Efficient Online Relative Comparison Kernel Learning

Eric Heim * Matthew Berger † Lee M. Seversky † Milos Hauskrecht *

Abstract

Learning a kernel matrix from relative comparison human feedback is an important problem with applications in collaborative filtering, object retrieval, and search. For learning a kernel over a large number of objects, existing methods face significant scalability issues inhibiting the application of these methods to settings where a kernel is learned in an online and timely fashion. In this paper we propose a novel framework called **E**fficient online **R**elative comparison **K**ernel **LE**arning (ERKLE), for efficiently learning the similarity of a large set of objects in an online manner. We learn a kernel from relative comparisons via stochastic gradient descent, one query response at a time, by taking advantage of the sparse and low-rank properties of the gradient to efficiently restrict the kernel to lie in the space of positive semidefinite matrices. In addition, we derive a passive-aggressive online update for minimally satisfying new relative comparisons as to not disrupt the influence of previously obtained comparisons. Experimentally, we demonstrate a considerable improvement in speed while obtaining improved or comparable accuracy compared to current methods in the online learning setting.

Keywords

Online Learning, Kernel Learning, Relative Comparisons.

1 Introduction

Learning a similarity model over a set of objects from human feedback is important to many applications in collaborative filtering, document and multimedia retrieval, and visualization. It has been shown that by incorporating human feedback, the overall performance of such applications can be greatly improved [11, 13, 15, 31]. In this work we focus on learning a similarity model from human feedback through relative comparisons. More specifically, we focus on the *relative comparison kernel learning* (RCKL) problem, in which the goal is to learn a positive semidefinite (PSD) kernel matrix from relative comparisons given by humans. Kernels are used for modeling object relationships in many learning techniques [23], and hence are applicable to many methods that utilize kernels for these applications.

*University of Pittsburgh, Department. of Computer Science
{eric, milos}@cs.pitt.edu

†Air Force Research Laboratory, Information Directorate
{matthew.berger.1, lee.seversky}@us.af.mil

In learning a kernel from human supervision, it is important to obtain feedback which is intuitive for the user to provide and informative for a learning algorithm to use. For instance, naive forms of supervision such as numerical judgments between pairs of objects have been shown to be very noisy [24]. A *relative comparison*, the response to a query of the form "*Is object A more similar to object B or C?*", is well known as an intuitive mechanism for soliciting human feedback and an effective way of learning similarity [12]. Recent works addressing fine-grained categorization [28] and perceptual visualization design [6] have shown the practicality and benefit of learning kernels from relative comparisons.

Many RCKL methods [1, 27] learn a kernel by solving a semidefinite program (SDP) in batch – from all available comparisons. However, in numerous practical applications, a batch approach is not appropriate due to the online and dynamic nature of the application. For example, in crowdsourcing it is often of interest to minimize the number of dispatched tasks, and thus the cost of the crowd, by leveraging active learning techniques [25, 10] to adaptively select the most informative relative comparison query. The success of these techniques depends on maintaining an up to date model so as to ensure the most informative query is selected, as well as an efficient learning method to quickly update the model so that no crowd participant is idle. Likewise, recommendation systems for online marketplaces obtain continuous feedback in the form of click-through data via user interaction. In order for the learned kernel to be up to date and reflect the latest user feedback, the learning method must be able to quickly incorporate feedback as it is received.

These scenarios motivate the need for an *efficient* and *online* method for learning from large-scale relative comparison data. Batch methods poorly scale for large object collections primarily because they must ensure their solutions are PSD. Without any prior assumptions on the data this operation is of $O(n^3)$ time complexity for n objects, which for large n is prohibitively slow for the aforementioned applications.

This work introduces a novel online RCKL framework called **E**fficient online **R**elative comparison **K**ernel **LE**arning (ERKLE) that achieves efficiency through the unique structure of RCKL gradients. First, ERKLE sequentially updates a kernel one query response at a time in $O(n^2)$ complexity by employing a stochastic gradient descent technique that takes advantage of the sparse and low-rank structure of the RCKL gradient over a single comparison for efficient PSD

projections. We show that the gradient structure not only enables an efficient update that requires finding only the smallest eigenvalue and eigenvector, but generalizes several well-known convex RCKL methods [1, 27]. Second, the structure of the gradient also reveals a simple way to bound the smallest eigenvalue after each gradient step, allowing certain updates to be performed in constant time. Third, motivated by previous work in online learning [5], we also derive a passive-aggressive version of ERKLE to ensure learned kernels model the most recently obtained relative comparisons without over-fitting. The passive-aggressive scheme in conjunction with the smallest eigenvalue bound allows us to skip many PSD projections, yielding a very efficient yet effective kernel learning method.

Experimentally, we show that ERKLE is able to efficiently produce high-quality kernels in an online setting under different scenarios: learning from a small number of relative comparisons, noisy comparisons, as well as low dimensional data distributions. We show that on synthetic and real-world datasets we obtain both improved performance and faster run times compared to batch RCKL methods. In summary, we believe our method now makes it practical to learn kernels from human-provided comparisons over large-scale datasets.

2 Related Work

The problem of learning a kernel matrix, driven by relative comparison feedback, has been the focus of much recent work. Most recent techniques primarily differ by the choice of loss function. For instance, Generalized Non-metric Multidimensional Scaling [1] employs hinge loss, Crowd Kernel Learning [25] uses a probabilistic, scale-invariant loss, and Stochastic Triplet Embedding [27] uses a logistic loss.

The aforementioned RCKL methods can be viewed as solving a kernelized special case of the classic non-metric multidimensional scaling problem [14], where the goal is to find an embedding of objects in $\mathbb{R}^d$ such that they satisfy given Euclidean distance constraints. In contrast to many of the kernel-learning formulations, their analogous embedding-learning counterparts are non-convex optimization problems, which only guarantee convergence to a local minimum. In the typical non-convex batch setting, multiple solutions are found with different initializations and the best is chosen among them. This strategy is poorly suited for the online setting where triplets are being observed sequentially, and which solution is best may change as feedback is received.

In this work we consider the online RCKL problem, where one is sequentially acquiring relative comparisons among a large collection of objects. Stochastic gradient descent techniques [21] are a popular class of methods for a very general class of functions [3] that can be applied to online learning. Recent stochastic, online techniques [30, 22] have demonstrated competitive performance with batch techniques. In addition, efficient methods have been developed to solve SDPs in an online fashion [8, 16]. The work of [4] shows how to devise efficient update schemes for solving SDPs when the gradient of the objective function is low-rank. We build upon and improve the efficiency of this work, by taking advantage of the sparse and low-rank structure of the gradient common in convex RCKL formulations.

Our passive-aggressive step size procedure is informed by [5] for other learning problems. In their work, the authors create a passive-aggressive online update rule for classic SVM formulations used in problems such as binary/multi-class classification and regression. In deriving such an update for different RCKL loss functions, we relate how different methods can be utilized under a common passive-aggressive framework. To our knowledge, such an update for RCKL problems and the associated analysis has not been done.

3 Preliminaries

In this section, we formally define RCKL and provide a brief overview of RCKL methods. Let S_+^n be the set of $n \times n$ PSD matrices, and $\mathbf{M}^{ab}$ be the entry at row a column b of a matrix $\mathbf{M}$. The goal of RCKL is to learn a PSD kernel matrix $\mathbf{K} \in S_+^n$ over n objects, given a set $\mathcal{T}$ of triplets:

$$\mathcal{T} = \{(a, b, c) \mid a \text{ is more similar to } b \text{ than } c\} \tag{3.1}$$

such that squared distance constraints are satisfied:

$$\begin{aligned} \forall_{(a,b,c)\in\mathcal{T}}: \quad & d^2_{\mathbf{K}}(a,b) < d^2_{\mathbf{K}}(a,c) \\ \text{where} \quad & d^2_{\mathbf{K}}(a,b) = \mathbf{K}^{aa} + \mathbf{K}^{bb} - 2\mathbf{K}^{ab}. \end{aligned} \tag{3.2}$$

We say a kernel $\mathbf{K}$ satisfies a triplet $t_i = (a_i, b_i, c_i) \in \mathcal{T}$ if the constraint in (3.2) corresponding to t_i is satisfied.

In this work, we consider triplets that are answers to relative comparison queries posed to one or more people. We define a query q to have three components, a "head" object h to be compared with two objects o^1 and o^2. A query $q = \left(h, \left\{o^1, o^2\right\}\right)$ can be answered by either the triplet $\left(h, o^1, o^2\right)$ or $\left(h, o^2, o^1\right)$. In order to model a complete notion of the desired human similarity space, a learned kernel should not only satisfy triplets that were obtained (observed), but also be general enough to satisfy unobtained (unobserved) triplets that reflect true relationships among objects.

3.1 RCKL Formulation Many RCKL methods can be generalized by the following SDP:

$$\begin{aligned} \min_{\mathbf{K}} \quad & L\left(\mathbf{K}, \mathcal{T}\right) + \tau \mathrm{Trace}(\mathbf{K}) \\ \text{s.t.} \quad & \mathbf{K} \succeq 0. \end{aligned} \tag{3.3}$$

The objective function is composed of two terms. The first term is a function L measuring how much loss $\mathbf{K}$ incurs for not satisfying triplets in $\mathcal{T}$. The second term is a trace regularization on $\mathbf{K}$. Trace regularization is used as a convex approximation of the non-convex rank function. Higher

values of τ enforce that (3.3) produces lower-complexity similarity models. Finally, $\mathbf{K}$ is constrained to be PSD.

The loss function in the objective can be decomposed into the sum of losses over individual triplets:

$$L(\mathbf{K}, \mathcal{T}) = \sum_{t \in \mathcal{T}} l(\mathbf{K}, t). \tag{3.4}$$

Existing RCKL methods differ in the choice of the loss function l. The Stochastic Triplet Embedding (STE) approach of [27] defines $l(\mathbf{K}, t) = -\log p_t^{\mathbf{K}}$ as the loss function, where $p_t^{\mathbf{K}}$ is the probability that a triplet is satisfied:

$$p_{t=(a,b,c)}^{\mathbf{K}} = \frac{\exp(-d_{\mathbf{K}}^2(a,b))}{\exp(-d_{\mathbf{K}}^2(a,b)) + \exp(-d_{\mathbf{K}}^2(a,c))}. \tag{3.5}$$

Generalized Nonmetric Multidimensional Scaling (GNMDS) [1] uses a hinge loss, where $l(\mathbf{K}, t = (a,b,c))$ is defined as:

$$\max(0, d_{\mathbf{K}}^2(a,b) - d_{\mathbf{K}}^2(a,c) + 1). \tag{3.6}$$

For either loss function l, (3.3) is a convex optimization problem and the globally optimal solution is found by performing projected gradient descent, which consists of two update steps. The first step is a simple descending step along the gradient of the objective:

$$\mathbf{K}_i' = \mathbf{K}_{i-1} - \delta_i \left(\nabla L(\mathbf{K}_{i-1}, \mathcal{T}) + \tau \mathbf{I}\right), \tag{3.7}$$

where i denotes the current iteration, δ_i is the learning rate. The second step projects the result of the first gradient step onto the PSD cone:

$$\mathbf{K}_i = \Pi_{S_+}(\mathbf{K}_i'). \tag{3.8}$$

These steps are iterated until convergence.

4 Efficient Online Relative Comparison Kernel Learning (ERKLE)

The main computational bottleneck of traditional RCKL methods is the projection onto the PSD cone, Π_{S_+}. This projection is commonly found by first taking the eigendecomposition of $\mathbf{K}_i' = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$ and setting all negative eigenvalues to 0, i.e. $\mathbf{K}_i = \mathbf{V}[\mathbf{\Lambda}]_+\mathbf{V}^t$, where $[\cdot]_+$ is defined on diagonal entries of a matrix as $[\mathbf{\Lambda}^{ii}]_+ = \max(0, \mathbf{\Lambda}^{ii})$. Absent of any prior knowledge on the structure of $\mathbf{K}_i'$, its full eigendecomposition is necessary for the projection. Since this is an $O(n^3)$ operation, the projection step renders batch methods computationally prohibitive for learning the similarity of a large number of objects in an online manner.

4.1 Stochastic Gradient Step To create an efficient and online framework for RCKL – ERKLE – we first leverage the form of common RCKL loss functions to produce a stochastic update with respect to a single triplet. As shown in (3.4), the loss function L naturally decomposes into the sum over losses l defined on individual observations (triplets in our case). From this decomposition, ERKLE first performs the following stochastic gradient step:

$$\mathbf{K}_j' \leftarrow \mathbf{K}_{j-1} - \delta_j \nabla l(\mathbf{K}_{j-1}, t_j), \tag{4.9}$$

where triplets $t_1, ..., t_{j-1}$ have been observed, $\mathbf{K}_{j-1}$ is the online solution after observing the $j-1$ triplet,

Performing a stochastic optimization gives ERKLE an advantage over current RCKL methods that perform batch optimizations. Batch methods attempt to minimize a loss function over a training set. Doing so minimizes empirical risk with respect to particular training samples. Classically, this is used to estimate the expected risk of the ground truth distribution over all samples. Obtaining triplets in an online fashion from a source can be viewed as directly sampling triplets from the ground truth distribution at random. As such, taking stochastic steps over samples directly minimizes expected risk with respect to the ground truth distribution of triplets. Because of this characteristic, stochastic methods tend to generalize better to unobserved samples. See [3] for more details on stochastic methods and their properties.

Note that our online formulation does not include trace regularization. Although this may impact our method in generalizing to unseen triplets, our online formulation achieves good generalization through carefully constructed, data-dependent step sizes δ_j, as detailed in Section 4.3.

4.2 Efficient Projection In order to retain positive semi-definiteness, after taking a stochastic gradient step the resulting matrix $\mathbf{K}_j'$ must be projected onto the PSD cone. Following the procedure of Π_{S_+} is prohibitively expensive for our online setting. Instead, for RCKL methods we can take advantage of the sparse and low-rank nature of the gradient to devise an efficient projection scheme. To this end, we introduce a *canonical gradient matrix* $\mathbf{G}$ over a triplet $t = (a,b,c))$, where the entries are defined as:

$$\mathbf{G}^{ij} = \begin{cases} -2 & \text{if } i=a, j=b \text{ or } i=b, j=a \\ 2 & \text{if } i=a, j=c \text{ or } i=c, j=a \\ 1 & \text{if } i=b, j=b \\ -1 & \text{if } i=c, j=c \\ 0 & \text{otherwise.} \end{cases} \tag{4.10}$$

Now consider the following choice for the stochastic step:

$$\nabla l(\mathbf{K}, t) = f(\mathbf{K}, t)\,\mathbf{G}, \tag{4.11}$$

where f is a real-valued function. With (4.11) as the gradient in (4.9), $\mathbf{K}_{j-1}$ is updated by increasing entries corresponding to the similarity between objects a and b and decreasing the similarity between a and c by a factor of $f(\mathbf{K}_{j-1}, t_j)$.

The function f can be defined such that we recover the gradients of l for different convex RCKL formulations. The stochastic gradient for STE can be obtained by defining f as:

$$f(\mathbf{K}, t) = 1 - p_t^{\mathbf{K}} \tag{4.12}$$

Similarly, by defining f to be:

$$f(\mathbf{K}, t) = \begin{cases} 1 & \text{if } d^2_{\mathbf{K}}(a,b) + 1 < d^2_{\mathbf{K}}(a,c) \\ 0 & \text{otherwise} \end{cases} \tag{4.13}$$

the stochastic gradient for GNMDS is obtained. Note, that this not only generalizes these two methods for use in our online framework but also suggests a simple way to create new online RCKL methods by designing a function f that weighs the contribution of individual triplets.

Decomposing the online updates in such a way reveals a key insight into how to perform efficient projections onto the PSD cone after the stochastic step. Algorithm 1 outlines the procedure for efficient projection in ERKLE. Here, $\lambda_\downarrow$ and $\mathbf{v}_\downarrow$ are the smallest eigenvalue and eigenvector of matrix $\mathbf{K}$. This procedure has a time complexity $O(n^2)$ due to finding $\lambda_\downarrow$ and $\mathbf{v}_\downarrow$. To show that Algorithm 1 does indeed perform the correct projection, we prove the following theorem:

THEOREM 4.1. *Algorithm 1 results in a PSD matrix $\mathbf{K}_j$ that is closest to $\mathbf{K}'_j$ in terms of Frobenius distance.*

Proof. Let $\mathbf{K}_0 \in S^n_+$ (i.e. identity). We use this as our base case and show inductively that after each iteration of the main loop, $\mathbf{K}_j$ remains PSD. Let $\gamma_j = \delta_j f(\mathbf{K}_{j-1}, t_j)$ be the *magnitude* of an update. By (4.11), the update in Equation (4.9) can be written as $\mathbf{K}_{j-1} - \gamma_j \mathbf{G}$. The only nonzero eigenvalues of $-\gamma_j \mathbf{G}$ are $\lambda_1 = 3\gamma_j$ and $\lambda_2 = -3\gamma_j$. It follows from Weyl's inequality that the matrix $\mathbf{K}'_j = \mathbf{K}_{j-1} - \gamma_j \mathbf{G}$ has *at most* one negative eigenvalue. If $\mathbf{K}'_j$ has no negative eigenvalues, then it is PSD (line 6 of Algorithm (1)). If $\mathbf{K}'_j$ has one negative eigenvalue, line 4 of Algorithm 1 results in a PSD matrix $\mathbf{K}_j$ that is closest to $\mathbf{K}'_j$ in terms of Frobenius distance by Case 2 of Theorem 4 in [4].

The important implication of Thm. 4.1 is that ERKLE can incorporate a triplet into a kernel in $O(n^2)$ time by performing the efficient projection outlined in Algorithm 1. Furthermore, if a step is sufficiently small, then no projection is needed at all. Let λ^0_j be the smallest eigenvalue of $\mathbf{K}_j$. By Weyl's inequality, if $\lambda^0_j - 3\gamma_j \geq 0$, then all eigenvalues of $\mathbf{K}'_{j+1}$ are greater than or equal to 0. This can be used to skip the projection step when the update is known to result in a PSD matrix. In our algorithm, we lower bound the smallest eigenvalue by maintaining a conservative estimate $\hat{\lambda}^0_j$. Initially, $\hat{\lambda}^0_0 \leftarrow \lambda^0_0$. It is updated each iteration with it's lower bound $\hat{\lambda}^0_j \leftarrow \hat{\lambda}^0_{j-1} - 3\gamma_j$. If $\hat{\lambda}^0_j < 0$, then Alg. 1 is used to project onto the PSD cone and $\hat{\lambda}^0_j \leftarrow \max(0, \lambda_\downarrow)$. Otherwise, no projection is performed. In the case where $\lambda^0_0 >> -3\gamma_j$, this simple lower-bounding procedure can save many eigenvalue/eigenvector computations.

Algorithm 1 Efficient PSD Projection

1: **procedure** $\Pi^1_+(\mathbf{K})$
2: Find $\lambda_\downarrow$ and $\mathbf{v}_\downarrow$ from $\mathbf{K}$
3: **if** $\lambda_\downarrow < 0$ **then**
4: **return** $\mathbf{K} - \lambda_\downarrow \mathbf{v}_\downarrow \mathbf{v}_\downarrow^T$
5: **else**
6: **return** $\mathbf{K}$
7: **end if**
8: **end procedure**

4.3 Passive-Aggressive Updates A key difference between the batch and stochastic RCKL updates is the magnitude of the updates. For both methods the magnitude of the updates with respect to a single triplet t is a function of a learning rate and how well the previous solution satisfies t. In the previous section we denoted the magnitude of an ERKLE update as γ_j. In the batch setting, the same learning rate δ_i is used for all triplets in a given step. In contrast, traditional stochastic methods use different learning rates δ_j over data samples to accelerate convergence, where the δ_j are designed to satisfy certain conditions. Early work [3] on the topic of learning rates suggest that δ_j should satisfy two constraints: $\sum_{j=1}^{\infty} \delta_j^2 < \infty$ and $\sum_{j=1}^{\infty} \delta_j = \infty$. For example $\delta_j = 1/j$ satisfies these constraints. Later work [18] suggests a more aggressive setting of $\delta_j = 1/\sqrt{j}$.

In our setting, however, we prefer to treat triplets equally: current triplets should not have more influence than preceding triplets. On the other hand, we do not wish to over-fit to the most recently obtained triplets. It is this observation that motivates Passive-Aggressive (PA) Online Learning [5]. In the RCKL setting, the general idea is that if the previous solution $\mathbf{K}_{j-1}$ satisfies a newly obtained triplet $t_j = (a, b, c)$ by a margin of 1, then do not update the kernel (passive). Otherwise, update the kernel so that the kernel is changed the minimal amount, but t_j is satisfied by a margin of 1 (aggressive). A fortunate side effect of choosing minimally sized updates is that updates are less likely to result in non-PSD matrices than larger steps, thus further reducing the number of projections onto the PSD cone via our conservative eigenvalue estimate (Section 4.2).

To derive a passive-aggressive update for ERKLE, we wish to learn a magnitude of a stochastic step $\gamma_j = \delta_j f(\mathbf{K}_{j-1}, t_j)$ with passive-aggressive properties. f as defined by GNMDS in (4.13) is inherently passive, but if $\mathbf{K}_{j-1}$ does not satisfy the margin constraint, it takes a step independent of how close the previous solution is to satisfying t_j. As such, we wish to find a δ_j that takes an aggressive step. We do this by solving the following optimization problem:

$$\begin{aligned} \min_{\delta_j} \quad & \delta_j^2 \\ \text{s.t.} \quad & d^2_{\mathbf{K}'_j}(a,b) + 1 \leq d^2_{\mathbf{K}'_j}(a,c), \delta_j \geq 0 \end{aligned} \tag{4.14}$$

By (4.11) and (4.13), the first constraint can be rewritten as:

$$d^2_{\mathbf{K}_{j-1}}(a,b) - d^2_{\mathbf{K}_{j-1}}(a,c) - 10\delta_j + 1 \leq 0 \tag{4.15}$$

With the assumption that the triplet is not satisfied by a margin of one in $\mathbf{K}_{j-1}$, no update is required; otherwise, only a positive value of δ_j can satisfy (4.15), making the positive constraint on δ_j redundant. Also, the smallest δ_j that satisfies (4.15) is the one that makes the left hand side exactly zero. As a result, the inequality constraint can be handled as equality. To find the optimum we first write the Lagrangian $\mathcal{L}(\delta_j, \alpha)$:

$$\delta_j^2 + \alpha\left(d^2_{\mathbf{K}_{j-1}}(a,b) - d^2_{\mathbf{K}_{j-1}}(a,c) - 10\delta_j + 1\right) \tag{4.16}$$

Taking the partial derivative of (4.16) with respect to δ_j, setting it to 0, and solving for δ_j results in $\delta_j = 5\alpha$. Substituting this back into (4.16) makes the Lagrangian:

$$-25\alpha^2 + \alpha\left(d^2_{\mathbf{K}_{j-1}}(a,b) - d^2_{\mathbf{K}_{j-1}}(a,c) + 1\right) \tag{4.17}$$

Taking the partial derivative of (4.17) with respect to α, setting it to 0, solving for α and then substituting this back into $\delta_j = 5\alpha$ results in the minimum step size that satisfies the margin constraint:

$$\delta_j = \frac{d^2_{\mathbf{K}_{j-1}}(a,b) - d^2_{\mathbf{K}_{j-1}}(a,c) + 1}{10} \tag{4.18}$$

A similar passive-aggressive update can be derived using the probability of a triplet being satisfied in STE. Consider the following optimization:

$$\begin{aligned} \min_{\delta_j} \quad & \delta_j^2 \\ \text{s.t.} \quad & p^{\mathbf{K}'_j}_{t_j} \geq P, \delta_j \geq 0 \end{aligned} \tag{4.19}$$

In (4.19) the minimal step size is chosen such that the probability that a triplet is satisfied after the update is greater than or equal to a given probability $P \in (0.5, 1)$. Using (4.19), we derive the following step size:

$$\delta_j = \frac{d^2_{\mathbf{K}_{j-1}}(a,b) - d^2_{\mathbf{K}_{j-1}}(a,c) + \kappa}{10} \tag{4.20}$$

where $\kappa = \log(P) - \log(1-P)$. The full derivation is given in the extended version of this work [9]. Both derivations reveal that passive-aggressive updates using STE and GNMDS are similar. Setting $P = \frac{e}{1+e}$ in (4.20) recovers the GNMDS passive-aggressive step in (4.18), and changing the margin in (4.18) recovers different settings of P.

Note that using (4.18) as a step size results in a $\mathbf{K}'_j$ with the intended passive-aggressive property, not necessarily $\mathbf{K}_j$ after the projection. We choose to find a passive-aggressive step size instead of a full update for computational efficiency. Finding a true passive-aggressive step size with respect to $\mathbf{K}_j$ would require iteratively projecting onto the PSD cone, which is computationally prohibitive in the online setting. In practice, $d^2_{\mathbf{K}'_j}$ is a good approximation to $d^2_{\mathbf{K}_j}$, as their difference is dependent on the magnitude of the (potentially) negative eigenvalue of $\mathbf{K}'_j$, which tends to be small.

Even for a proper setting of δ_j, it has been shown that stochastic methods perform best when multiple rounds of updates or passes are performed on the observed samples [2, 20, 29]. For our problem setting, this indicates that ERKLE may benefit from revisiting triplets that were previously used to update the kernel. In our experiments we perform a simple multi-pass scheme where for each new triplet, ERKLE not only steps over the most recently obtained triplet, but also a number of randomly sampled triplets from the set of previously obtained triplets. We denote the number of "passes" ERKLE performs each time a new triplet is observed as β. Algorithm 1 of [9] describes this process in more detail. Using this simple approach is sufficient for ERKLE to maintain high accuracy while still ensuring computational efficiency.

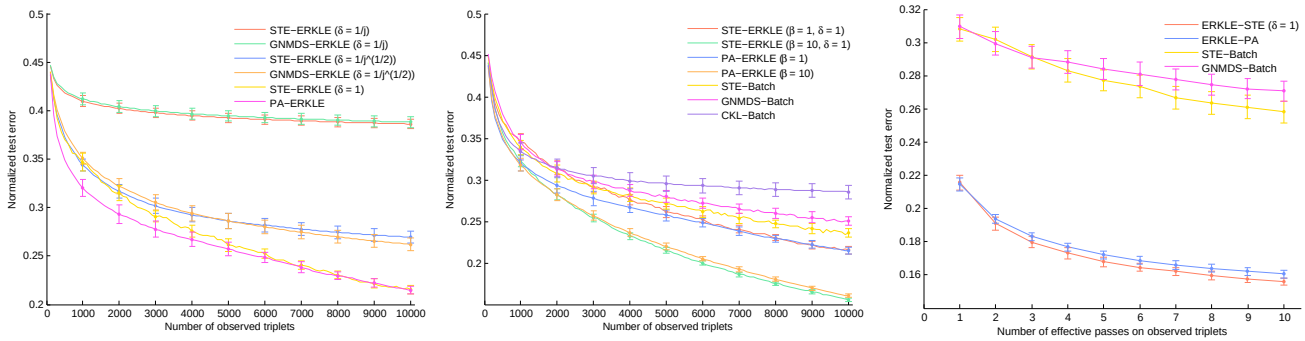
5 Experiments

In this section, we evaluate ERKLE by comparing it to batch RCKL methods. Batch methods are not truly applicable to the online learning setting, but can be applied in what is often called "mini-batches". In the mini-batch learning setting, every time a new batch of m triplets are received, batch RCKL is trained over all triplets obtained so far (e.g. if $m = 100$ after two mini-batches are received, then the batch methods are trained using 200 triplets). Thus, after all triplets are received via mini-batches, the batch methods are trained on the full training set, as in the true batch setting.

We evaluate each method on four different data sets. First, we start with a small-scale synthetic experiment to evaluate how the methods perform in an idealized setting. Second, a large-scale synthetic experiment is run to show how ERKLE and batch compare in terms of practical run time. Third, a data set of triplets over popular music artists is used to evaluate how the methods perform in a real-world setting with moderate triplet noise. Finally, ERKLE and batch RCKL are evaluated on a data set of triplets over scene images, which consist of a small number of triplets, thus focusing on the performance of these methods with very little feedback.

For these experiments, we wish to see how the learned kernels generalize to held out triplets, as triplets are obtained. This is important in real-world applications where the goal is to accurately model all the relationships among objects, not just the observed ones. Because of this, we use *normalized test error* for evaluation, which we define as the total number of unsatisfied test triplets by a learned kernel divided by the total number of test triplets. This metric effectively measures how well each method can utilize the obtained (training) triplets to generalize to unseen relative comparisons.

Unless otherwise noted, the experiments were run with



(a) Comparison of ERKLE learning rates δ (b) Test Error vs. # of observed triplets (c) Test Error vs. # of effective passes

Figure 1: Results from experiments on the small synthetic data set (10 trials)

the following specifications. Each method started with an initial kernel set to identity in order to give no method an advantage (all methods initially satisfy no triplets). All batch methods were terminated after a maximum of 1000 iterations or when the change in objective between iterations was less than 10^{-7}. We denote the batch methods with the suffix "-Batch" (e.g. STE-Batch), the ERKLE variants with "-ERKLE" (e.g. STE-ERKLE), and passive-aggressive ERKLE as PA-ERKLE. The mini-batch size is 100, and all methods are evaluated every 100 observed triplets.

We used the batch STE, GNMDS, and CKL (Crowd Kernel Learning [25]) MATLAB implementations specified by [27] in which the *eig* MATLAB function is used to perform eigdecomposition for projection onto the PSD cone. ERKLE was also implemented in MATLAB, where the *eigs* function is used to find a single eigenvalue/eigenvector pair with smallest eigenvalue. The τ hyperparameter was chosen to be the best performing setting over ten varying options. The timed experiments were performed on an Intel Core i5-4670K CPU @ 3.4 GHz with 16 GB of RAM on a single thread. Each experiment was performed with ten trials, each with different, randomly chosen test, train and validation sets. The error bars in the graphs represent the 95% confidence interval.

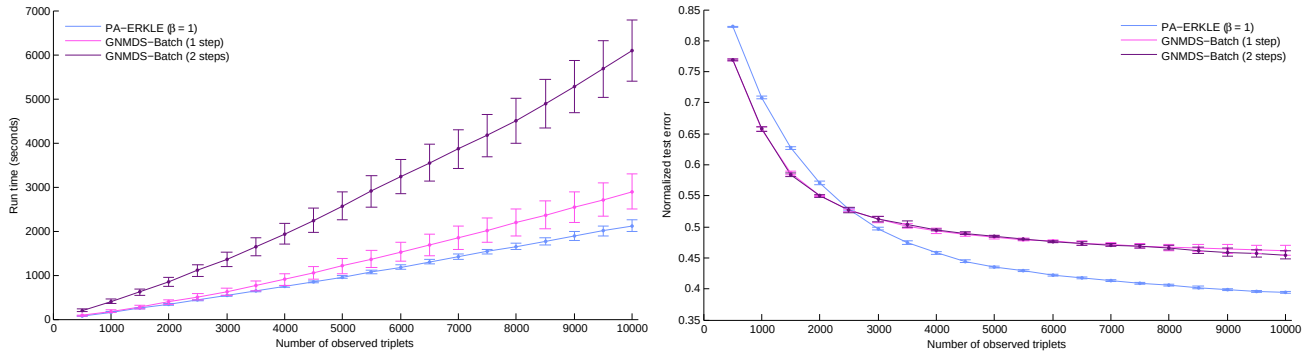
5.1 Small-Scale Synthetic Data Our first experiment is to test each method on an ideal, small-scale, synthetic data set. We created the synthetic data set by first generating 100 data points ($n = 100$) in $\mathbb{R}^{50}$ from $\mathcal{N}(0, 1)$. Using the distances between points, we answered all possible relative comparison queries, resulting in 485,100 triplets. 10,000 triplets were used as the train set and the rest were used as the test set.

Discussion: Figure 1a shows the effect that the learning rate parameter δ_j has on the performance of ERKLE as more triplets are observed in an online fashion. For a setting of $1/j$, the learning rate decays too rapidly to improve performance significantly after $j = 3000$. The learning rate $1/\sqrt{j}$ performs better, but still levels off, faster than the final two methods. The last two methods have learning rates that are independent of the number of observed triplets. STE-ERKLE with a constant learning rate and PA-ERKLE take steps solely based on how well the current solution satisfies the observed triplet, and vastly outperform the alternative learning rates based on number of observations. This result indicates that reducing the influence of a triplet because it was observed later has an adverse effect on the ability of a learned kernel to generalize to unobserved triplets.

Figure 1b shows the performance of STE-ERKLE (with δ_j set to 1), and PA-ERKLE compared to three batch RCKL methods. The τ hyperparameter was chosen by selecting the best setting over choices as evaluated on the test set. With a single pass over the data ($\beta = 1$), both ERKLE methods outperformed all batch methods slightly. With ten passes over the data, the ERKLE methods outperformed the batch methods by a large margin. In addition, the batch methods level off more quickly than the ERKLE methods, indicating that if more triplets were obtained, the ERKLE methods would further outperform even the batch methods. We believe that these results show that by minimizing the expected risk directly, ERKLE is able to learn a more general kernel than batch methods that minimize empirical risk.

To compare the methods in an implementation-independent manner, we evaluate two ERKLE methods and two batch RCKL methods as a function of how many effective "passes" each method performed on the data. For ERKLE, this amounts to the setting of the β parameter. For the batch RCKL methods, this is the number of full gradient steps it takes. Each method was run over all training triplets with the step size δ_j validated on the test set for the batch methods. Figure 1c shows the results, and clearly indicates that if only few passes through the data can be performed, then ERKLE will outperform batch methods by a wide margin.

5.2 Large-Scale Synthetic Data Next, we evaluated how PA-ERKLE compared to batch GNMDS in terms of practical



(a) Run time (seconds) vs. number of observed triplets

(b) Test error vs. # of observed triplets

Figure 2: Results from experiments on the large-scale synthetic data set (5 trials)

run time on a large scale experiment. For this experiment, we generated 5,000 data points in the same manner as the small-scale synthetic data. For each of the 5 trials, 10,000 randomly generated triplets were used as the train set and 50,000 were used as the test set. The batch methods were run in mini-batches of 500 triplets due to time constraints. The hyperparameter τ and the step size δ_i were chosen as the settings that best performed on the test set.

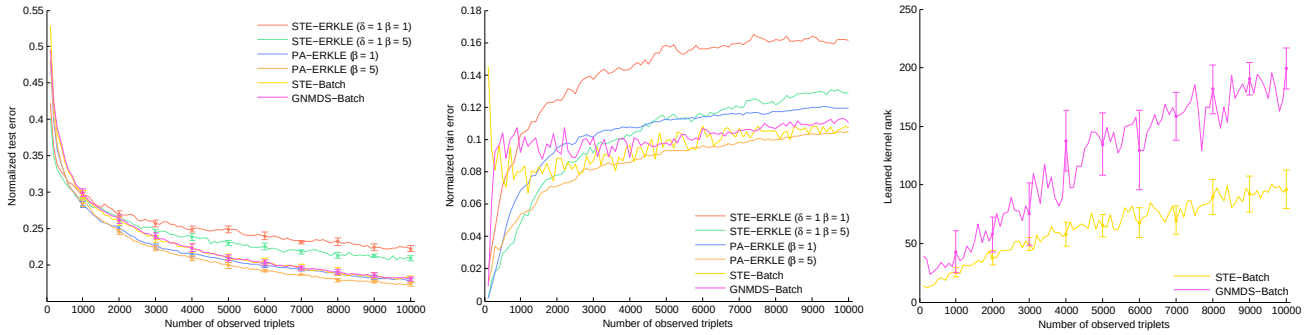
Discussion: Figure 2a shows the cumulative run time of one pass of PA-ERKLE, and 1 and 2 steps of batch GNMDS. The times shown for the batch methods are for the best chosen τ and not for the total time it took to find it. The figure shows that a single pass of PA-ERKLE is often significantly faster than a single gradient step of batch GNMDS. Two steps of GNMDS takes even longer. ERKLE can perform online updates much faster due to the efficient projection procedure as well as the ability to skip certain projections by estimating the lower bound. In this experiment, the mean number of eigenvalue/eigenvector computations over the 5 trials was 724.2 with a standard deviation of 3.7. Hence PA-ERKLE was able to skip the projection step roughly 93% of the time. Figure 2b depicts the test errors of each method. Initially, the batch methods perform better, but at around 2,500 triplets, PA-ERKLE outperforms the batch methods. This indicates that PA-ERKLE can produce truly online solutions in a single pass over the data, while maintaining competitive results with batch methods and having a faster run time.

5.3 Music Artist Similarity For the last two experiments we performed evaluations on real-world data sets. First, we performed an experiment using relative comparisons among popular music artists gathered from a web survey. The *aset400* data set [7] contains 16,385 relative comparisons over 412 artists. We randomly chose 10,000 triplets as the train set, 1,000 as the validation set for the τ parameter, and the rest were used as the test set. The aset400 data set presents a challenge not present in the synthetic data: It has a moderate amount of conflicting triplets, thus methods used in the evaluation must deal with noise within the data.

Discussion: Figure 3a shows how ERKLE and batch RCKL methods generalize to the test set. STE-ERKLE performs considerably worse than the other methods, most likely due to the noise in the observed triplets. The probability $p_t^{\mathbf{K}}$ used in STE-ERKLE decays rapidly. Thus, triplets that are in agreement with previously obtained triplets do not influence the learned kernel greatly. However, a conflicting triplet will make STE-ERKLE perform a relatively more drastic update. PA-ERKLE, however, is much more robust to noise due to the minimal step size taken to satisfy a triplet. Because of this, PA-ERKLE performs as well as the batch methods and often better when multiple passes are taken.

Figure 3b shows the training errors of each method. We use normalized training error as an objective-independent measure of how well each method fits to the observed triplets. The STE-ERKLE models are greatly effected by the presence of conflicts in that they do not learn a kernel that fits to a large number of the observed triplets. PA-ERKLE, on the other hand, is able to fit better to the set of observed triplets.

As previously discussed, dissimilar from batch methods ERKLE does not use trace regularization. Experimentally, however, we nevertheless find that our method outperforms batch methods that use trace regularization, in either producing low-rank or high-rank kernels. To demonstrate this, in Figure 3c we plot the ranks of the kernels learned by the batch methods. In our experiments, the range of potential τ values was set so that the batch methods never chose either the upper or lower bound. We did this to ensure that the range of regularization options were sufficiently strict or lenient. We observe that the batch methods generally produce low-rank kernels under a small number of triplets, but as the number of triplets are observed the rank increases. Our method is able to better generalize without using trace regularization, regardless of the preferred rank, due to the PA updates only satisfying triplets to the necessary extent.



(a) Test error vs. number of observed triplets (b) Train error vs. # of observed triplets (c) Kernel rank vs. # of observed triplets

Figure 3: Results from experiments on the aset400 data set (10 trials)

5.4 Outdoor Scene Similarity Our final experiment used triplets over 200 randomly chosen images of scenes from the Outdoor Scene Recognition (OSR) data set [19]. Relative comparison queries were posed to 20 people via an online system. After an initial 1200 randomly chosen queries were answered , 20 "rounds" of 200 triplets were chosen according to the adaptive selection criterion in [25], resulting in 3,600 total triplets. For each trial of this experiment, 1,000 triplets were randomly chosen as the test set, 1,000 as the train set, and 600 as the validation set for τ. This experiment is especially challenging for two reasons. First, this is the smallest experiment in terms of triplets, highlighting how the methods perform with little feedback. In addition, the adaptive selection algorithm chooses queries with the highest information gain, hence, the triplets are intentionally chosen to give disparate information about how the objects relate.

Discussion: Figure 4a depicts test errors on each method. We observe that STE-ERKLE consistently outperforms STE-Batch, and in particular STE-ERKLE performs well under a small number of triplets relative to all other methods. PA-ERKLE is comparable or outperforms its batch counterpart in GNMDS-Batch, given enough triplets (at least 500). However, PA-ERKLE performs quite well in training error compared to all other methods, indicating that even in such a challenging scenario, the passive-aggressive update scheme minimally interferes with previously obtained triplets.

6 Conclusion and Future Work

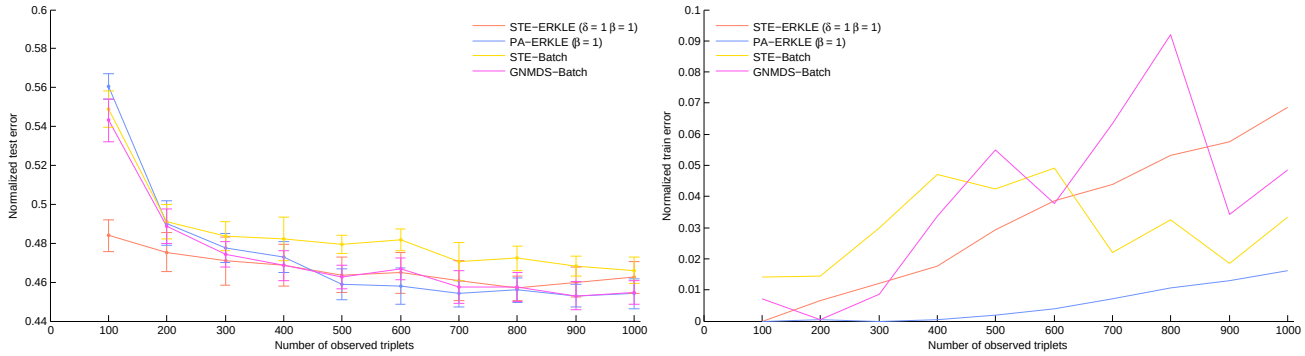
In this work, we developed a method to learn a PSD kernel matrix from relative comparisons given in an online fashion. By taking advantage of the sparse and low-rank structure of the online formulation, we show how to take stochastic gradient descent updates of complexity $O(n^2)$. We show how passive-aggressive online learning benefits our method in terms of generalizing to unseen triplets, and in conjunction with the stochastic gradient structure, enables us to perform a small number of necessary PSD projections in practice. Experimentally, we show on synthetic and real-world data that our method learns kernels that generalize as well and often better to held out relative comparisons than batch methods, while demonstrating improved run-time performance.

For future work, we wish to improve online RCKL in three ways. First, will explore the use of online trace regularization. If trace regularization is naively applied to the stochastic gradient in (4.9), the update becomes full-rank and our efficient projection procedure cannot be used. However, an efficient update scheme should be possible if the kernel itself is low-rank. We will investigate novel methods for appropriately weighting the trace in an online manner, so that we are consistent with the parameter-free property of PA-ERKLE. Second, PA-ERKLE performed well in our experiments with moderate triplet noise, however, it could be beneficial to explicitly handle conflicting triplets when they are observed. This can be done out of model using a denoising method [17], or in model using a threshold on the passive-aggressive learning rate. Third, while we improve the run time of RCKL methods in this work, ERKLE still cannot be practically applied to a web-scale [26] number of objects. To this end, we will investigate how to further improve the run-time of our method. Finally, one of the main benefits of having an online learning algorithm is the natural application of active learning methods. Prior work has proposed an adaptive selection scheme which operates in mini-batches [25]; however, such a scheme is too expensive to be applied online. We will investigate novel adaptive triplet selection methods which are both efficient and informative.

References

[1] S. Agarwal, J. Wills, L. Cayton, G. Lanckriet, D.J. Kriegman, and S.J. Belongie. Generalized non-metric multidimensional scaling. In *AISTATS*, 2007.

[2] A. Bordes, N. Usunier, and L. Bottou. Sequence labelling svms trained in one pass. In *Machine Learning and Knowledge Discovery in Databases*, pages 146–161. Springer, 2008.



(a) Test error vs. # of observed triplets

(b) Train error vs. # observed triplets

Figure 4: Results from experiments on the OSR data set (10 trials)

[3] L. Bottou. Online learning and stochastic approximations. *On-line learning in neural networks*, 17:9, 1998.

[4] J. Chen, T. Yang, and S. Zhu. Efficient low-rank stochastic gradient descent methods for solving semidefinite programs. In *AISTATS*, 2014.

[5] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *JMLR*, 7:551–585, 2006.

[6] Ç. Demiralp, M.S. Bernstein, and J. Heer. Learning perceptual kernels for visualization design. *Transactions on Visualization and Computer Graphics*, 20(12):1933–1942, 2014.

[7] D.P.W. Ellis, B. Whitman, A. Berenzweig, and S. Lawrence. The quest for ground truth in musical artist similarity. In *ISMIR*, 2002.

[8] E. Hazan and S. Kale. Projection-free online learning. In *ICML*, 2012.

[9] E. Heim, M. Berger, L.M. Seversky, and M. Hauskrecht. Efficient online relative comparison kernel learning (extended tech report). *arXiv preprint arXiv:1501.01242*, 2015.

[10] K.G. Jamieson and R.D. Nowak. Low-dimensional embedding using adaptively selected ordinal data. In *Allerton*, 2011.

[11] T. Joachims. Optimizing search engines using clickthrough data. In *SIGKDD*, 2002.

[12] M.G. Kendall. Rank correlation methods. 1948.

[13] A. Kovashka, D. Parikh, and K. Grauman. Whittlesearch: Image search with relative attribute feedback. In *CVPR*, 2012.

[14] J.B. Kruskal. Nonmetric multidimensional scaling: a numerical method. *Psychometrika*, 29(2):115–129, 1964.

[15] M. Levy and M. Sandler. Music information retrieval using social tags and audio. *Transactions on Multimedia*, 11(3):383–395, 2009.

[16] M. Mahdavi, T. Yang, R. Jin, S. Zhu, and J. Yi. Stochastic gradient descent with only one projection. In *NIPS*, 2012.

[17] B. McFee and G. Lanckriet. Learning multi-modal similarity. *JMLR*, 12:491–523, 2011.

[18] E. Moulines and F.R. Bach. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *NIPS*, 2011.

[19] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *IJCV*, 42(3):145–175, 2001.

[20] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011.

[21] H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[22] N.L. Roux, M. Schmidt, and F.R. Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *NIPS*. 2012.

[23] B. Schölkopf and A.J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[24] N. Stewart, G.D.A. Brown, and N. Chater. Absolute identification by relative judgment. *Psychological review*, 112(4):881, 2005.

[25] O. Tamuz, C. Liu, O. Shamir, A. Kalai, and S.J. Belongie. Adaptively learning the crowd kernel. In *ICML*, 2011.

[26] A. Torralba, R. Fergus, and W.T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, 2008.

[27] L. Van Der Maaten and K. Weinberger. Stochastic triplet embedding. In *MLSP*, 2012.

[28] C. Wah, G. Van Horn, S. Branson, S. Maji, P. Perona, and S.J. Belongie. Similarity comparisons for interactive fine-grained categorization. In *CVPR*, 2014.

[29] Z. Wang, K. Crammer, and S. Vucetic. Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale svm training. *JMLR*, 13(1):3103–3131, 2012.

[30] L. Xiao. Dual averaging method for regularized stochastic learning and online optimization. In *NIPS*, 2009.

[31] E. Zudilova-Seinstra, T. Adriaansen, and R. van Liere. *Trends in interactive visualization: state-of-the-art survey*. Springer, 2008.