Learning User Preferences By Adaptive Pairwise Comparison[*]

Li Qian
University of Michigan, Ann Arbor
eql@umich.edu

Jinyang Gao
Nataional University of Singapore
jinyang.gao@comp.nus.edu.sg

H. V. Jagadish
University of Michigan, Ann Arbor
jag@umich.edu

ABSTRACT

Users make choices among multi-attribute objects in a data set in a variety of domains including used car purchase, job search and hotel room booking. Individual users sometimes have strong preferences between objects, but these preferences may not be universally shared by all users. If we can cast these preferences as derived from a quantitative user-specific preference function, then we can predict user preferences by learning their preference function, even though the preference function itself is not directly observable, and may be hard to express.

In this paper we study the problem of preference learning with pairwise comparisons on a set of entities with multiple attributes. We formalize the problem into two subproblems, namely *preference estimation* and *comparison selection*. We propose an innovative approach to estimate the preference, and introduce a binary search strategy to adaptively select the comparisons. We introduce the concept of an *orthogonal query* to support this adaptive selection, as well as a novel S-tree index to enable efficient evaluation of orthogonal queries.

We integrate these components into a system for inferring user preference with adaptive pairwise comparisons. Our experiments and user study demonstrate that our adaptive system significantly outperforms the naïve random selection system on both real data and synthetic data, with either simulated or real user feedback. We also show our preference learning approach is much more effective than existing approaches, and our S-tree can be constructed efficiently and perform orthogonal query at interactive speeds.

1. INTRODUCTION

Users often have to choose entities of interest from a large set of multi-attribute objects. While most systems allow users to specify selection conditions on individual attributes to reduce the size of the set, users may not always be good at tightly specifying such conditions. Furthermore, even within acceptable ranges of values for each individual attribute, there may still be too many entities for users to examine individually to choose the ones that they prefer.

Consider a hotel booking scenario. Every user may wish for an inexpensive hotel with luxury amenities located right on the beach. In the absence of such a hotel, each user has to trade off between these attributes. These tradeoffs are individualistic, and hard for most users to state explicitly as an objective formula. For example, few would say explicitly that every 100m distance to the beach is worth 10 dollars in the room rate, even if that is the function they are intuitively applying to make their choices. This paper focuses on learning such quantitative user preference [1, 19].

Skyline queries have been proposed to address such queries. However, they often return a large fraction of the data set, particularly in a multi-attribute scenario. Furthermore, users often have very different individual preferences (whereas techniques such as skyline queries focus on global preferences). This is perhaps most obvious in an online dating scenario, but is equally true of hotels, houses, jobs and so on. We will show this heterogeneity later in this paper.

In cognitive psychology, the Thurstone's Law of Comparative Judgement indicates that pairwise comparison is a more effective way to learn a preference function than directly choosing the set of preferred entities or deriving the overall ranking, In fact, there is considerable literature on learning and ranking by pairwise comparison in the machine learning community [26]. Recent works in crowdsourced ranking [10, 35] also leverage pairwise comparisons.

In this paper, we study the problem of quantitative preference learning by pairwise comparison on structured entities. Specifically, given a set of entities with associated attributes, we choose a set of entity pairs and ask the user which entity is more preferable in each pair. Based on the feedback we estimate the overall quantitative preference function. Once a system learns a user preference function, it can use it in many ways. For example, imagine a travel site that knows your preferences and automatically ranks hotels in your preferred order; or a shopping site that sends you an email when a promotion on a particular camera makes it likely to be your top choice. It is easy to come up with many such uses, and developing these is outside the scope of this paper, which is focused on a formulation of preference learning that can be used in such scenarios. We note that other notions of preference learning have been posed, in other contexts [18, 26]. We discuss them in Section 7.

For such a preference learning scheme to be practical, it must meet three conditions. First, pairwise comparisons should be easy to answer. Second, one should be able to learn at least an approximation of the preference function with only a few comparisons. And third, the system must be able to pose such comparison questions at interactive speed. The first condition is immediately satisfied in most applications of interest: given two object instances, a user is quickly able to determine which one is preferred. The remaining two conditions are challenging, and addressing them constitutes the bulk of the technical work that follows.

[*]This work was supported in part by NSF grant IIS-1250880.

Proceedings of the VLDB Endowment, Vol. 8, No. 11
Copyright 2015 VLDB Endowment 2150-8097/15/07.

To summarize, this paper has the following contributions:

- We formalize the problem of preference learning by pairwise comparison on structured data, and define two subproblems, *preference estimation* and *comparison selection* (Section 2).
- We study the problem of preference estimation based on pairwise comparison feedback. We transform the problem to a maximum margin optimization problem, which can be easily solved by a typical SVM (Section 3).
- We develop a theoretical analysis of the comparison selection problem and introduce *adaptive comparison selection*, with effectiveness comparable to a theoretical optimum (Section 4).
- We introduce a new type of query, namely *orthogonal query*, to facilitate the adaptive comparison selection. We further propose S-tree, an innovative high-dimensional index on unit sphere for efficient orthogonal query execution (Section 5).
- We implement a system integrating all these parts. We show that our adaptive system significantly outperforms a random system on both synthetic and real data, with either simulated or real feedback. We also show our preference learning approach is much more effective than existing approaches, and our S-tree index can be easily built and supports our orthogonal queries in interactive speed (Section 6).

2. PROBLEM FORMALIZATION

We situate our problem in the context of quantitative preference [1, 19]. We denote an entity by e and the set of all entities by $\mathbf{E}$. Without loss of generality, we define the preference to be a function $\mathcal{O} : \mathbf{E} \rightarrow \mathbb{R}$. Intuitively, we say e is *preferable* to e' if $\mathcal{O}(e) > \mathcal{O}(e')$, and vice versa. Transitivity is trivially implied. Given N input entities, our task is to learn the user preference function $\mathcal{O}$, which is not directly observable. We assume no ties and will further discuss this assumption at the end of Section 3 and 4.

Definition 1 (Pairwise Comparison and Feedback) *Given an entity set* $\mathbf{E}$ *and a preference* $\mathcal{O}$*, a* pairwise comparison *is an ordered pair* $\langle e, e' \rangle \in \mathbf{E} \times \mathbf{E}$*. The* feedback *is a function* $\mathcal{C} : \mathbf{E} \times \mathbf{E} \rightarrow \{-1, 1\}$*, such that:*

$$\mathcal{C}(\langle e, e' \rangle) = \begin{cases} 1 & \textit{if } \mathcal{O}(e) > \mathcal{O}(e') \\ -1 & \textit{if } \mathcal{O}(e) < \mathcal{O}(e') \end{cases} \tag{1}$$

$\mathbf{D} \subseteq \mathbf{E} \times \mathbf{E}$ *is a set of pairwise comparisons, with feedback* $\mathcal{C}(\mathbf{D}) = \{\mathcal{C}(\langle e, e' \rangle) | \langle e, e' \rangle \in \mathbf{D}\}$*.*

Since $\mathcal{C}(\langle e, e' \rangle) = -\mathcal{C}(\langle e', e \rangle)$, we only need to consider one of the two. We define the *candidate comparisons* of $\mathbf{E}$ to be $\mathcal{D}(\mathbf{E}) = \{\langle e_i, e_j \rangle | e_i, e_j \in \mathbf{E}; i < j\}$. We now define the problem:

Definition 2 (Preference Learning by Pairwise Comparison) *Given a set of entities* $\mathbf{E}$*, an unobservable preference* $\mathcal{O}$*, choose a set of pairwise comparisons* $\mathbf{D} \subseteq \mathcal{D}(E)$*, and derive* $\mathcal{O}$ *based on* $\mathcal{C}(\mathbf{D})$*.*

While this problem definition is general, it can be impractical if the number of comparisons needed ($|\mathbf{D}|$) is large. Clearly, pairwise comparison of each pair in $\mathcal{D}(\mathbf{E})$ is enough, requiring that the user labels $N(N-1)/2$ pairs. We can improve this quite a bit if we follow a standard sorting algorithm, where in the worst case we need the user to label $N \cdot log(N)$ comparisons to derive the complete preference $\mathcal{O}$. Even in the best case, we need $N - 1$ comparisons to derive $\mathcal{O}$. Unfortunately, N can often be large in real scenarios. For instance, there may be hundreds of potential hotels and thousands of potential movies to consider. It is not reasonable to expect the user to label N pairs. In fact, one major motivation to learn a preference function is so that we can determine a user's preference for an item with only limited prior user input.

How can we infer a relative preference for an object for which the user has told us nothing? This is what we need to do, if we cannot afford N comparisons. One direction is followed by recommender systems, which try to guess based on other users' preference for the object. However, such schemes work well only if we have access to large communities of somewhat similar users. They further require that every object be rated by at least some users. These assumptions may not hold in many real applications. In such cases, we may still be able to induce a preference as a function of object attributes.

For instance, in the hotel booking scenario, a hotel may have attributes such as stars, location, price, etc. Intuitively, the user preference should be correlated with these attributes. As a result, by looking at the user feedbacks to several comparisons, even if we do not have any comparison involving a specific hotel, we should still be able to predict how likely the user would prefer it, according to its attributes. This is the direction we explore in this paper.

For this reason, we model each entity as a structured tuple. Specifically, we assume the set of attributes is universal across the entity set, and denote it by $\mathrm{A} = \{\mathrm{A}_1, \mathrm{A}_2, ..., \mathrm{A}_\mathrm{k}\}$. We use $e[\mathrm{A}_\mathrm{i}]$ to denote the value on attribute A_i of entity e. For convenience, we further assume all these attribute values are numeric. (Categorical attributes can be mapped to numeric values using standard SVM convention). For a given e, we represent the entity by a k dimensional vector $\vec{e}$:

$$\vec{e} = (e[A_1], e[A_2], ..., e[A_k])^T \tag{2}$$

Furthermore, for each e, we assume $\mathcal{O}(e)$ to be a linear combination of these attribute values. Specifically:

$$\mathcal{O}(e) = \vec{w} \cdot \vec{e} \tag{3}$$

where $\vec{w}$ is a k dimensional weight vector. Here we omit the constant offset since it has no impact on comparisons. Similarly, because the comparisons are independent of the magnitude of $\vec{w}$, we assume $\|\vec{w}\| = 1$. Note that, under these assumption, $\vec{w}$ uniquely characterizes $\mathcal{O}$. For this reason, we call $\vec{w}$ the *preference vector*.

In practice, we do not know that the preference function is linear. However, it is common to build linear models as approximations of reality, and they often turn out to be good enough. If need be, individual attribute values can be scaled non-linearly, for example, through the use of a logarithm or some kernel function. Thus, linear regression is used widely, as are linear models in machine learning. We thus refine our preference learning problem as the following.

Definition 3 (Preference Learning by Pairwise Comparison) *Given a set of entities* $\mathbf{E}$*, where each entity e is a k-dimensional vector, and a preference function* $\mathcal{O}(e) = \vec{w} \cdot \vec{e}$ *where* $\vec{w}$ *is a normalized k-dimensional vector not directly observable, select a set of pairwise comparisons* $\mathbf{D} \subseteq \mathcal{D}(\mathbf{E})$*, and estimate* $\vec{w}$ *based on* $\mathcal{C}(\mathbf{D})$*.*

Definition 3 comprises two subproblems: 1) select a subset of the candidate pairwise comparisons to ask the user for feedback and 2) estimate the preference vector $\vec{w}$ based on the feedback. We name the first subproblem **pairwise comparison selection** and the second **preference estimation**. We first discuss the preference estimation problem in Section 3, and then study the pairwise comparison selection problem in Section 4.

3. PREFERENCE ESTIMATION

Suppose we have already selected a set of pairwise comparisons $\mathbf{D}$, in this section we focus on estimating the weight vector $\vec{w}$ based on the user feedback $\mathcal{C}(\mathbf{D})$. We first describe some intuitions about the preference estimation problem under the linear preference model, and then formalize the problem and propose an algorithm to estimate $\vec{w}$. We denote our *estimate* of $\vec{w}$ by $\hat{\vec{w}}$ and the

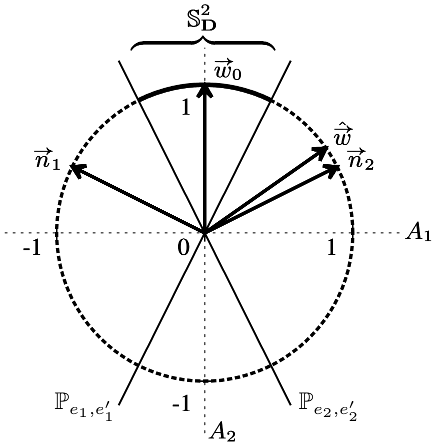


Figure 1: A Simple Preference Estimation in 2-D.

true value of $\vec{w}$ by $\vec{w}_0$. To give an intuitive geometric interpretation, we begin with an assumption that the user makes no error in reporting pairwise comparisons. In other words, for every pair $\langle e, e' \rangle \in \mathbf{D}$, $\mathcal{C}(\langle e, e' \rangle) = 1$ if and only if $(\vec{e} - \vec{e}') \cdot \vec{w}_0 > 0$. We will relax this assumption at the end of this section.

Recall that $\vec{w}$ is a k-dimensional vector and $\|\vec{w}\| = 1$, all $\vec{w}$ form a **hypersphere** $\mathbb{S}^k = \{\vec{w} | \|\vec{w}\| = 1\}$ in the k-dimensional space. Now, imagine we assign a pairwise comparison $\langle e, e' \rangle$ to the user. The user returns 1 if e is preferable to e', and -1 if e' is preferable to e. If we review Equation 1, this can be rewritten as[1]:

$$\mathcal{C}(\langle e, e' \rangle) \cdot (\mathcal{O}(e) - \mathcal{O}(e')) > 0 \tag{4}$$

By substituting Equation 3, we obtain:

$$\mathcal{C}(\langle e, e' \rangle) \cdot (\vec{e} - \vec{e}') \cdot \vec{w}_0 > 0 \tag{5}$$

This implies the following geometric intuition. A pairwise comparison $\langle e, e' \rangle$ and its feedback $\mathcal{C}(\langle e, e' \rangle)$ uniquely identify a **hyperplane** with normal vector $\mathcal{C}(\langle e, e' \rangle) \cdot (\vec{e} - \vec{e}')$ in the k-dimensional vector space of $\vec{w}$. The hyperplane cuts $\mathbb{S}^k$ into two symmetric hemispheres, and $\vec{w}_0$ must be on the hemisphere $\{\vec{w} \mid \vec{w} \in \mathbb{S}^k, \mathcal{C}(\langle e, e' \rangle) \cdot (\vec{e} - \vec{e}') \cdot \vec{w} > 0\}$. In other words, each such hyperplane prunes half of $\mathbb{S}^k$ on which $\vec{w}_0$ could reside. For this reason, we call such a hyperplane the **pruning plane** and denote it by $\mathbb{P}_{\langle e, e' \rangle}$. Since the pruning is independent of the magnitude of the normal vector, we normalize it and define $\vec{n}_{\langle e, e' \rangle} = \mathcal{C}(\langle e, e' \rangle) \cdot (\vec{e} - \vec{e}') / \|\vec{e} - \vec{e}'\|$. We further denote the hemisphere on the same side of $\vec{n}_{\langle e, e' \rangle}$ by $\mathbb{S}^k_{\langle e, e' \rangle}$, and call it the **remaining hemisphere**. Formally, $\mathbb{S}^k_{\langle e, e' \rangle} = \{\vec{w} | \vec{n}_{\langle e, e' \rangle} \cdot \vec{w} > 0, \vec{w} \in \mathbb{S}^k\}$.

Intuitively, the more pairwise comparison feedbacks we obtain, the fewer possible values remain for $\vec{w}$. Specifically, given a set of pairwise comparisons $\mathbf{D}$, the remaining possible values for $\vec{w}$ can be characterized by the intersection of all the remaining hemispheres $\bigcup_{\langle e, e' \rangle \in \mathbf{D}} \mathbb{S}^k_{\langle e, e' \rangle}$, which forms a *spherical polygon* in k dimensions. We denote this spherical polygon by $\mathbb{S}^k_{\mathbf{D}}$ and simply call it the **remaining sphere**.

Under our error-free assumption, $\vec{w}_0$ must exist in the remaining sphere $\mathbb{S}^k_{\mathbf{D}}$. Therefore, we should estimate $\vec{w}$ within $\mathbb{S}^k_{\mathbf{D}}$. The following example illustrates the intuition in two dimensions.[2]

Example 1 (Simple 2-D Estimation) *Suppose our entities have only two attributes, A_1 and A_2. Furthermore, we assume an extreme case, where the preference is only determined by A_2. In other words, $\vec{w}_0 = (0, 1)$. Suppose we have two comparisons, $\langle e_1, e_1' \rangle$ and $\langle e_2, e_2' \rangle$, where $\vec{e_1} - \vec{e_1}' = (2, 1)$ and $\vec{e_2} - \vec{e_2}' = (-2, 1)$. Under the error-free assumption, $\mathcal{C}(\langle e_1, e_1' \rangle) = \mathcal{C}(\langle e_2, e_2' \rangle) = 1$. As a result, $\vec{n}_1 = (2, 1)/\sqrt{5}$ and $\vec{n}_2 = (-2, 1)/\sqrt{5}$. In 2-dimension, the two pruning planes corresponding to $\vec{n}_1$ and $\vec{n}_2$ degrade to two lines shown in Figure 1. The remaining sphere $\mathbb{S}^2_{\mathbf{D}}$ degrades to the bold arc on the unit circle. In this case, a reasonable estimate of $\hat{\vec{w}}$ lies in the middle of $\vec{n}_1$ and $\vec{n}_2$, which overlaps $\vec{w}_0$.*

The above example indicates that the estimate in the "middle" of the normal vectors of all pruning planes may be a "good" estimate. In order to formalize this intuition, we first define such "goodness".

Let us first examine the simple case, where the user submits a feedback $\mathcal{C}(\langle e, e' \rangle)$ to a pairwise comparison $\langle e, e' \rangle$. To recall, this creates a pruning plane $\mathbb{P}_{\langle e, e' \rangle}$ with normal vector $\vec{n}_{\langle e, e' \rangle} = \mathcal{C}(\langle e, e' \rangle) \cdot (\vec{e} - \vec{e}')$. On one extreme, if $\vec{w}_0$ is orthogonal to $\mathbb{P}$ (parallel to $\vec{n}$), we can tell this is a very *straightforward* comparison. Because, the difference between $\vec{e}$ and $\vec{e}'$ is maximized by $\vec{w}_0$. Thus the user should have little difficulty in answering the comparison. On the other extreme, if $\vec{w}_0$ is almost parallel to $\mathbb{P}$ (orthogonal to $\vec{n}$), the comparison would be very *difficult*. Because the difference between $\vec{e}$ and $\vec{e}'$ is only slightly reflected by the projection on $\vec{w}_0$. Intuitively, we prefer estimate $\vec{w}$ to be parallel to $\vec{n}$, since it tends to render the comparison more *confident* for the user. The notion of confidence is formalized below:

Definition 4 (Confidence) *Given a pairwise comparison $\langle e, e' \rangle$ and an estimated weight vector $\hat{\vec{w}}$, the confidence of this estimate is $\mathrm{C}^{\hat{\vec{w}}}_{\langle e, e' \rangle} = \vec{n}_{\langle e, e' \rangle} \cdot \hat{\vec{w}}$.*

Note that, the confidence must be positive according to the error-free assumption.

Informally, our goal is to maximize this confidence for all comparisons. If there is only one comparison, the estimate parallel to $\vec{n}$ is the optimal solution. In case of multiple comparisons, there are various ways to define the overall confidence. For example, we could define the overall confidence to be the sum of all individual confidences, which leads to the following problem definition:

Definition 5 (Preference Estimation by Total Confidence) *Given a set of pairwise comparison $\mathbf{D}$ and the user feedback $\mathcal{C}(\mathbf{D})$, find a preference estimate $\hat{\vec{w}}$, such that $\sum_{\langle e, e' \rangle \in \mathbf{D}} \mathrm{C}^{\hat{\vec{w}}}_{\langle e, e' \rangle}$ is maximized.*

The solution to this problem is trivial, where we estimate $\hat{\vec{w}}$ to be parallel to $\sum \vec{n}_{\langle e, e' \rangle}$. However, such $\hat{\vec{w}}$ may not always satisfy $\mathrm{C}^{\hat{\vec{w}}}_{\langle e, e' \rangle} > 0$. For instance, in Example 1, if we continue to receive dozens of comparison feedbacks with normal vectors around $\vec{n}_2$, $\hat{\vec{w}}$ will be pulled to $\vec{n}_2$ and fall out of $\mathbb{S}^2_{\mathbf{D}}$, as shown in Figure 1. This violates our error free assumption.

In fact, we prefer a metric such that repetitive comparisons with same or similar normal vectors do *not* shift our final estimate. In the previous example, ideally, no matter how many comparisons overlap $\vec{n}_1$ or $\vec{n}_2$, a fair estimate of $\vec{w}$ should always point to the middle, the same as $\vec{w}_0$ in Figure 1.

In fact, if we have various additional comparisons with normal vectors between $\vec{n}_1$ and $\vec{n}_2$, they should contribute little to the estimate, because their ability to constrain $\mathbb{S}^2_{\mathbf{D}}$ is dominated by $\vec{n}_1$ and $\vec{n}_2$. In other words, the only comparisons that matter are those whose normal vectors are *most orthogonal* to $\vec{w}_0$. And according to our definition of confidence, these are exactly the comparisons the user is *least confident* with, which matches our intuition. In this case, our goal is to maximize those confidence values, which is formalized by the following revised problem statement:

[1] We use the dot operator to represent both scalar multiplication and vector inner product when the context is clear.

[2] Hereafter, we will omit the subscript $\langle e, e' \rangle$ of $\vec{n}_{\langle e, e' \rangle}$ for readibility when the context is clear.

Definition 6 (Preference Estimation by Maximal Least Confidence) *Given a set of pairwise comparison $\mathbf{D}$ and the user feedback $\mathcal{C}(\mathbf{D})$, find a preference vector estimate $\hat{\vec{w}}$, such that $\min_{\langle e,e'\rangle \in \mathbf{D}} \mathrm{C}^{\hat{\vec{w}}}_{\langle e,e'\rangle}$ is maximized. Or formally, obtain:*

$$\begin{aligned} &\operatorname*{argmax}_{\vec{w}} \min_{\langle e,e'\rangle \in \mathbf{D}} \mathrm{C}^{\vec{w}}_{\langle e,e'\rangle} \\ &\textit{subject to:} \quad \|\vec{w}\| = 1 \end{aligned} \tag{6}$$

Since the magnitude of $\vec{w}$ does not matter, we remove the constraint $\|\vec{w}\| = 1$ and transform the above problem into the following standard maximum margin optimization problem:

$$\begin{aligned} &\operatorname*{argmin}_{\vec{w}} \|\vec{w}\| \quad \text{subject to:} \\ &\forall \langle e,e'\rangle \in \mathbf{D}, \vec{n}_{\langle e,e'\rangle} \cdot \vec{w} \geq 1 \end{aligned} \tag{7}$$

In this revised problem, the margin to be maximized is our minimal confidence value. Therefore, we can use a linear SVM to solve it efficiently. Specifically, for each comparison the user made, we create a positive example $(\vec{n}, 1)$ and a negative example $(-\vec{n}, -1)$ and call SVM to train all the support vectors. We then use the sum of these support vectors weighted on α as our estimate $\hat{\vec{w}}$. Details about the algorithm can be found in Section 5.2.

In case the user feedback is error-prone, no $\vec{w}$ will yield a positive confidence margin. To solve this, we adopt the typical slack variant methods commonly used in SVM to enable error-tolerance. The problem is then revised as follows.

Definition 7 (Noise-tolerance Preference Estimation by Maximal Least Confidence) *Given a set of pairwise comparison $\mathbf{D}$ and the user feedback $\mathcal{C}(\mathbf{D})$, find a preference vector estimate $\hat{\vec{w}}$, such that:*

$$\begin{aligned} &\operatorname*{argmin}_{\vec{w}} \|\vec{w}\| + C \sum \xi_{\langle e,e'\rangle} \quad \textit{subject to:} \\ &\forall \langle e,e'\rangle \in \mathbf{D}, \vec{n}_{\langle e,e'\rangle} \cdot \vec{w} \geq 1 - \xi_{\langle e,e'\rangle} \end{aligned} \tag{8}$$

where C is the slack variant indicating the error tolerance level.

This also implies the following. If the entities being compared have very similar preferences, the user does not need to explicitly specify a tie. Even if she gives the wrong feedback, the error contributes little to the final precision.

4. COMPARISON SELECTION

So far we have discussed how to estimate the weight vector $\vec{w}$ given a set of pairwise comparisons $\mathbf{D}$. We now consider how these pairwise comparisons are selected. Admittedly, a randomly selected set of comparisons could still lead to a reasonably good estimate. The question is, can we do better?

To give a clear geometric interpretation, we keep the error-free assumption. Later, we will illustrate how user errors are fixed when Noise-tolerance Preference Estimation is applied.

To recall, each comparison $\langle e, e'\rangle$ with its feedback $\mathcal{C}(\langle e, e'\rangle)$ uniquely identifies a pruning plane $\mathbb{P}^k_{\langle e,e'\rangle}$, which prunes half of the k-dimensional sphere $\mathbb{S}^k$. These planes keep pruning and finally restrict $\vec{w}$ to a spherical polygon $\mathbb{S}^k_{\mathbf{D}}$, the remaining sphere.

Both $\vec{w}_0$ and our estimate $\hat{\vec{w}}$ lie on this remaining sphere. Obviously, the best estimate is the one that is close to $\vec{w}_0$. We define the **precision** of an estimate $\hat{\vec{w}}$ to be the inner product $\hat{\vec{w}} \cdot \vec{w}_0$. Intuitively, the higher the precision is, the better the estimate $\hat{\vec{w}}$ is. Given this definition, the worst precision we could have given $\mathbb{S}^k_{\mathbf{D}}$ and $\vec{w}_0$ is $\min_{\hat{\vec{w}} \in \mathbb{S}^k_{\mathbf{D}}} \hat{\vec{w}} \cdot \vec{w}_0$. Considering the fact that $\vec{w}_0$ can be anywhere inside $\mathbb{S}^k_{\mathbf{D}}$, the worst precision given only $\mathbb{S}^k_{\mathbf{D}}$ is $\min_{\vec{w}_1, \vec{w}_2 \in \mathbb{S}^k_{\mathbf{D}}} \vec{w}_1 \cdot \vec{w}_2$. We define the precision of $\mathbb{S}^k_{\mathbf{D}}$ by this worst precision and denote it by $\mathcal{P}(\mathbb{S}^k_{\mathbf{D}})$. We define the angle between such $\vec{w}_1$ and $\vec{w}_2$ in the worst case to be the **diameter** of $\mathbb{S}^k_{\mathbf{D}}$ and denote it by $\mathcal{D}(\mathbb{S}^k_{\mathbf{D}})$. Since $\|\vec{w}\| = 1$, $\mathcal{P}(\mathbb{S}^k_{\mathbf{D}}) = \cos(\mathcal{D}(\mathbb{S}^k_{\mathbf{D}}))$.

In other words, the diameter of the remaining sphere determines its precision. Intuitively, the more user comparisons we have, the smaller the remaining sphere is, the smaller its diameter is and the higher the precision we obtain. However, because each comparison places a user burden, the total number of cuts is limited in practice. Also, due of user error, we cannot indefinitely refine the remaining sphere to achieve arbitrarily high precision. Based on these observations, we formalize the comparison selection problem:

Definition 8 (Pairwise Comparison Selection) *Given a set of entities $\mathbf{E}$ and a positive integer M, find a sequence of pairwise comparisons $\mathbf{D} = \{\langle e_1, e'_1\rangle, \langle e_2, e'_2\rangle, ..., \langle e_M, e'_M\rangle\}$, where $\langle e_i, e'_i\rangle \in \mathcal{D}(\mathbf{E})$, $i \in 1..M$, such that $\mathcal{P}(\mathbb{S}^k_{\mathbf{D}})$ is maximized.*

In the remainder of this section, we start with a naïve approach with random selection, and theoretically analyze it. We then propose a novel selection strategy that is theoretically optimal.

4.1 A Naïve Approach

To recall, each comparison identifies a pruning plane that cuts the k-sphere $\mathbb{S}^k$ across origin into two equal halves and prunes one of them according to the user feedback. Consequently, all the N comparisons will cut $\mathbb{S}^k$ into many small pieces, each of which is a spherical polygon. Under the error-free assumption, exactly one of these spherical polygons forms the remaining sphere $\mathbb{S}^k_{\mathbf{D}}$.

Here we analyze the best case average precision one can obtain after M randomly chosen cuts. Since $\vec{w}_0$ can fall into any of the remaining sphere, the best case is achieved when the sphere $\mathbb{S}^k$ is uniformly cut. Using $\mathcal{V}(\cdot)$ to denote volume, the best case average volume of the remaining sphere can be estimated as:

$$\mathcal{V}(\mathbb{S}^k_{\mathbf{D}}) = \frac{\mathcal{V}(\mathbb{S}^k)}{\mathcal{M}^k(M)} \tag{9}$$

where $\mathcal{M}^k(M)$ is the number of spherical polygons cut by M pruning planes in the k-dimensional space. According to the literature [17]:

$$\mathcal{M}^k(M) = 2 \cdot \left[\binom{M-1}{0} + \binom{M-1}{1} + ... + \binom{M-1}{k-1}\right] \tag{10}$$

Also, the volume of a k-sphere[3] with radius R is:

$$\mathrm{V}_k = 2 \cdot \frac{\pi^{k/2} R^{k-1}}{\Gamma(\frac{k}{2})} \tag{11}$$

Since $\mathbb{S}^k$ has radius one, its volume is:

$$\mathcal{V}(\mathbb{S}^k) = 2 \cdot \frac{\pi^{k/2}}{\Gamma(\frac{k}{2})} \tag{12}$$

By substituting Equation 12 into Equation 9, we have:

$$\mathcal{V}(\mathbb{S}^k_{\mathbf{D}}) = \frac{2}{\mathcal{M}^k(N)} \cdot \frac{\pi^{k/2}}{\Gamma(\frac{k}{2})} \tag{13}$$

On the other hand, given the remaining sphere with volume $\mathcal{V}(\mathbb{S}^k_{\mathbf{D}})$, the best precision is achieved when the remaining sphere is as evenly distributed as possible. When the remaining sphere is small enough, this can be approximated as a ball in the $k-1$ dimension. According to the literature, the volume of a k-ball with radius R is:

$$\mathrm{U}_k = \frac{\pi^{k/2} R^k}{\Gamma(1 + \frac{k}{2})} \tag{14}$$

[3] We define the k-sphere in the k-dimensional space instead of the $(k-1)$-dimensional space.

Therefore, we have:

$$\mathcal{V}(\mathbb{S}^k_{\mathbf{D}}) = \mathrm{U}_{k-1} = \frac{\pi^{\frac{k-1}{2}} R^{k-1}}{\Gamma(\frac{k+1}{2})} \tag{15}$$

Substituting Equation 15 into Equation 13 and solving R results:

$$R = \left(\frac{2\sqrt{\pi}}{\mathcal{M}^k(M)} \cdot \frac{\Gamma(\frac{k+1}{2})}{\Gamma(\frac{k}{2})} \right)^{\frac{1}{k-1}} \tag{16}$$

For a small remaining sphere $\mathbb{S}^k_{\mathbf{D}}$ shaped like a ball in the (k-1)-dimensional space, its diameter $\mathcal{D}(\mathbb{S}^k_{\mathbf{D}})$ can be estimated by $2R$. Therefore, its precision can be obtained by:

$$\mathcal{P}(\mathbb{S}^k_{\mathbf{D}}) = \cos(2R) \tag{17}$$

Substituting Equation 16 into Equation 17 results in:

$$\mathcal{P}(\mathbb{S}^k_{\mathbf{D}}) = \cos \left(2 \cdot \left(\frac{2\sqrt{\pi}}{\mathcal{M}^k(M)} \cdot \frac{\Gamma(\frac{k+1}{2})}{\Gamma(\frac{k}{2})} \right)^{\frac{1}{k-1}} \right) \tag{18}$$

As a final note for this section, since $\mathcal{M}^k(M)$ is upper bounded by $2 \cdot (M-1)^{k-1}$ for $M > 1$, we have:

$$\mathcal{V}(\mathbb{S}^k_{\mathbf{D}}) \geq \frac{\mathcal{V}(\mathbb{S}^k)}{2 \cdot (M-1)^{k-1}} = \frac{1}{(M-1)^{k-1}} \cdot \frac{\pi^{k/2}}{\Gamma(\frac{k}{2})} \tag{19}$$

for $M > 1$. That is to say, for any given dimension k, one cannot reduce the volume of the remaining sphere more than *polynomially* with respect to the number of comparisons using a random cutting strategy, where the volume determines final precision. If we want to break this limit, a new cutting strategy has to be innovated.

4.2 Binary Cutting Strategy

In this section, we consider if we can perform any better than a random cutting strategy. Without loss of generality, we assume that given an arbitrary hyperplane, there always exists in the candidates a comparison whose pruning plane overlaps the given hyperplane. We will remove this assumption in Section 4.3.

If we have only one comparison to prepare, the problem is straightforward. Indeed, a random cut of the initial sphere $\mathbb{S}^k$ is the optimal choice. This is because, no matter what the user feedback is, the corresponding pruning plane will prune half of the sphere, or reduce half of the volume. However, if we continue to perform more cuts, their positions matter.

Example 2 (Cutting Strategy) *We continue to illustrate the simple case in two-dimension, where $\mathbb{S}^2$ degrades to a unit circle, as shown in Figure 2. Suppose we first ask the user to compare entity e_1 and e_1'. This comparison identifies a pruning plane $\mathbb{P}_{\langle e_1, e_1' \rangle}$ with normal vector $\vec{e_1} - \vec{e_1}'$, as shown in Figure 2. We further assume the user feedback $\mathcal{C}(\langle e_1, e_1' \rangle)$ equals 1. Consequently, the plane prunes the bottom left half of the unit circle, leaving a remaining sphere highlighted by the bold arc $\widehat{QT}$.*

Now consider the next comparison. Assume there are two candidates: $\langle e_2, e_2' \rangle$ and $\langle e_3, e_3' \rangle$, with corresponding pruning planes and normal vectors illustrated in Figure 2. If we choose $\langle e_2, e_2' \rangle$ as the next comparison, the pruning effect will depend on the user feedback. Specifically, if $\mathcal{C}(\langle e_2, e_2' \rangle) = 1$, the top-left side of $\mathbb{P}_{\langle e_2, e_2' \rangle}$ will be pruned, leaving arc $\widehat{RT}$. If $\mathcal{C}(\langle e_2, e_2' \rangle) = -1$, the bottom-right part of $\mathbb{P}_{\langle e_2, e_2' \rangle}$ will be pruned and arc $\widehat{QR}$ will remain.

This is not an ideal choice, because in the worst case, only a small portion of the remaining sphere ($\widehat{QR}$) will be pruned. Overall, it will result in a worse average precision because our precision function is concave with respect to volume. In contrast, if we choose $\langle e_3, e_3' \rangle$ instead, regardless of the user feedback, half of the remaining sphere (either $\widehat{QS}$ or $\widehat{ST}$) will be pruned.

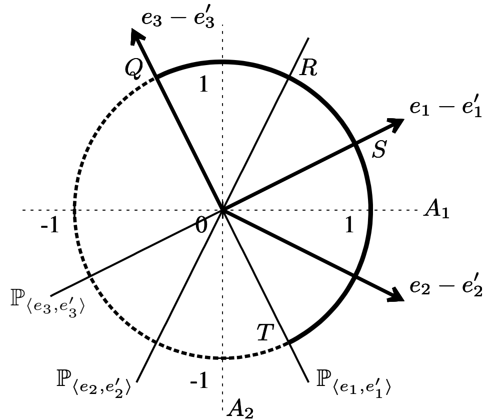

$e_3 - e_3'$
Q
1
R
$e_1 - e_1'$
S
-1
0
1
A_1
$e_2 - e_2'$
$\mathbb{P}_{\langle e_3, e_3' \rangle}$
T
$\mathbb{P}_{\langle e_2, e_2' \rangle}$
-1
$\mathbb{P}_{\langle e_1, e_1' \rangle}$
A_2

Figure 2: Binary Cutting Example in 2-Dimensions. (Example 2)

Example 2 elicits the following theory:

Theory 1 (Binary Cutting Strategy) *In order to maximize the best case average precision, each comparison should cut the current remaining sphere into two equal halves.*

The intuition behind the theory is similar to the traditional binary search. We omit the proof due to space.

If we manage to choose a sequence of comparisons strictly satisfying the condition in Theorem 1, after M comparisons, the volume of the remaining sphere is:

$$\mathcal{V}(\mathbb{S}^k_{\mathbf{D}}) = \frac{\mathcal{V}(\mathbb{S}^k)}{2^M} \tag{20}$$

Compared with Equation 13 for random cut, this is much better because we can reduce the volume of the remaining sphere *exponentially* with respect to the number of comparisons.

By substituting Equation 12 into Equation 20, we obtain:

$$\mathcal{V}(\mathbb{S}^k_{\mathbf{D}}) = \frac{1}{2^{M-1}} \cdot \frac{\pi^{k/2}}{\Gamma(\frac{k}{2})} \tag{21}$$

Following a similar analysis as in Section 4.1, we can obtain the theoretical upper bound of the precision after M comparisons:

$$\mathcal{P}(\mathbb{S}^k_{\mathbf{D}}) = \cos \left(2 \cdot \left(\frac{\sqrt{\pi}}{2^{M-1}} \cdot \frac{\Gamma(\frac{k+1}{2})}{\Gamma(\frac{k}{2})} \right)^{\frac{1}{k-1}} \right) \tag{22}$$

By fixing $\mathcal{P}(\mathbb{S}^k_{\mathbf{D}})$ and solving M in the above equation, we obtain the *number of optimal cuts* to achieve a certain precision:

$$M_{optimal} = \frac{\log_2 \pi}{2} + \log_2 \frac{\Gamma(\frac{k+1}{2})}{\Gamma(\frac{k}{2})} + (k-1) \cdot \log_2 \frac{2}{\arccos(\mathcal{P}(\mathbb{S}^k_{\mathbf{D}}))} \tag{23}$$

On the other hand, we have:

$$\lim_{k \to +\infty} \log_2 \frac{\Gamma(\frac{k+1}{2})}{\Gamma(\frac{k}{2})} = \frac{\psi^{\{0\}}(\frac{k}{2})}{2} = \frac{\log(\frac{k}{2})}{2} \tag{24}$$

In Equation 23, the first term is a constant, the second term converges to $(\log_2 k - 1)/2$, and the third term is the product of $k - 1$ and a value determined by the target precision. To give an intuition, by setting the precision to 0.95, we need to perform approximately 2.65 optimal cuts per additional dimension.

4.3 Adaptive Comparison Selection

In the previous section we presented a theoretical analysis of the optimal comparison sequence. To recall, each time we desire the next comparison to cut the remaining sphere into two equal halves. Unfortunately, such an *optimal* comparison rarely exists in practice, because we have a limited number of comparison candidates. In this section, we instead examine the problem of finding *adaptively* the next comparison that cuts the remaining sphere *as equally as possible*. We also show how this strategy deals with noisy answers.

We start by following the two-dimensional scenario in Example 2. After the initial cut, the remaining sphere degrades to arc QT. Since the *center* of QT is S, one should select a comparison whose pruning plane is closest to the center S. This is trivial in 2-d, because the pruning plane degrades to a line passing through the origin. Thus, we only need to find the comparison whose decision plane (line) has the smallest angle with the line passing through the origin and the "center" of the remaining sphere (arc), which can be accomplished by a simple binary search. However, the following example demonstrates its complexity in higher dimensions.

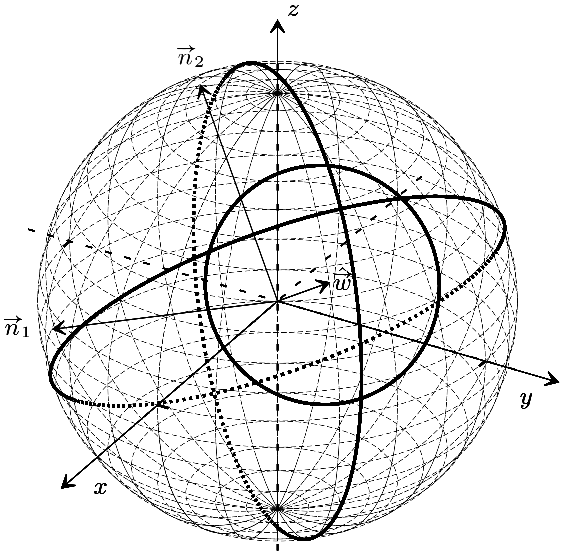


Figure 3: Two Pairwise Comparison Candidates in 3-Dimension

Example 3 (Comparison Selection in 3D) *For simplicity, assume the remaining sphere $\mathbb{S}^k_{\mathbf{D}}$ is uniformly distributed around center $\hat{\vec{w}}$ on the unit 3-sphere, as shown in Figure 3. Suppose we have two candidate comparisons: one with pruning plane identified by normal vector $\vec{n}_1$ and the other with pruning plane identified by $\vec{n}_2$. Intuitively, the plane with $\vec{n}_1$ is the better pick because it cuts the remaining sphere more evenly.*

This examples indicates the following two points. First, the problem is non-trivial in higher dimension, because there is no static ordering of the candidate comparisons. Indeed, the order of candidates are dependent on the current estimate $\hat{\vec{w}}$. Therefore, the simple binary search strategy in 2-d no longer applies. As a result, the naïve approach requires a scan of all the candidate comparisons, whose number is square of the total number of entities.

Second, the notion of "closeness" between the pruning plane and $\hat{\vec{w}}$ is no longer effective, because the pruning plane is a hyperplane in general. This leads us to think about the relationship between $\hat{\vec{w}}$ and the normal vector $\vec{n}$, which is a better characteristic of the pruning plane. In fact, given a pruning plane with normal vector $\vec{n}$ and a remaining sphere with estimated center $\hat{\vec{w}}$, we notice that the more *orthogonal* $\vec{n}$ is with respect to $\hat{\vec{w}}$, the more evenly the plane cuts the remaining sphere. To quantify this, we have:

Definition 9 (Orthogonality) *Given two unit vectors $\vec{u}$ and $\vec{v}$, the orthogonality between $\vec{u}$ and $\vec{v}$ is $1 - |\vec{u} \cdot \vec{v}|$. Given a remaining sphere with center estimate $\hat{\vec{w}}$ and a candidate pruning plane with normal vector $\vec{n}$, we also say the orthogonality between the pruning plane and the remaining sphere is $1 - |\vec{n} \cdot \hat{\vec{w}}|$.*

Since both vectors are normalized, their orthogonality ranges from zero to one. It achieves its maximum when $\vec{n} \cdot \hat{\vec{w}}$ equals zero, or when the pruning plane passes through $\hat{\vec{w}}$. For simplicity, hereafter we will use the normal vector, the pruning plane and the corresponding comparison interchangeably, if the context is clear.

We also note, even a comparison with orthogonality one may not cut the remaining sphere into exactly equal halves. This is because the remaining sphere in practice has a very complex geometry. To distinguish such comparison from the optimal comparison we discussed in Section 4.2 , we call such comparison the *best next comparison* and have the following problem definition.

Definition 10 (Adaptive Best Next Comparison Selection) *Given an estimate $\hat{\vec{w}}$, find among all the remaining pairwise comparisons $\mathcal{D}(\mathbf{E})\backslash\mathbf{D}$ the best comparison, such that the orthogonality between the comparison and $\hat{\vec{w}}$ is maximized. Formally, obtain:*

$$\underset{\langle e,e' \rangle \in \mathcal{D}(\mathbf{E})\backslash\mathbf{D}}{\operatorname{argmin}} \left| \vec{n}_{\langle e,e' \rangle} \cdot \hat{\vec{w}} \right| \tag{25}$$

This problem can be generalized as to find the vector that is most *orthogonal* to a given vector. This is very different from the traditional geographic queries which mostly focus on the nearest neighbors. To distinguish, we name this type of query **orthogonal query**.

Further, we give an intuition why this method will be noise-tolerance. Recall that the maximum margin optimization with slack variant relaxes the condition for those answers which is too hard to be satisfied. That is to say, when noise occurs, it will usually not used as the pruning plane since its condition is not satisfied. In the query selection phase, suppose one question is answered incorrectly, then the preference estimation will fall into another remaining sphere. However, as more and more questions are asked in the orthogonal plane of the estimation, the optimization function will choose to relax the condition of the noise comparison. Intuitively, the false plane are finally removed and the new estimation will live in the correct remaining sphere.

In case the user is not able to answer the most orthogonal comparison and has to specify a tie, our approach can be easily extended to return the top-k orthogonal comparisons, and ask the user the next most orthogonal question if she fails the previous one.

5. EVALUATING ORTHOGONAL QUERIES

From the previous section we know that the system must evaluate an orthogonal query to find good candidate pairs for obtaining user feedback next. Orthogonal queries in higher dimension are nontrivial because a naïve approach requires a scan of all the candidate comparisons. But do we really need to consider all the candidates? Assume $\hat{\vec{w}}$ points to the north pole of the sphere, all we care about are comparisons whose normal vectors are near the equator. We may not want to scan normal vectors near either polar circle because they are unlikely to be orthogonal to the north pole, especially when we are sure better choices do exist near the equator. In this section, we develop the notion of a spherical cap as an effective way to cluster similar candidates and organize them in a novel index structure called the **spherical tree**.

5.1 Spherical Cap

We would like to assign the normal vectors of all the candidate comparisons into smaller groups on the sphere for bounding and pruning. Ideally, these groups should have as few parameters as possible, while still being effective in pruning. A *Spherical Cap* offers such properties.

Definition 11 (Spherical Cap) *Given a center normal vector $\vec{c}$ and an angular radius θ, a* spherical cap *in dimension k, denoted by $\mathbb{S}^k_{\vec{c},\theta}$, is the set of all vectors on the unit k-sphere, such that the angle between these vectors and $\vec{c}$ is no larger than θ. Formally:*

$$\mathbb{S}^k_{\vec{c},\theta} = \{\vec{n} | \vec{n} \in \mathbb{S}^k \wedge \arccos(\vec{n} \cdot \vec{c}) \leq \theta\} \tag{26}$$

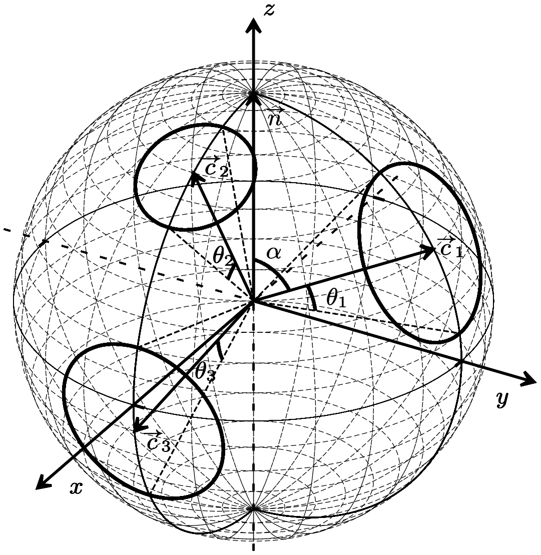


Figure 4: Three Spherical Caps on a Unit 3-Sphere

Figure 4 illustrates three spherical caps on the unit 3-Sphere. Each cap is identified by the center vector $\vec{c}$ and the angular radius θ. Given a normal vector $\vec{n}$ and a spherical cap, for instance, we can easily obtain the range of angle between $\vec{n}$ and the vectors in the spherical cap. For instance, in Figure 4, given normal vector $\vec{n}$ which points to the north pole and $\mathbb{S}_{\vec{c}_1,\theta_1}$, it is straightforward that the range is $[\alpha - \theta_1, \alpha + \theta_1]$, where $\alpha = \arccos(\vec{n} \cdot \vec{c}_1)$.

This indicates we can easily bound the orthogonality between any vector in a given spherical cap and a given vector as follows:

Definition 12 (Lower and Upper Bounds of Orthogonality) *Given a normal vector $\vec{n}$ and a spherical cap $\mathbb{S}_{\vec{c},\theta}$, the upper bound of the orthogonality between $\vec{n}$ and vectors in $\mathbb{S}_{\vec{c},\theta}$ is:*

$$\mathrm{UB}(\vec{n}, \mathbb{S}_{\vec{c},\theta}) = \begin{cases} 1 & \textit{if } \alpha - \theta < \pi/2 < \alpha + \theta \\ 1 - \min(|\cos(\alpha+\theta)|, |\cos(\alpha-\theta)|) & \textit{otherwise} \end{cases} \tag{27}$$

Similarly, the lower bound is:

$$\mathrm{LB}(\vec{n}, \mathbb{S}_{\vec{c},\theta}) = \begin{cases} 0 & \textit{if } \alpha < \theta \textit{ or } \alpha + \theta > \pi \\ 1 - \max(|\cos(\alpha+\theta)|, |\cos(\alpha-\theta)|) & \textit{otherwise} \end{cases} \tag{28}$$

We can use these bounds to prune candidate comparisons effectively. For example, in Figure 4, given an estimate $\hat{\vec{w}}$ that points to the north pole, we do not need to consider $\mathbb{S}_{\vec{c}_2,\theta_2}$ given $\mathbb{S}_{\vec{c}_1,\theta_1}$, because $\mathrm{LB}(\hat{\vec{w}}, \mathbb{S}_{\vec{c}_1,\theta_1}) > \mathrm{UB}(\hat{\vec{w}}, \mathbb{S}_{\vec{c}_2,\theta_2})$. However, we still have to consider $\mathbb{S}_{\vec{c}_3,\theta_3}$, because $\mathrm{UB}(\hat{\vec{w}}, \mathbb{S}_{\vec{c}_3,\theta_3}) > \mathrm{LB}(\hat{\vec{w}}, \mathbb{S}_{\vec{c}_1,\theta_1})$, even if they do not intersect geometrically.

Algorithm 1 shows how to find the best next comparison using spherical cap pruning, given all candidate normal vectors have been partitioned into a set of spherical caps (see Section 5.2).

The algorithm takes as input the estimate and the set of spherical caps. It maintains a priority queue (line 2), where the priority is the inverse of the orthogonality upper bound (i.e., the queue top has the smallest upper bound). It also records the current maximum of lower bound in the queue (line 3). For every spherical cap, we prune it if its upper bound is smaller than the current maximum queue lower bound (line 5). Otherwise we push it into the queue (line 11). Before the actual push we repeatedly remove the queue top if its upper bound is smaller than the lower bound of the enqueuing spherical cap (line 7-8), and update the queue lower bound accordingly (line 9-10). The vector most orthogonal to the estimate must be in one of the returned spherical caps.

Algorithm 1 Spherical Cap Pruning

```
 1: procedure SC-PRUNING($w$, $\mathbf{S}$)
 2:     $\mathbf{Q} \leftarrow \emptyset$
 3:     $maxLB \leftarrow 0$
 4:     for $\mathbb{S} \in \mathbf{S}$ do
 5:         if $\mathrm{UB}(w, \mathbb{S}) < maxLB$ then
 6:             continue
 7:         while $\mathrm{UB}(w, \mathbf{Q}.top()) < \mathrm{LB}(w, \mathbb{S})$ do
 8:             $\mathbf{Q}.pop()$
 9:         if $\mathrm{LB}(w, \mathbb{S}) > maxLB$ then
10:             $maxLB = \mathrm{LB}(w, \mathbb{S})$
11:         $\mathbf{Q}.push(\mathbb{S})$
12:     return $\mathbf{Q}$
```

As one can imagine, the algorithm strongly depends on how independent the [LB, UB] intervals are for the input spherical caps. In case these intervals largely overlap with each other, poor pruning results. We will address this problem in Section 5.3. In the next section, we first show how to construct the set of spherical caps.

5.2 Spherical Clustering

We need to cluster the candidate normal vectors into spherical caps before we can use the caps to find the best next comparison. An intuitive choice is K-means clustering based on cosine similarity (we omit the pseudo code due to space). In short, we adopt K-means clustering to iteratively partition the set of vectors into L groups, until the average center shift for these groups is less than a threshold (or the average dot product of the new and old centers is larger than p).

We need to further construct the spherical caps using these clusters. Meanwhile, we need these spherical caps to be as small as possible. Because the larger the clusters are, the larger the [LB, UB] intervals are and the higher chances they overlap, and the less effective our pruning algorithm becomes. Specifically, we have the following problem definition:

Definition 13 (Minimum Spherical Cap Construction) *Given a set of normal vectors $\mathbf{V}$, find the minimum spherical cap that contains $\mathbf{V}$. Formally, obtain:*

$$\mathbb{S}_{min}(\mathbf{V}) = \operatorname*{argmin}_{\mathbb{S}(\vec{c},\theta)} \theta \tag{29}$$

$$\textit{subject to:} \mathbf{V} \subset \mathbb{S}(\vec{c}, \theta) \tag{30}$$

Because given the center $\vec{c}$ and the constraint $\mathbf{V} \subset \mathbb{S}(\vec{c}, \theta)$, we can lower bound θ by $\max_{\vec{n} \in \mathbf{V}} \arccos(\vec{c} \cdot \vec{n})$, we can rewrite Equation 29 to:

$$\mathbb{S}_{min}(\mathbf{V}) = \operatorname*{argmin}_{\mathbb{S}(\vec{c},\theta)} \max_{\vec{n} \in \mathbf{V}} \arccos(\vec{c} \cdot \vec{n}) \tag{31}$$

Again, this is very similar to Equation 25, and can be reduced to the maximum margin classifier problem, where $\vec{c} \cdot \vec{n}$ is just the margin between he origin and the classifier boundary. Therefore, we can use the existing quadratic programming algorithm to easily find the supporting vectors, and calculate $\vec{c}$ and θ for $\mathbb{S}_{min}(\mathbf{V})$ accordingly, as shown in Algorithm 2.

We use $\mathbb{S}.V$ to refer to the set of vectors $\mathbf{V}$ that constructs the cap. Hereafter, by saying "vectors in a spherical cap $\mathbb{S}$" we mean $\mathbb{S}.V$ rather than the vectors in Definition 11. If $\mathbf{V}$ contains only one vector $\vec{n}$, the cap degrades to a point with $\vec{c} = \vec{n}$ and $\theta = 0$.

Algorithm 2 Spherical Cap Construction

```
procedure SC-CONSTRUCTION($\mathbf{N}$)
    $\mathbf{N}^+ \leftarrow \emptyset, \mathbf{N}^- \leftarrow \emptyset$
    for $n \in \mathbf{N}$ do
        $\mathbf{N}^+.push(n, 1)$
        $\mathbf{N}^-.push(-n, -1)$
    $(sv, \alpha) \leftarrow SVM(\mathbf{N}^+, \mathbf{N}^-)$
    $c = \Sigma sv \cdot \alpha$
    $\theta = \arccos \min_{v \in sv} c \cdot v$
    return $\mathbb{S}(c, \theta, \mathbf{V})$
```

5.3 Spherical Tree

Now we have the set of spherical caps and the pruning algorithm, let us focus on the question at the end of Section 5.1. In case the bounding intervals overlap a lot, the algorithm will return many spherical caps. Consequently, the follow-up algorithm has to scan all the vectors in these caps, and find the one with the maximum orthogonality, which is inefficient.

To improve this, we note that both input and output of Algorithm 1 are sets of spherical caps. For each of the returned caps, if we further partition them into smaller groups, we can recursively call Algorithm 1 for efficient branch-and-bound. This motivates a hierarchical index structure of the spherical caps, which is similar to the traditional R-Tree. We call this structure a **spherical tree**.

Definition 14 (S-Tree) *A spherical tree, or S-Tree in short, is a tree structure satisfying the following properties:*

- *It has a root, which is the whole unit sphere.*
- *Each node (except the root) is a spherical cap.*
- *Each node has exactly L children, except for the leaves.*
- *Each leaf contains no more than L vectors.*

Because we assume a static entity set, the S-tree is static too. Thus here we do not consider operations such as insertion/deletion. Rather, we focus on answering the orthogonal query using S-tree.

We build the S-tree by extending our spherical clustering algorithm. We adopt a top-down hierarchical K-means clustering algorithm to recursively partition the vectors at each level of the tree, and generate the spherical cap from each partition. The recursion is terminated when the S-tree node has no more than L vectors.

We now introduce our algorithm to find the most orthogonal vector using S-tree, as shown in Algorithm 3. It is a depth-first search with a bounding variable *best*, which stores the current global best answer. In termination cases, we pick the most orthogonal vector among the vectors in the leaf and the current *best* (line 3). Otherwise, we call SC-Pruning on the children spherical caps and obtain the set of possible candidate caps (line 4). We reverse the order of these caps by a stack (line 5-8). We then recurse on the child spherical caps from higher upper bound to lower upper bound, with respect to the query vector q (line 12). Each recursion updates the current *best* so subsequent sibling caps with lower upper bound are likely to be pruned (line 11).

6. EXPERIMENT

We have assembled the preference estimation and adaptive comparison selection into an **adaptive** system. Our system first builds an S-Tree from all the candidate comparisons and randomly picks an initial comparison. It then iteratively refines the preference estimation. In each iteration, the system asks for user feedback for the given comparison. It adjusts the sign of the comparison's normal vector according to the feedback, and estimates the preference using all answered comparisons as described in Section 3. It then uses the current estimation to query the S-Tree for the most orthogonal comparison, and use that comparison to repeat the iteration.

Algorithm 3 Orthogonal Search

```
 1: procedure ORTHOGONALSEARCH($node, q, best$)
 2:     if $node.isLeaf()$ then
 3:         return $\text{argmax}_{v \in node.V \cup \{best\}}\ orth(v, q)$
 4:     $\mathbf{Q} \leftarrow$ SC-PRUNING($q, node.children()$)
 5:     $\mathbf{S} \leftarrow \emptyset$                                ▷ Initialize stack
 6:     while !$\mathbf{Q}.isEmpty()$ do
 7:         $\mathbf{S}.push(\mathbf{Q}.top())$
 8:         $\mathbf{Q}.pop()$
 9:     while !$\mathbf{S}.isEmpty()$ do
10:         $t \leftarrow \mathbf{S}.top()$
11:         if UB$(t, q) > orth(best, q)$ then
12:             $best \leftarrow$ ORTHOGONALSEARCH($t, q, best$)
13:         $\mathbf{S}.pop()$
14:     return $best$
```

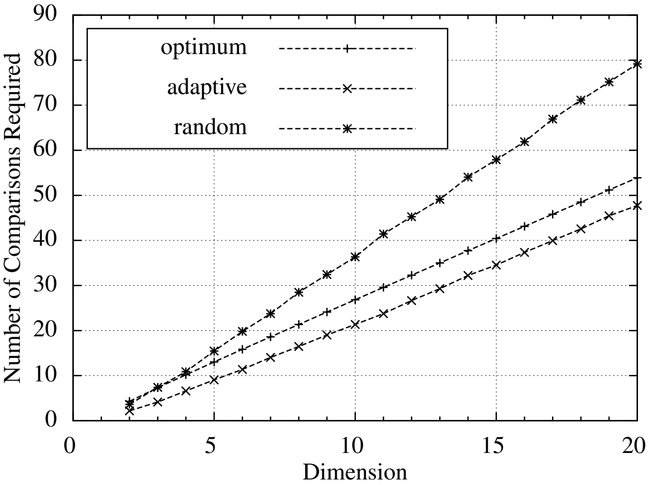


Figure 5: Number of Comparisons Required to Achieve a 0.95 Precision, for the Naïve System, the Optimum, and Our System

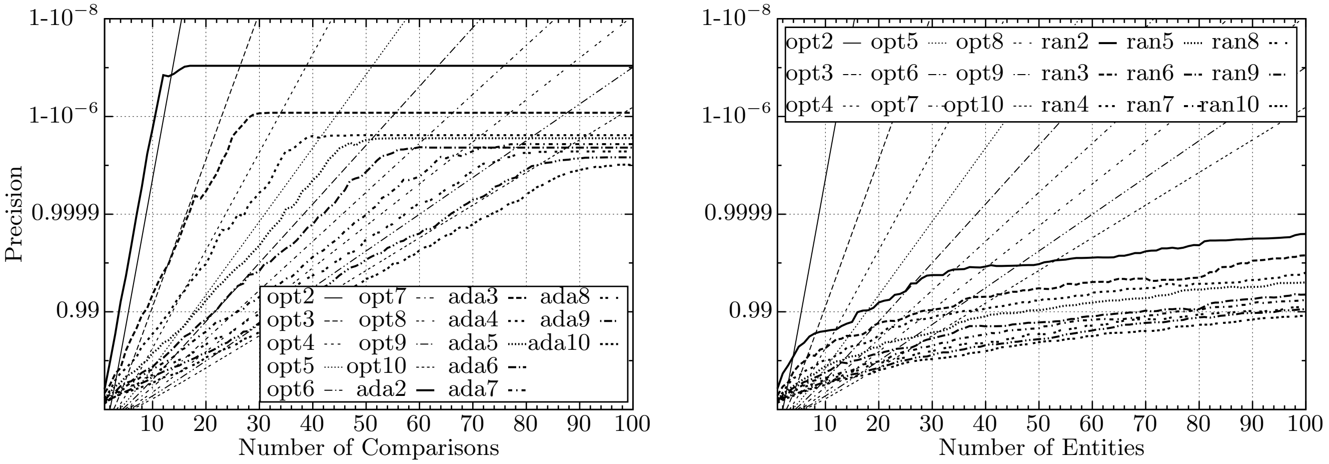
In this section we describe our evaluation of the adaptive system. The most important question is whether the correct preference function can be learned. We measure this correctness in terms of **precision**: the fraction of test (pairwise) comparisons correctly labeled by the system. We do so on both real and synthetic data. We also report results on additional experiments to evaluate the performance of the S-tree index structure.

As a baseline, we compare adaptive against a system that randomly chooses comparison pairs for training. This **random** system still performs cuts in space and correctly computes the preference function based on observed preferences (following the same estimation in Section 3). Thus, the superior results obtained by our algorithm can then all be attributed to its adaptive binary selection.

The algorithms were implemented in C++ referencing LIBSVM [9]. All experiments were run on a unix machine with a 2.7 GHz Intel i7 processor and 8GB 1600 MHz DDR3 memory. The user study was built with a web user interface referencing the system via CGI.

6.1 Experiment With Synthetic Data

The synthetic experiment is set up as follows. For each dimension, we randomly generated ten sets of one hundred entities. For each entity set, we randomly generated ten $\vec{w}_0$. And for each $\vec{w}_0$, we repeated the experiment ten times. We averaged the number of comparisons needed over all these variables. We randomly gener-



(a) Our Adaptive System (ada) v.s. the Theoretical Optimum (opt)

(b) The Random System (ran) v.s. the Theoretical Optimum (opt)

Figure 6: Precision Achieved by Various Systems w.r.t. the Number of Comparisons Performed on Synthetic Data with Demension 2 to 10

ated the source of truth $\vec{w}_0$, used Equation 1 to simulate the user feedback, estimated $\hat{\vec{w}}$ according to the discussion in Section 3, and calculated the precision as $\vec{w}_0 \cdot \hat{\vec{w}}$.

We first measured the average number of comparisons needed to achieve a target precision of 0.95. The result is shown in Figure 5.

According to the result, **adaptive** dramatically outperforms **random**, which randomly selects the next candidate comparison. On average, our system requires 40% less comparisons to achieve the same precision. This is because, the random strategy does not guarantee the quality of the next cut. As the remaining sphere shrinks, it is increasingly hard for a random cut to pass through the remaining sphere. Even for the cuts that do pass through, they are likely to be imbalanced. In consequence, the random system requires a much larger number of total comparisons.

We also plot the theoretical optimum, assuming that perfect binary cutting is possible, as described in Section 4.2, and results in a number of comparisons according to Equation 23 and a worst case precision defined in Equation 22. This is an analytically computed worst case number. Our adaptive system actually performs similar to and slightly better than this theoretical optimum. Note that this is not a contradiction. The theoretical optimum is defined with worst case precision, while in our system the precision is the observed average, which is better than the worst case.

To better understand the behavior beyond the target precision, we conducted a second experiment examining the precision change with respect the the number of comparisons performed. We kept the same experiment settings except we did not terminate the process when a target precision is reached. Instead, for each run, we performed one hundred comparisons, and recorded the average precision achieved after each comparison. We repeated the experiment for each dimension from 2 to 10. The result, as well as the theoretical optimum (derived from Equation 22) are shown in Figure 6a.

In general, the precision increases with the number of comparisons performed. It also decreases with dimension, because the higher the dimension is, the larger radius the remaining sphere has (see Equation 16). We also notice that, below 1-10^{-5}, the precision obtained by our **adaptive** system is very close to the **theoretical optimum**. Our adaptive outperforms the optimum for small number of comparisons because the approximation we used for the theoretical bound is only valid for larger number of comparisons.

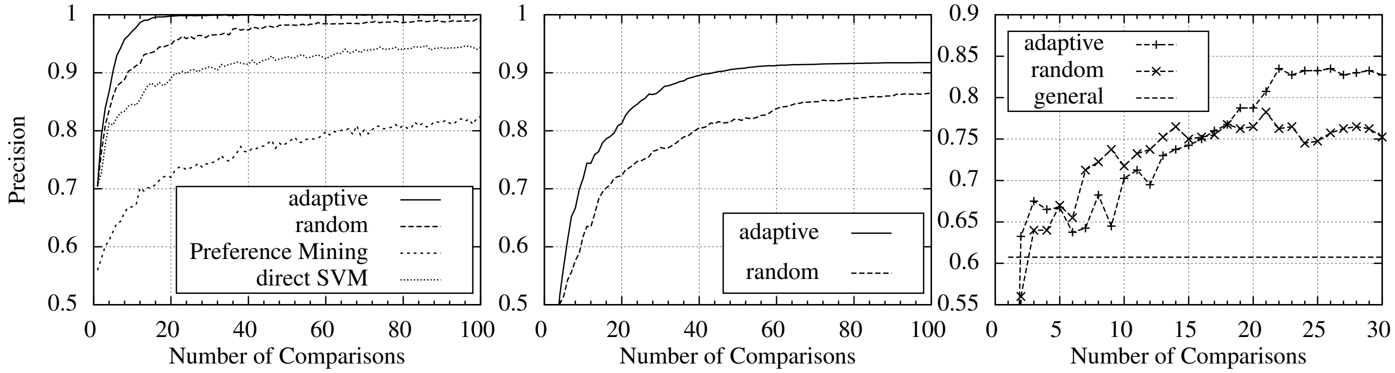
For comparison purposes, we performed the same experiment on the **random** system with random selection and the result is shown in Figure 6b. The result shows that the precision for the random system is significantly less than the optimum. By comparing this result with Figure 6a, we see the error rate for adaptive $(1 - p)$ is **three orders of magnitude** less than the random system.

In Figure 6a, we also notice that our precision reaches an upper bound after a certain number of comparisons. This is due to the limited number of entities in practice. In theory, we can cut the remaining sphere infinitely, thus achieving arbitrarily high precision. Unfortunately, in practice, once we arrive at a remaining sphere such that no candidate comparison further cuts it, we cannot improve our estimate anymore. And the average precision we can obtain on such a *finest remaining sphere* is determined by the number of entities.

Our last synthetic experiment compares our preference learning system with existing approaches. The first approach we compare with is Preference Mining [18], which detects strict partial order preferences from user log data. Since our system has no user log, we combine the pairs of entities in all performed pairwise comparisons and use them as input.The input data has a positive feedback if the corresponding entity is preferred in the pairwise comparison, and has a negative feedback otherwise. We also compare with a direct SVM application widely adopted in the information retrieval community. The SVM directly trains a preference using the same labeled entities set described above.

We generated 10 synthetic datasets. For each generation, we trained the preference and generated 100 test pairs to compare the precision in terms of percent of correctly estimated preference, for our adaptive approach, the random approach, Preference Mining and the direct SVM. The result is shown in Figure 7a.

The result shows that both our adaptive and random approaches significantly outperform the other two approaches. The poor performance of Preference Mining can attribute to its lack of quantitative internal model. Because it mines the preference from value frequency, it is not able to derive reliable preference when the input size is small. The direct SVM performs worse than our random approach, because the input can be highly inconsistent (E.g., even both entities in a given comparison are highly preferred, one of them has to receive a negative label). In contract, our adaptive and random approaches both train on the difference between entities, guaranteeing consistency according to our pairwise comparison formulation in Section 2.



(a) Precision Achieved by Our Adaptive System, Our Random System, Preference Mining and direct SVM on Synthetic Data

(b) Precision Achieved by Our Adaptive System and the Naïve Random System on Synthetic Data

(c) Precision Achieved by Our Adaptive System and the Random System in User Study

Figure 7: Precision Achieved by Various Systems w.r.t. the Number of Comparisons Performed on Yahoo Used Cars Dataset

6.2 Experiment with Real Data

To further demonstrate the improvement introduced by our adaptive comparison selection, we conducted a second experiment on real data. We used the Yahoo Used Car Dataset and randomly picked one thousand cars (one million comparisons). For each car, we represented it using nine dimensions including price, mileage, year etc., and normalized the data.

We measured the precision change with respect to the number of comparisons made, with the same settings as the previous experiment, expect that we replaced the synthetic data with the real used car dataset. The result is shown in Figure 7b.

Overall, we observe a drop in precision for both our **adaptive** system and the **random** system, compared to the previous experiment with synthetic data. This is because, the real data cannot guarantee uniform distribution, for several reasons. For instance, the distributions of some attributes are quite skewed by themselves (e.g., price and mileage). Furthermore, certain categorical attributes are split, causing the comparison vectors to have very few values on certain dimensions.

Despite the overall drop, our adaptive system still significantly outperforms the random system. After 40 comparisons, the adaptive precision is 10% higher. We also notice that the adaptive precision is capped after around 60 comparisons, while the random system is catching up. Intuitively, this cap is introduced by the uneven distribution of real data. The random system will eventually reach the cap but with a much slower convergence rate.

6.3 User Study

Since real users may not have linear preference and may make mistakes while providing feedback, we conducted a user study to further analyze our system in real scenarios. We recruited ten subjects and asked them to complete an online survey. The subjects were recruited from among university students and recent graduates in Singapore and United States. 6 were male, 4 were female, and they ranged in age from 18 to 30. The survey consisted of one hundred pairwise comparisons of used cars. The cars are from the same set of one thousand cars used in the previous experiment.

We separated the one hundred comparisons into three buckets: An adaptive training bucket with thirty adaptively selected comparisons, a random training bucket with thirty randomly selected comparisons, and a testing bucket with forty random comparisons. We randomized the order of these buckets to counterbalance the learning effect.

We respectively trained the user preference on the adaptive training bucket and random training bucket, and tested the preference on the testing bucket. The result is illustrated in Figure 7c.

According to the result, our adaptive system is slightly worse than the random system before 15 comparisons. This is because our adaptive algorithm is much more sensitive to user errors in early stages. After 20 comparisons, our adaptive system significantly outperforms the random system. Furthermore, after around 20 comparisons, the precision obtained with real user responses (Figure 7c) is almost as good as the precision with simulated responses (Figure 7b). For example, at 30 comparisons, the precision with real user feedback is 83% and 75% for **adaptive** and **random** respectively, while it is 87% and 77% with simulated (ideal) feedback. This demonstrates that the linear preference assumption is reasonable and our system is error-tolerant in practice.

Finally, we conducted an experiment to determine the importance of computing individual preferences. For this purpose, we trained a baseline *general* preference from the union of all training data from all users, and tested this preference function on each user's individual testing bucket. The resulting precision averaged over users is also displayed in Figure 7c. This precision is significantly lower than both the adaptive and the random precisions. This indicates that real users do exhibit very heterogenous preference in the used car scenario and there is not a universal preference that can satisfy all the users.

6.4 S-tree and Orthogonal Search Performance

In the previous section we show that our system is able to achieve very high precision (close to the practical upper bound determined by the total number of entities) with just a few user labeled comparisons. However, in real-life databases we have thousands of entities and millions of comparisons. In such cases, selecting the next best candidate comparison to ask the user can be expensive. On the other hand, since such selection is adaptive, it has to be done at interactive speed. In this section, we evaluate the performance of our adaptive selection and show that we are able to build the S-tree within reasonable time and answer orthogonal queries within seconds, for millions of comparisons. We do not evaluate the performance for preference estimation because the time for the estimation is negligible given the small number of labeled comparisons.

6.4.1 S-tree Construction

We first measured the time to construct the S-tree, by varying the entity dimension from 2 to 10, and the number of entities from 50 to 500. We set L to 50 and p to 0.95 for K-means. Every data point was averaged over ten runs.

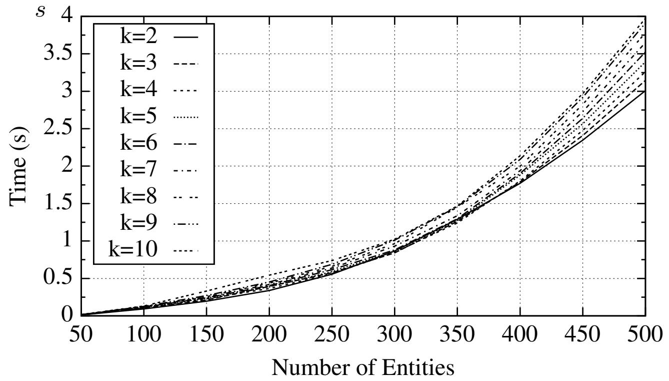


Figure 8: S-tree Construction Time with Dimension from 2 to 10, Number of Entities from 50 to 500, $L = 50$ and $p = 0.95$

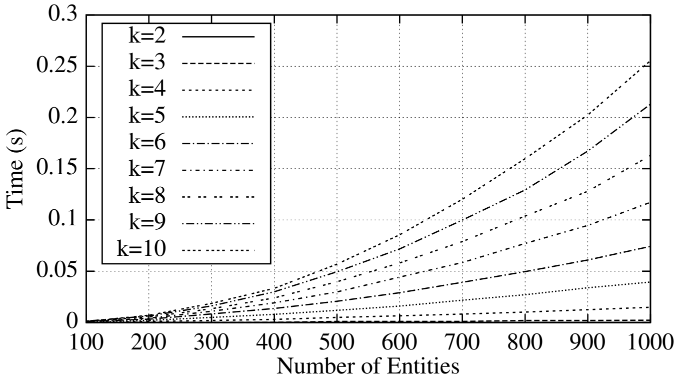


Figure 9: Orthogonal Query Time Using S-tree, with Dimension from 2 to 10, Number of Entities from 100 to 1000

The result in Figure 8 shows that the S-tree can be constructed within seconds for an entity set with reasonable size (note $N = 500$ indicates $125K$ difference vectors to index). In theory, given a number of M vectors, SVM is able to learn the center vector in $\mathcal{O}(M^3)$ time. In practice, we can achieve between $\mathcal{O}(M^2)$ and $\mathcal{O}(M^3)$. Since M squares N, the overall complexity is proportional to $\mathcal{O}(N^{4-6})$. However, this increase is not tremendous with $N < 500$.

In case of larger N where SVM is no longer applicable, we have two choices. First, we can sample the whole dataset to a relatively smaller subset, and learn the preference from the subset of entities. This will sacrifice some precision upper bound, but has no effect on the convergence rate or the number of comparisons needed to achieve a target precision below the upper bound. Alternatively, we can loosen the criteria for spherical clustering. For example, we can decrease p for the K-means clustering or even replace the SVM call in Algorithm 2 with a simple average function. This will results in larger spherical caps and more intersections between them, thus degrading the overall orthogonal query performance.

6.4.2 *Orthogonal Query*

We evaluated orthogonal query performance with the S-tree index. We varied the dimension from 2 to 10, and the number of entities N from 100 to 1000. For each combination, we randomly generated a query vector and repeated the query for ten times. The average query time is shown in Figure 9.

The result shows using S-tree we are able to answer orthogonal queries within one second for millions of comparisons. The query time presents no significant increase with respect to the number of entities at this scale. It also increases steadily for small dimensions.

7. RELATED WORK

Preferences have been extensively studied in the literature, including preference logics [34, 16], preference reasoning [33, 5] and logic programming [13, 29]. Recently, research on preferences has attracted broad interest in the database community [19, 11, 22]. This work can be classified into two categories, qualitative and quantitative. Qualitative preference is defined directly as binary preference relations [4, 11, 22, 23], while quantitative preference is defined with a numeric scoring function [1, 19]. While these works study the formulation of preferences in the context of relational database query, the preference in our paper is user-specific and query independent. As a result, we focus on deriving the hidden user preference rather than applying the preference in queries.

Since we adopt a user-interactive approach with simple input (pairwise comparison), an expressive preference formulation increases the risk for over fitting, according to Occam's razor principle. Therefore, we define our preference as a simple quantitative function, similar to previous quantitative preference work [1, 19].

Learning to order things [30] studies the ranking problem when transitivity is not guaranteed among comparison results. It takes as input the results of a set of comparisons between entities with no internal structure, and tries to find a ranking that minimizes the inconsistency among these comparison results. In contrast, our paper focuses on learning a general quantitative preference function from just a few input pairwise comparisons on structured entities.

User preferences have been recognized as important in ranking results in information retrieval. However, the database tradition has been to return all results the satisfy a specified predicate, unranked, leaving to the user the burden of tuning the predicate to get the desired results. Our work is aimed at decreasing this user burden.

There is a large body of literature on *learning to rank*, which employs machine learning techniques to construct ranking models[26, 25]. Recent learning to rank approaches using pairwise comparisons include Active Ranking [20, 36, 14], RankNet [6], IR SVM [8] and LambdaRank [7]. While learning to rank requires the query in the ranking model, our work is fundamentally different in that we require no query and learn the user preference from pairwise comparison feedbacks.

Active Ranking [20] adopts active learning [31] in ranking. It models the query as a common reference point in $\mathbb{R}^d$ and the ranking as the distance from the entities to that point. This is comparable to our work, where we define the preference to be the inner product between the entities and the user preference vector. However, active ranking neither guarantees the quality of the selected comparison nor illustrates how to select them. In contrast, our work provides specific algorithms to select the next comparison that is guaranteed to cut the remaining sphere as equally as possible.

Our work is also strongly related to traditional high dimensional database indexing, which includes a variety of multi-dimensional tree indices such as R-tree [15], R*-tree [3] and M-tree [12]. However, there are two essential differences. First, our data points (comparison vectors) are normalized and distributed on a unit sphere, as opposed to traditional geometric objects which are in $\mathbb{R}^d$. Second, our S-tree is specially designed for orthogonal query, which it not supported by any of the existing tree indices. As a result, indices such as R-tree cannot be directly applied to our case. M-tree offers a general indexing technique for objects whose relative distances form a general metric space. Unfortunately, our orthogonality does not satisfy the triangle inequality requested by the metric space.

The problem of searching vectors that is orthogonal to a given vector also has a lot of potential applications outside our scenario. For instance, in computer vision [28, 21] and signal processing [24, 2], algorithms were developed to find the residual vector of a given vector from a huge search space. S-tree proposed in this paper is potentially helpful to boost their search performance.

There is also a recent trend for crowdsourced ranking [10, 35, 27, 32]. However, most of them focus on learning a single ranking function across the population, while our work assumes different preferences among individuals.

8. CONCLUSIONS

Users often have complicated and individualized preference functions over multi-attribute data. Given a large collection of data, it is important for database systems to understand these preferences to be able to return results that are immediately useful. Preference understanding becomes particularly important as data sets grow in size: it is unlikely a user will want to wade through a thousand candidate used cars returned by a database.

To address this need, we have studied the problem of preference learning on structured entities from pairwise comparisons. Under a linear preference assumption, we formalized the problem and decomposed it into two subproblems, preference estimation and adaptive comparison selection.

For preference estimation, we derived the mathematical foundation of the problem and solved it using an innovative application of SVM. For adaptive comparison selection, we theoretically analyzed a random selection approach and an optimal binary selection strategy, and proposed our adaptive selection approach. We abstracted the adaptive selection into an innovative orthogonal query, and proposed a new type of index S-tree to answer it.

We described our algorithms for various parts and integrated them into a preference learning system. We further implemented the system and evaluated its effectiveness and performance. Our experiment with synthetic data showed that our system is able to achieve a high precision with just a few comparisons, coming close to the theoretical binary selection optimum. Both our experiment and user study with real data demonstrated that our system significantly outperforms the random selection system. We also showed that our preference learning approach is much more effective than existing approaches, and our S-tree index can be built efficiently and serve orthogonal query in interactive speed with reasonable scale. Our experiments also showed that our linear preference assumption is reasonable in practice, and our method is able to tolerate natural user inconsistencies in pairwise preference reporting.

In the current approach, the user still needs to performs dozens of comparisons in order to achieve a practical precision. One possible direction to further boost the convergence rate it to analyze the distribution of the data set and refine our adaptive selection accordingly. Also, our current work solves the problem for each individual separately. Given the potential similarities between individuals, there indeed are opportunities for cross-learning, which can further decrease the number of questions required. Finally, combining our approach with offline log data learning would potentially boost the starting precision. We will pursue these directions in future work.

9. REFERENCES

[1] R. Agrawal and E. L. Wimmers. A framework for expressing and combining preferences. *SIGMOD*, 29(2):297–306, 2000.

[2] C. F. Barnes, S. A. Rizvi, and N. M. Nasrabadi. Advances in residual vector quantization: a review. *TIP*, 1996.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. *The R*-tree: an efficient and robust access method for points and rectangles*. ACM, 1990.

[4] S. Borzsony, D. Kossmann, and K. Stocker. The skyline operator. In *Data Engineering, 2001. Proceedings. 17th International Conference on*, pages 421–430. IEEE, 2001.

[5] C. Boutilier, R. I. Brafman, H. H. Hoos, and D. Poole. Reasoning with conditional ceteris paribus preference statements. In *Uncertainty in artificial intelligence*, pages 71–80, 1999.

[6] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. ICML, 2005.

[7] C. J. Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 2010.

[8] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon. Adapting ranking svm to document retrieval. In *SIGIR*, 2006.

[9] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM TIST*, 2011.

[10] X. Chen, P. N. Bennett, and K. e. a. Collins-Thompson. Pairwise ranking aggregation in a crowdsourced setting. In *WSDM*, 2013.

[11] J. Chomicki. Querying with intrinsic preferences. In *EDBT*, pages 34–51. 2002.

[12] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, 1997.

[13] J. P. Delgrande, T. Schaub, and H. Tompits. Logic programs with compiled preferences. *arXiv preprint cs/0003028*, 2000.

[14] A. Fujii, T. Tokunaga, K. Inui, and H. Tanaka. Selective sampling for example-based word sense disambiguation. *Computational Linguistics*, 1998.

[15] A. Guttman. *R-trees: a dynamic index structure for spatial searching*. ACM, 1984.

[16] S. O. Hansson. Preference logic. In *Handbook of philosophical logic*, pages 319–393. 2002.

[17] C. Ho and S. Zimmerman. On the number of regions in an m-dimensional space cut by n hyperplanes. *Gazette of Australian Math Society*, 2006.

[18] S. Holland, M. Ester, and W. Kießling. Preference mining: A novel approach on mining user preferences for personalized applications. In *PKDD*, pages 204–216. 2003.

[19] V. Hristidis, N. Koudas, and Y. Papakonstantinou. Prefer: A system for the efficient execution of multi-parametric ranked queries. In *SIGMOD*, volume 30, pages 259–270, 2001.

[20] K. G. Jamieson and R. Nowak. Active ranking using pairwise comparisons. In *NIPS*, 2011.

[21] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *TPAMI*, 2011.

[22] W. Kießling. Foundations of preferences in database systems. In *PVLDB*, pages 311–322, 2002.

[23] W. Kießling and G. Köstler. Preference sql: design, implementation, experiences. In *PVLDB*, pages 990–1001, 2002.

[24] F. Kossentini, M. J. Smith, and C. F. Barnes. Entropy constrained residual vector quantization. IEEE, 1993.

[25] H. Li. Learning to rank for information retrieval and natural language processing. *Human Language Technologies*, 2011.

[26] H. Li. A short introduction to learning to rank. *IEICE*, 2011.

[27] S. Negahban, S. Oh, and D. Shah. Iterative ranking from pair-wise comparisons. In *NIPS*, 2012.

[28] S. A. Rizvi and N. M. Nasrabadi. Predictive residual vector quantization [image coding]. *TIP*, 1995.

[29] C. Sakama and K. Inoue. Prioritized logic programming and its application to commonsense reasoning. *Artificial Intelligence*, 123(1):185–222, 2000.

[30] W. W. C. R. E. Schapire and Y. Singer. Learning to order things. In *Advances in Neural Information Processing Systems 10: Proceedings of the 1997 Conference*, volume 10, page 451. MIT Press, 1998.

[31] B. Settles. Active learning literature survey. *University of Wisconsin, Madison*, 2010.

[32] O. Tamuz, C. Liu, S. Belongie, O. Shamir, and A. T. Kalai. Adaptively learning the crowd kernel. *arXiv:1105.1033*, 2011.

[33] S.-W. Tan and J. Pearl. Specification and evaluation of preferences for planning under uncertainty. *Proc. of KR*, 94, 1994.

[34] G. H. Von Wright. The logic of preference. 1965.

[35] J. Yi, R. Jin, S. Jain, and A. Jain. Inferring users preferences from crowdsourced pairwise comparisons: A matrix completion approach. In *HCOMP*, 2013.

[36] H. Yu. Selective sampling techniques for feedback-based data retrieval. *DMKD*, 2011.