of Chicago in 1995, the MS degree in computer science from the University of Chicago in 1997, and the PhD degree in computer science from Boston University in 2006. In 2005-2006 he worked as a researcher at Siemens Corporate Research, developing methods for database-guided medical image analysis. In 2006-2007 he was a postdoctoral research associate at the Computer Science department at Boston University. Since August 2007 he is an assistant professor at the Computer Science and Engineering department at the University of Texas at Arlington. His research interests include computer vision, machine learning, and data mining. His recent work has focused on efficient similarity-based retrieval, gesture recognition, shape modeling and detection, and medical image analysis.

Correspondence and offprint requests to: Vassilis Athitsos, Computer Science and Engineering Department, University of Texas at Arlington, Arlington, Texas 76019, USA. Email: athitsos@uta.edu