EFFICIENTLY AND EFFECTIVELY LEARNING MODELS OF SIMILARITY FROM HUMAN FEEBACK

by

Eric Heim

B.S. in Computer Science, Kutztown University, 2009

M.S. in Computer Science, University of Pittsburgh, 2010

Submitted to the Graduate Faculty of

the Kenneth P. Dietrich School of Arts and Sciences in partial

fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2015

UNIVERSITY OF PITTSBURGH

KENNETH P. DIETRICH SCHOOL OF ARTS AND SCIENCES

This dissertation was presented

by

Eric Heim

It was defended on

November 20th, 2015

and approved by

Milos Hauskrecht, Professor, University of Pittsburgh

Rebecca Hwa, Associate Professor, University of Pittsburgh

Jingtao Wang, Assistant Professor, University of Pittsburgh

Aarti Singh, Associate Professor, Carnegie Mellon University

Dissertation Director: Milos Hauskrecht, Professor, University of Pittsburgh

EFFICIENTLY AND EFFECTIVELY LEARNING MODELS OF SIMILARITY FROM HUMAN FEEBACK

Eric Heim, PhD

University of Pittsburgh, 2015

Vital to the success of many machine learning tasks is the ability to reason about how objects relate. For this, machine learning methods utilize a model of similarity that describes how objects are to be compared. While traditional methods commonly compare objects as feature vectors by standard measures such as the Euclidean distance or cosine similarity, other models of similarity can be used that include auxiliary information outside of that which is conveyed through features. To build such models, information must be given about object relationships that is beneficial to the task being considered. In many tasks, such as object recognition, ranking, product recommendation, and data visualization, a model based on human perception can lead to high performance. Other tasks require models that reflect certain domain expertise. In both cases, humans are able to provide information that can be used to build useful models of similarity. It is this reason that motivates similarity-learning methods that use human feedback to guide the construction of models of similarity.

Associated with the task of learning similarity from human feedback are many practical challenges that must be considered. In this dissertation we explicitly define these challenges as being those of efficiency and effectiveness. Efficiency deals with both making the most of obtained feedback, as well as, reducing the computational run time of the learning algorithms themselves. Effectiveness concerns itself with producing models that accurately reflect the given feedback, but also with ensuring the queries posed to humans are those they can answer easily and without errors. After defining these challenges, we create novel learning methods that explicitly focus on one or more of these challenges as a means to improve on the state-of-the-art in similarity-learning.

Specifically, we develop methods for learning models of perceptual similarity, as well as models that reflect domain expertise. In doing so, we enable similarity-learning methods to be practically applied in more real-world problem settings.

Keywords: Similarity Learning, Kernel Learning, Metric Learning, Online Learning, Active Learning.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

1.0 INTRODUCTION

The ability to reason about how objects relate in a problem domain is of central importance for many machine learning methods to perform a task. To this end, a *model of similarity* that dictates how objects are compared to one another is required. Most commonly in machine learning, objects are represented by so-called feature vectors that are composed of uniquely defining characteristics drawn from raw data. Then, standard measures of similarity or distance are used on this predefined representation of objects. Take, for example, classical k Nearest Neighbor (kNN) classification. Class membership is determined by standard Euclidean distance between feature vectors. In this scenario, both the features and the Euclidean distance metric, together, define a model of similarity that allows the kNN classifier to perform a task. The success of many machine learning methods, including the kNN classifier in this case, is highly dependent on the assumed model of similarity on which it acts. As a result, when applying machine learning methods to a task, it is of the utmost importance to have a model of similarity among objects that is useful for that task.

Unfortunately, it is often the case in real-world applications that simple features constructed from raw data alone do not result in similarity models that that lead to desirable performance. In these instances, a model of similarity must be designed to reflect a notion of how objects relate that is beneficial for a task. It is this observation that has motivated the development of *similarity-learning* methods [157, 1, 75, 10]. Similarity-learning methods are a class of supervised learning methods that learn a model of similarity over a domain of objects given supervision or feedback regarding how objects relate. For instance, in classification tasks, the class labels themselves obviously give strong indication about which objects should be deemed similar and which should be deemed different in a problem domain. If that information was used to learn a model of similarity in which objects with the same class label are close together, then a kNN classifier using this model could have improved performance over using one ignorant of this information.

However, like other supervised learning methods, similarity-learning methods require a source of feedback to provide supervision. For many tasks, *humans* are called upon to be this source of feedback. Humans tend to have an innate understanding of how common objects relate that is difficult to capture from raw data alone. As a result, this *human perception* that most humans possess is useful in tasks where that require a model of similarity reflecting how the average human reasons about objects, such as in product recommendation, or for tasks that humans find easier to accomplish than computers, such as visual object recognition. In addition, certain humans have specific domain knowledge that allows them to make assessments on objects that the most humans cannot. Accordingly, *expert feedback* can be leveraged to learn a model of similarity for settings that require expertise in a specific domain.

In this dissertation, we focus on the general problem of learning models of similarity over objects from feedback obtained from humans. We consider both learning *perceptual similarity* that can be used in tasks that require a model that reflects how the average person views the relationships among objects, as well as *expert similarity* for tasks that call for a model conveying specific expertise in a domain. Associated with this problem are a considerable number of challenges related to the methodology in learning similarity models as well as practical challenges in obtaining and using human feedback. In the next section, we define these challenges.

1.1 CHALLENGES

Though having a model of similarity based on human feedback is useful in many machine learning applications, the act of learning such a model faces numerous challenges. To better understand these challenges we separate them into four challenge categories formed by the union of two criteria depicted in Fig. 1. One criterion separates the practical challenges that arise when prompting humans for feedback from methodological challenges in the learning algorithms themselves. The second criterion contrasts challenges in *efficiency* (reducing effort or cost) versus those in *effectiveness* (increasing accuracy). Below we explain each category:

Human Efficiency Obtaining human feedback requires time, and often money, especially in domains where expert feedback is required. As such, practically obtaining a sufficient amount of

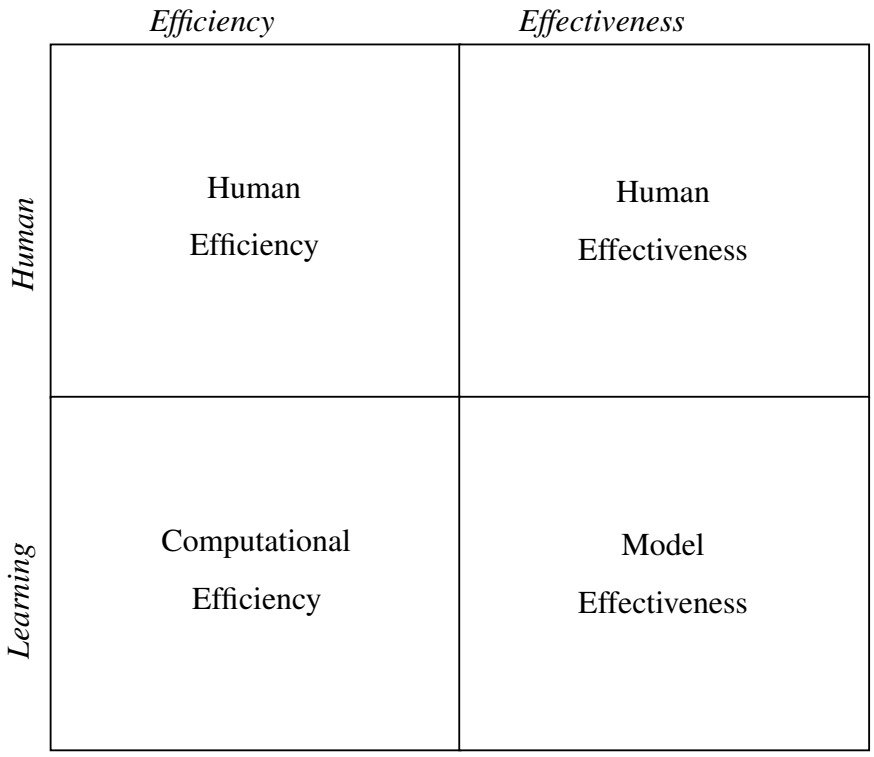


Figure 1: The four categories of challenges considered in this work

feedback to learn an accurate model is often the most costly step in learning similarity. Thus, effort should be made to reduce the burden placed on humans by making the most of their service. By doing so, one can lessen the amount of feedback required to learn a useful model. Stated more plainly, a human efficient similarity-learning method would require as little human feedback as possible to learn an accurate similarity model.

Computational Efficiency In order for a model of similarity to be useful, it needs to take a form in which learning algorithms can use as input. For this reason, the models considered in this work take forms that allow their use in many popular techniques in machine learning. Unfortunately, this often imposes strict constraints that an lead to considerable computational overhead in learning the models. If no attempt is made to mitigate this overhead, the practical run time of learning a similarity model could be too high to warrant their use for certain applications.

Human Effectiveness Many of the models considered in this work represent complex relationships among a large number of objects. Humans may find it difficult to directly provide the similarity being conveyed in these models, which can lead to inaccurate or contradictory feedback. This noise that can lead to learned models that do not faithfully reflect the desired relationships of objects. To ensure a similarity-learning method is human effective, one must either assume that the form of feedback being given leads to many noisy responses, and attempt to reduce the effect of the noise, or ask queries that humans find easy to answer, and adapt their methods to learn from a form of feedback that may not directly convey the same information the learned model provides.

Model Effectiveness Another important characteristic that models of similarity should possess is accuracy. More specifically, a model should accurately reflect the notion of similarity being conveyed through given feedback. Furthermore, a learned model should also convey a more general sense of similarity that captures object relationships not explicitly given, resulting in a more complete model than one that only echoes obtained supervision. Less obviously, it is also important to ensure that both the model being learned and the questions being asked are expressive enough to capture the true relationships among objects that the human is trying to convey. If the choice for either cannot express object relationships essential for a task, then no similarity-learning method can produce a model useful for that task.

The similarity-learning methods discussed in Chaps. 5, 4, and 6 provide methods with explicit focus on one or more of these challenges. Yet, providing a solution to a challenge in one category often creates a problem in another. For instance, one may choose to ask queries that humans find easy to answer (high human effectiveness), but by doing so, the number of queries needed to learn an accurate model may be large (low human efficiency). Or, one could learn a model of similarity that very closely adheres to given feedback (high model effectiveness), but the algorithm to learn such a model has a long run-time (low computational efficiency). In short, there is a trade-off between the challenge categories that must be considered when developing similarity-learning methods.

1.2 THESIS STATEMENT

Having defined these challenges, we are able to state the thesis of this dissertation:

By defining and explicitly attending to the practical challenges in learning models of similarity from human feedback, more efficient learning methods can be developed to learn more effective models that can be used in machine learning and data mining methods.

In order to confirm this thesis, we introduce four novel similarity-learning methods that focus on improving performance in one or more of the challenge categories defined in the previous section over previous methods, while ensuring not to reduce the performance in others to the point where they are impractical for real-world learning tasks. Indeed, the individual works discussed in the subsequent share a common goal: To develop similarity-learning methods can be used in practical, real-world learning scenarios where human feedback is obtained.

Two important items relating to this thesis statement must be defined. First, what is meant by a "model" of similarity? In this work, we define a model of similarity between objects to have two components: a *representation* of objects and a *measure* in which they can be compared with respect to that representation. Without a representation, objects cannot be compared to one another. Once objects have a representation, the way in which they are compared must be described by a well-defined measure. Often the line between these two components can be blurred. We make a focused effort on explicitly defining both components in the similarity models used in the similarity-learning methods outlined in this dissertation.

The other item needed to be defined is how one evaluates a similarity-learning method's performance in a challenge category. To measure computational efficiency, we use time complexity of the learning algorithms themselves as it a well-understood standard for algorithm analysis. In some cases we also show actual run-time of the algorithms to further emphasize differences in efficiency. The analogue of time complexity (number of elementary operations) for human efficiency is the number of queries asked to learn a given model. Human effectiveness is as much of a psychological issue as it is a machine learning one. Because of this, measuring human effectiveness is somewhat more nebulous than measuring computational or human efficiency. We dedicate a large

portion of Sec. 2.3 to discussing human effectiveness in terms of the forms human feedback can take.

Finally, the model effectiveness of a method depends on task-specific measures of accuracy. In settings where the goal is to learn a perceptual model, which then can be used as a basis for numerous other machine learning tasks, measures of accuracy should reflect how well the learned model adheres to the general similarity being conveyed by humans. In similarity-learning settings with a particular task in mind, the model can be judged by it's use for that task, thus task-appropriate measures of accuracy can be used. The distinction is subtle, but important. If one learns a model of similarity for the specific task of kNN classification, then metrics for classification accuracy can be used to measure model effectiveness. If the goal is to simply learn a model that reflects how humans reason about objects that can be used in various machine learning tasks, then accuracy is determined by how well the learned model captures human similarity. As we evaluate the methods in this dissertation, we will introduce evaluation metrics that measure task-appropriate accuracy.

Because of the close relationship the challenge categories share, we evaluate similarity-learning methods as a function of two or more metrics measuring performance in different categories. For instance, in Chap 5 we evaluate methods in terms of run time as a function of the number of queries answered. In Chap 4, we evaluate methods by a measure of model accuracy versus the number of queries answered. Though much of the discussion may revolve around performance in a single challenge category, the results can be viewed as comparing methods over multiple different categories. Through the evaluation of the methods introduced in this dissertation in terms of their ability to manage the effect of similarity-learning challenges, we aim to verify our hypothesis.

1.3 CONTRIBUTIONS

In Chap. 4 we address the inherent human inefficiency of learning a model of similarity from a particular form of human feedback called relative triplet feedback. Relative triplet feedback is useful for modeling human perception as it is both sufficiently human and model effective for that task. However, it it inherently human inefficient. To mitigate this human inefficiency, we developed a novel framework for incorporating auxiliary information about object relationships

into common similarity-learning methods that use relative triplet feedback as input. By doing so, we permit the auxiliary information to model trends that allow the similarity models learned within our framework to generalize to object relationships not explicitly given, while requiring less feedback. Then, we use the intuition gained by developing this framework, and a specific similarity-learning technique as a basis to develop a novel active learning method that intelligently selects queries in order to further increase human efficiency.

In Chap. 5 we address the computational inefficiency of common similarity-learning methods that learn from triplet feedback in the online learning setting. Traditional methods for this task consider the batch learning case. Their learning methods typically include a projection step that has time complexity $O\left(n^3\right)$. In the online learning setting, this is prohibitively expensive. In response to this, we develop a novel framework that can update models with online feedback in $O\left(n^2\right)$ time in the worst case. By taking advantage of the structure of our particular updates, we can even skip the projection step and perform constant time updates under certain conditions. In addition, we leverage recent techniques in online learning to determine how newly obtained feedback should affect the model. In doing so, we eliminate the need for users of our method to manually set any hyperparameters. Even with the improvements in computational efficiency and ease of use, we show that our method can learn models of similarity that are as accurate as traditional methods.

In Chap. 6 we address the human efficiency and model effectiveness of learning a model of similarity to be used in a classification task in the clinical medical domain. Here, we consider the application of learning an inference model to alert a clinician when a patient is at risk for a disease from patient Electronic Health Record (EHR) data and expert provided class labels. For such a method to be practically useful, the model of similarity used to make inferences should have two properties. First, it should be able to convey more information about patient relationships with respect to disease than simply providing alerts. Second, it should be as human efficient as possible as expert supervision is especially costly. We develop a similarity-learning method with a focus on these two goals. The model our method learns can be used to give additional information about a prediction, such as how confident the model is, and what specific characteristics of the patient factored into the prediction. In addition, we formulate a version of our method that can take auxiliary confidence information from expert labels and utilize it to

learn more accurate classification models from fewer labeled patient instances.

1.4 OUTLINE

The rest of this dissertation proceeds as follows. Chapter 2 provides a review of foundational topics to be used in subsequent chapters, including those in fundamental models of similarity and the forms human feedback can take. Chapter 3 discusses work related to the methods presented in this dissertation to confirm out thesis. More specifically, we review two classes of similarity-learning methods, and two orthogonal learning settings. Chapters 4, 5, and 6 present the main contributions of this dissertation. Finally, Chap. 7 concludes by recapitulating the contributions of this dissertation and discussing avenues of future work.

2.0 PRELIMINARIES

In order to begin our study of learning models of similarity from human feedback, we must first introduce the basic formalisms commonly considered similarity-learning. In this chapter, we define forms both human feedback and similarity models can take, and review foundations in each. Also, we provide a brief summary of the learning settings in which our methods act. We begin this chapter by defining the notation used in the remainder of this dissertation. Then, we review two fundamental measures of similarity that lay a foundation for the methods outlined in subsequent sections. Next, we discuss common forms of human feedback, provide example applications in literature, and compare them in terms of the challenge categories defined in the introduction. After that, we compare online learning to traditional batch learning as a means to motivate our online similarity-learning method in Chap. 5. Finally, we discuss active vs. random selection of queries to provided background for our active query selection algorithm presented in Chap. 4.

2.1 NOTATION

The following notation will be used in the subsequent. We denote vectors as bold, lower-case Latin or Greek letters (i.e. $\mathbf{a}$), and use subscripts on non-bold letters to denote individual elements in vectors (e.g. a_b is the bth element in $\mathbf{a}$). Matrices are denoted by bold, upper-case letters (e.g. $\mathbf{A}$), and we use superscripts to denote individual elements (e.g. $\mathbf{A}^{bc}$ is the element in the bth row and cth column). Script capital letters denote sets (e.g. $\mathcal{A}$). Italicized letters (i.e. a or A) denote either scalars or abstract objects. Which is being used should be clear in context. Set membership is denoted by subscripts (i.e. if $a_b \in \mathcal{A}$, then b uniquely identifies the object a_b from others in $\mathcal{A}$).

Special sets used throughout this work include: $\mathbb{R}$, $\mathbb{R}_+$, $\mathbb{R}^A$, and $\mathbb{R}^{A\times B}$, which are the set of

real numbers, the set of non-negative real-numbers, the set of real row vectors of length A, and the set of real matrices with A rows and B columns, respectively. For both vectors and matrices, the superscript T denotes the transpose. The rank of a matrix (number of linearly independent rows or columns) is denoted by the function rank. The trace of a matrix (sum of diagonal elements) is denoted by the function trace. A matrix $\mathbf{M} \in \mathbb{R}^{A \times A}$ is denoted as positive semi-definite (PSD), i.e. has no negative eigenvalues, by $\mathbf{M} \succeq 0$. The function $[\cdot]_+$ is the "hinge" function that is equivalent to $\max(0, \cdot)$.

2.2 FUNDAMENTAL MODELS OF SIMILARITY

A common way to reason about the relationships among objects is through *pair-wise* measures, meaning similarity is expressed over pairs of objects. Pair-wise models of similarity are attractive because they provide a direct way to compare one object to another, but also can be used to express more complex relationships. Similarity between a large group of objects can be determined by measuring how similar each pair in the group is. If similarity to a single object is of importance to a task (e.g. for ranking or clustering using centroids), all other objects in a domain can be compared, pair-wise, to an object of importance. For this reason, many standard methods in machine learning, data mining, and information retrieval assume a pair-wise model of similarity. However, some of these methods do not necessarily use a measure of similarity, but one of *distance*. In this work, we consider similarity and distance to be one and the same. Distance is most commonly interpreted as a measure of *dissimilarity*, which is the converse of similarity. As such, a measure of distance implies a measure of lack of similarity. Furthermore, we consider two specific models of similarity in this dissertation, each can be used to define a distance or similarity measure. These two measures represent a fundamental basis for which similarity-learning methods can build off of.

2.2.1 Distance Metrics

One of the most natural ways to quantify the relationships between pairs of objects is though the concept of *distance*. Distance, as most humans understand it, is concrete, but distance can also me

abstractly defined when considering objects without a specific representation. Let $\mathcal{X}$ be an abstract domain of objects, where $x_i \in \mathcal{X}$ is a single object. Let $d : \mathcal{X}$ x $\mathcal{X} \rightarrow \mathbb{R}$ be a *distance measure* that conveys a notion of distance or dissimilarity between two objects. This abstract definition of distance can characterize well understood notions of distance, such as physical distance between tangible objects ("That tree and that rock are 10 feet apart"), or less tangible ones ("The distance between these two directed graphs is 10"). The power in the abstraction of distance is that object in domains for which distance is not well-understood can have a means of comparison.

Distance measures allow for unitless real numbers to be considered as distances between objects without further definition. A more principled measure of distance can be defined through *metrics*. The function d' is a distance metric if it is a distance measure and also follows the following four properties for all $x_i, x_j, x_k \in \mathcal{X}$:

1. $d'(x_i, x_j) \geq 0$ (**Non-negativity**)
2. $d'(x_i, x_j) = 0$ iff $x_i = x_j$ (**Distinguishability**)
3. $d'(x_i, x_j) = d'(x_j, x_i)$ (**Symmetry**)
4. $d'(x_i, x_k) \leq d'(x_i, x_j) + d'(x_j, x_k)$ (**Triangle Inequality**)

In a sense, these properties express intuitive notions about the concept of distance, such as, distance should not be negative (non-negativity) and if a distance between two objects is zero, then those two objects are actually the same object (distinguishability). The space of objects $\mathcal{X}$ in conjunction with a distance metric d' over objects in $\mathcal{X}$ together make a *metric space*. Metric spaces can be viewed as models of similarity as they contain both components necessary to fulfill our definition of a model of similarity: a representation $\mathcal{X}$ and a means to compare objects d'.

Many useful measures of distance can be defined as metric spaces. Possibly the most popular one of which is the *Euclidean metric space*, where objects are represented as real vectors ($\mathcal{X} = \mathbb{R}^m$) and distance between objects is defined as:

$$d_2(\mathbf{x}_i, \mathbf{x}_j) = ||\mathbf{x}_i - \mathbf{x}_j||_2 = \sqrt{\sum_{k=1}^{d} \left(\mathbf{x}_i^k - \mathbf{x}_j^k\right)^2} \tag{2.1}$$

Here, $x_i = \mathbf{x}_i \in \mathbb{R}^m$, and $|| \cdot ||_2$ is the ℓ_2 or Euclidean norm:

$$||\mathbf{x}||_2 = \sum_{i=1}^{m} \sqrt{\mathbf{x}^i * \mathbf{x}^i} \tag{2.2}$$

This metric space is particularly convenient in machine learning where objects are typically represented as *feature vectors*. In data-driven tasks, object representations are often derived by extracting uniquely-defining characteristics about each objects from data. In many machine learning applications, these characteristics are modeled as real-valued features, which when combined create a vector of real values that define an object. The Euclidean distance between these representations define and "ordinary" or "straight-line" distance between objects, which machine learning methods can use as a way to compare objects.

2.2.1.1 Generalized Mahalanobis Distance Metrics

Because it is such an intuitive way to describe distance, the Euclidean distance metric is popular among machine learning methods to compare objects in a feature space. However, other distance measures can be used. Consider the case where objects $\mathbf{x}_i$ and $\mathbf{x}_j$ are drawn from a normal distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The *Mahalanobis distance metric* for these two objects is defined as follows:

$$d_{\boldsymbol{\Sigma}}(\mathbf{x}_i, \mathbf{x}_i) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)\,\boldsymbol{\Sigma}^{-1}\,(\mathbf{x}_i - \mathbf{x}_j)^T} \tag{2.3}$$

In short, (2.3) measures the distance between points with consideration of how the objects are distributed by normalizing based on the covariance between and within features. When $\boldsymbol{\Sigma}^{-1}$ is identity, (2.3) reduces to Euclidean distance. When $\boldsymbol{\Sigma}^{-1}$ is diagonal with elements $\sigma_i, ...\sigma_m$, (2.3) is known as *normalized Euclidean distance*:

$$d_{\Sigma}(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{k=1}^{d} \frac{\left(\mathbf{x}_i^k - \mathbf{x}_j^k\right)^2}{\sigma_j}} \tag{2.4}$$

In practice, $\boldsymbol{\Sigma}$ is estimated from data. However, the only requirement for (2.3) to be a valid metric is that $\boldsymbol{\Sigma}^{-1}$ be PSD. As a result, one can define a *generalized Mahalanobis distance metric*:

$$d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_i) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)\,\mathbf{M}\,(\mathbf{x}_i - \mathbf{x}_j)^T} \tag{2.5}$$

(2.5) can be understood by interpreting $\mathbf{M}$ as a *Gram matrix*. By definition of a Gram matrix, $\mathbf{M}$ can be decomposed: $\mathbf{M} = \mathbf{L}\mathbf{L}^T$, where $\mathbf{L} \in \mathbb{R}^{mxm'}$. Thusly, (2.5) can be written as:

$$\begin{aligned} d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) &= \sqrt{(\mathbf{x}_i - \mathbf{x}_j)\,\mathbf{M}\,(\mathbf{x}_i - \mathbf{x}_j)^T} \\ &= \sqrt{(\mathbf{x}_i - \mathbf{x}_j)\,\mathbf{L}\mathbf{L}^T\,(\mathbf{x}_i - \mathbf{x}_j)^T} \\ &= \sqrt{(\mathbf{x}_i\mathbf{L} - \mathbf{x}_j\mathbf{L})\,(\mathbf{x}_i\mathbf{L} - \mathbf{x}_j\mathbf{L})^T} \\ &= d_2(\mathbf{x}_i\mathbf{L}, \mathbf{x}_j\mathbf{L}) = d_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_j) \end{aligned} \tag{2.6}$$

With this, (2.5) can be interpreted as simply the Euclidean distance metric after a linear transformation of the objects into a m'-dimensional metric space. If $\mathbf{L}$ is not full-rank, then different objects in the original feature space can be projected onto the same point in the new space, and in doing so, breaking distinguishability. Thus, if $\mathbf{L}$ is not full-rank, then (2.6) is a *pseudometric* by having all the properties of a metric except distinguishability.

Because the only requirement for (2.5) to be a valid metric or pseudometric is that $\mathbf{M}$ is PSD, one can learn $\mathbf{M}$ or $\mathbf{L}$ so that the metric has properties that are beneficial to a task. This is the goal of *metric learning* [75, 10] methods: use information regarding how objects relate to learn a generalized Mahalanobis distance metric that reflects this information. Note, that "metric learning" is sometimes used broadly to mean the problem of learning any form of a metric. For the sake of this work, we use the term restrictively to mean learning a generalized Mahalanobis distance metric. For the problem settings considered in this dissertation, one can employ metric learning methods to learn a model of similarity that reflects a given set of human feedback. We review metric learning methods in section 3.1.

2.2.2 Kernels

While distance metrics represent a principled way to measure dissimilarity between objects, *kernels* can be used as a way to measure similarity based on many of the same foundation as metrics. Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a *kernel function*. Formally speaking, a kernel function is the canonical inner product of objects in $\mathcal{X}$. Similar to a metric space, an *inner product space* is defined by $\mathcal{X}$ and k together with the following assumptions. First, $\mathcal{X}$ must be a *vector space* for which vector addition and scalar multiplication are defined. Second, k must have three properties:

1. $k(x_i, x_j) = k(x_i, x_j)$ **(Symmetry)**
2. $k(a * x_i, x_j) = a * k(x_i, x_j)$ **(Linearity)**
3. $k(x_i, x_i) = 0 \implies x_i = 0$ **(Positive Definiteness)**

An instructive example that illustrates the relationship between kernel functions and similarity is *cosine similarity*:

$$sim(\mathbf{x}_i, \mathbf{x}_j) = \frac{\langle \mathbf{x}_i, \mathbf{x}_j \rangle}{||\mathbf{x}_i||_2 * ||\mathbf{x}_j||_2} \tag{2.7}$$

Here, objects are real vectors ($\mathcal{X} \in \mathbb{R}^m$), and $\langle \cdot, \cdot \rangle$ is the canonical inner product k for real vectors, which is also known as the the *Euclidean inner product*

$$\langle \mathbf{x}_i, \mathbf{x}_j \rangle = \mathbf{x}_i^T \mathbf{x}_j = \sum_{k=1}^{d} \mathbf{x}_i^k * \mathbf{x}_j^k \tag{2.8}$$

Cosine similarity is simply the cosine of the angle between two vectors, which can be interpreted as a magnitude-independent measure of how similar two vectors are. If both feature vectors are normal, then the denominator in (2.8) is one. As a result, if objects are represented by normalized feature vectors, the inner product between them is equivalent to the cosine similarity, and thus can be used as a method of comparison. Also, Euclidean distance can be defined in terms of inner products:

$$\begin{aligned} d_2(\mathbf{x}_i, \mathbf{x}_j) &= ||\mathbf{x}_i - \mathbf{x}_j||_2 \\ &= \sqrt{(\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T} \\ &= \mathbf{x}_i \mathbf{x}_i^T + \mathbf{x}_j \mathbf{x}_j^T - 2\mathbf{x}_i \mathbf{x}_j^T \\ &= \langle \mathbf{x}_i, \mathbf{x}_i \rangle + \langle \mathbf{x}_j, \mathbf{x}_j \rangle - 2 * \langle \mathbf{x}_i, \mathbf{x}_j \rangle \end{aligned} \tag{2.9}$$

Thus, a model of similarity defined by feature vectors and the Euclidean inner product as a kernel function that can be used to define distance, as well.

The power of kernels, however, comes from the so-called "kernel trick", popularized first by it's use in Support Vector Machine (SVM) classification [29]. Since then, there have been numerous machine learning methods that utilize kernels as a model of similarity to perform various tasks [115], collectively called *kernel methods*. The intuition behind the trick is as follows. Instead of assuming that the domain of objects is a vector space, assume that there exist a *feature mapping* $\phi : \mathcal{X} \to \mathcal{H}$ from the space of objects to an inner product space $\mathcal{H}$. Now, we can write the inner product of objects as $\langle \phi(x_i), \phi(x_j) \rangle_{\mathcal{H}} = k_{\mathcal{H}}(x_i, x_j)$. The mapping ϕ allows for abstract objects to

be given a representation. However, ϕ need not be explicitly defined as long as k is a valid inner product. Mercer's theorem states that a ϕ exists for a given kernel function k as long as k satisfies Mercer's Condition:

$$\int_{x_i \in \mathcal{X}} \int_{x_j \in \mathcal{X}} k\left(x_i, x_j\right) g\left(x_i\right) g\left(x_j\right) dx_i dx_j \geq 0 \tag{2.10}$$

for all square integrable functions g. As a result, objects from a domain can be given a model of similarity by choosing k that satisfies (2.10).

This observation is powerful for two reasons. First, for some domains there may not be clear ways to represent objects. One can still define a well-principled model of similarity that can be used in kernel methods by designing a valid kernel function over that domain. For instance, it may not be clear how to represent strings of characters so that they can be successfully used in machine learning methods. Nevertheless, kernels over strings have been developed that allow machine learning methods to be applied to tasks such as text classification [86]. In addition, objects may have vector representations, but they may be ill-suited for a task. For instance, if a linear classifier is applied to a setting where the objects are not linearly separable, then performance of that classifier could be poor. However, a proper choice of a kernel function k, can implicitly map the objects into a space where they are linearly separable, thus potentially increasing the performance of the classifier. For this, many different kernel functions have been designed to create more expressive models of similarity than just using the unchanged, original feature vector representation of objects.

A comprehensive review of all kernel functions currently in use is outside the scope of this dissertation. However, the *Gaussian kernel function* is of special note due to it's relationship to work later in this document:

$$k\left(\mathbf{x}_i, \mathbf{x}_j\right) = \frac{\exp\left(-d_2\left(\mathbf{x}_i, \mathbf{x}_j\right)/\sigma^2\right)}{\sigma\sqrt{2\pi}} \tag{2.11}$$

Here, it is assumed that the domain of objects are real-valued feature vectors. This function is at its maximum when the distance between objects is zero, and decreases as their distance increases. The parameter σ determines how rapidly the value of k decreases as distances between objects increase, and is often called the "bandwidth" of the kernel due to it's relationship to the width of the contours of a Gaussian distribution. The setting of the bandwidth has a rather large effect on the resulting model of similarity, which has lead to the popularity of the Gaussian kernel function

amongst practitioners, as it can approximate a large class of functions with different settings of the bandwidth [92]. Note, that the Gaussian kernel is a function of Euclidean distance. However, other distance metrics can be substituted to easily change the distance metric into a kernel function.

2.2.2.1 Kernel Matrices If an application requires a model of similarity not over a domain of objects, but a finite set of objects, then a function over the entire domain is unnecessary. In terms of kernels, this means that if similarity is only relevant to a task over a given set of objects, then simply having the values of the kernel function applied to all pairs of objects in the set is sufficient. To model this, often a *kernel matrix* is used. Consider a set of objects $\{x_1, x_2, ..., x_n\} \subset \mathcal{X}$ for which we require a model of similarity. A kernel matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$ is a model of similarity in which $\mathbf{K}^{ij} = k(x_i, x_j)$ for $i, j = 1, 2, ..., n$; thus, $\mathbf{K}$ models similarity amongst all objects in question.

Learning such a model is practically useful. Much like having a valid kernel function eliminates the need to explicitly define a mapping function ϕ, learning a valid kernel matrix directly eliminates the need to define a specific kernel function k. The only two requirements for $\mathbf{K}$ to be a valid kernel matrix is that it must be symmetric ($\mathbf{K}^{ij} = \mathbf{K}^{ji}$) and PSD ($\mathbf{K} \succeq 0$). A special case of this is the *correlation matrix*, which imposes the additional constraint that the diagonal entries are one ($\mathbf{K}^{ii} = 1$ for $i = 1, 2, ...n$). Such a constraint means that the objects in the inner product space induced by $\mathbf{K}$ are normal. As a result, the entries of a correlation matrix are equivalent to the cosine similarity among objects. In addition, the Euclidean distance between objects in $\mathcal{H}$ can be defined with simple operations on the elements of $\mathbf{K}$:

$$d_{\mathbf{K}}(x_i, x_j) = \sqrt{\mathbf{K}^{ii} + \mathbf{K}^{jj} - 2\mathbf{K}^{ij}} \tag{2.12}$$

This comes directly as a consequence of construction of $\mathbf{K}$, the definition of k as an canonical inner product, and (2.9).

2.3 FORMS OF HUMAN FEEDBACK AND THEIR APPLICATIONS

The mathematical constructs used to model similarity are only one component in modeling human notions of similarity. Another component is the *feedback* used to learn these models. Humans are

able to convey how they view the relationships among objects in many different ways. Because of this, feedback can take different *forms* that impart different kinds of information about object relationships. In this section, we compare and contrast some of the most widely-used forms of human feedback in terms of the similarity-learning challenge categories. More specifically, different forms are more expressive than others, affecting model effectiveness. Some forms give less information per query, thus having an impact on human efficiency. Finally, humans have the ability to provide certain kinds of feedback more easily than others, clearly influencing human effectiveness. Note that other forms of feedback exist that are not explicitly mentioned in this section. We limit the discussion to a sampling of the most popular forms, but much of the analysis performed here can be applied easily to those not included in this dissertation.

For a more formal viewpoint, first let $\mathcal{Q} = \{q_1, q_2, ..., q_m\}$ be a set of *queries*, and $\mathcal{R} = \{r_1, r_2, ..., r_m\}$ be a set of valid *responses*. Though we refer to them as responses to emphasize the fact that they are how humans respond to queries, a name for elements in $\mathcal{R}$ more common to supervised learning settings is *supervision*. We use the terms interchangeably. We can view humans providing feedback as a function $f : \mathcal{Q} \rightarrow \mathcal{R}$, and a set of human feedback to be pairs of queries and their corresponding responses $\{(q_1, r_1), (q_2, r_2), ...\}$. Indeed, the goal of many similarity-learning applications is to estimate f by learning model M using a set of obtained feedback and a predefined way of answering queries $\hat{f}_M$. Different forms of feedback are defined by the queries that can be asked $\mathcal{Q}$, the valid responses to those queries $\mathcal{R}$, and the interpretation of f. The space of queries is defined by the number of objects considered in a single query. For instance, let $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ be a collection of objects. Label feedback asks queries about single objects, which we denote $\mathcal{Q} = \mathcal{Q}_1 = \mathcal{X}$. For pair-wise feedback, $\mathcal{Q} = \mathcal{Q}_2 = \mathcal{X} \times \mathcal{X}$, and so forth. Each of the following subsections discuss forms of feedback with a different choice for $\mathcal{Q}$.

2.3.1 Labels

The most basic form of feedback humans can provide is judgments on single objects, whose responses are commonly called *labels*. A *class label* indicates that an object is the member of a discrete category, i.e., $f(x_i) = r_i \in \{C_1, C_2, ...C_c\}$. Class labels are one of the most studied forms of feedback, as they are the supervision considered in classification, a problem studied as early as

the 1930s [48] with entire textbooks dedicated to the subject [93, 38]. When $c = 2$, class labels are called binary labels, which typically indicate the presence or absence of a characteristic. When $c > 2$, then the labels are called multi-class labels. Another special case is when a single object has multiple labels [137], for which classification methods are often called multi-label classifiers.

Labels can also be real-valued ($y_i \in \mathbb{R}$). Supervised learning techniques that use real-valued labels are regression methods, which have an even longer history than classification methods [82, 51]. Important special cases of regression labels can be created by restricting their domains. For instance, regression labels can have a probabilistic interpretation by restricting the labels to $[0, 1]$. Also, ordinal labels [89, 49] are values on on an arbitrary scale where only the relative ordering of the labels is considered significant. For example, if $y_i < y_j$, then this implies that $x_i \prec x_j$ (interpreted as "precedes") in a total ordering, or ranking of objects. This differs from traditional regression labels in that the exact difference between y_i and y_j is considered meaningless.

Of those listed in this section, the two kinds of labels classically used to obtain human feedback are class and ordinal labels. Class labels obtained from humans have been used in a variety of applications, ranging from perceptual tasks such as object recognition [36, 42, 46] to expert tasks like medical diagnosis [71, 107, 127], or bird call [2], leaf [77], and music genre classification [140]. There has even been work where human-perceptual class labels are used to mimic domain expertise [18]. As for ordinal labels, most recent usage has been in point-wise learning to rank techniques [32, 125, 27]. Here, the ordinal categories are meant to convey a human's preference of the objects. As such, learning to rank techniques are motivated by applications where the goal is to model human preference, such as document retrieval or collaborative filtering.

In all these works, the labels are obtained for the sole purpose of learning an inference model to predict the label of unseen objects, i,e, they aim to find $\hat{f}_M : \mathcal{X} \rightarrow \mathcal{R}$ that estimates f. However, recently there has been work to prompt humans for mid-level binary *attributes* [47, 79, 45, 103] whose purpose is to describe objects. The reason attributes are described as "mid-level" is that they are meant to represent concepts more abstract than what is typically reflected in features extracted from raw data, but less abstract than higher level concepts commonly represented by class labels. For instance, when describing images of scenes, features drawn from images of the scene can capture information about the color and shape of objects within the scenes. Binary attributes obtained from humans can describe more abstract concepts such whether scenes appear "cold",

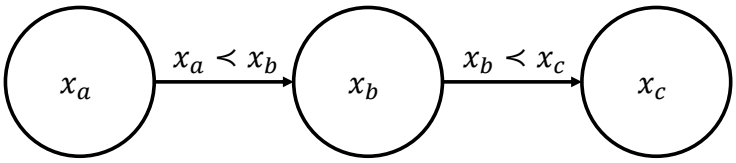
"man-made", and/or "enclosed". These human-perceptual attributes can be incorporated into a similarity model that can then be used in tasks where perception is valuable.

2.3.2 Pair-wise Feedback

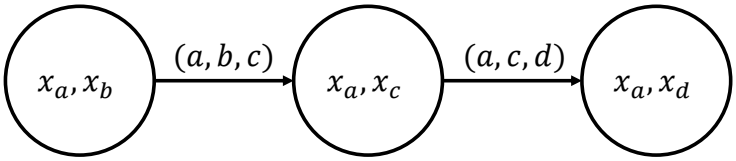
Humans can provide pair-wise feedback by considering two objects at a time, i.e. $f\left(q_i = (x_a, x_b)\right) = r_i$. For some forms of pair-wise feedback, it is assumed that queries are symmetric, meaning $f\left(x_a, x_b\right) = f\left(x_b, x_a\right)$. This is the case in pair-wise similarity/dissimilarity assessments. Here, it is assumed that pairs of objects belong to one of two classes: similar pairs of objects $\mathcal{S}$, or dissimilar pairs of objects $\mathcal{D}$. A response r_i indicates whether $(x_a, x_b) \in \mathcal{S}$ or $(x_a, x_b) \in \mathcal{D}$ for $x_a \neq x_b$. As such, r_i can be seen as a binary class label over pairs of objects. In a more practical sense, this form of feedback is most commonly used to indicate whether pairs of objects are members of the same cluster or not [146, 156, 9]. This allows for "human in the loop" clustering [44], where a human is shown a visualization of the objects organized into clusters. Then, the human can provide feedback indicating where the clustering is wrong by indicating examples of similar and dissimilar pairs not reflected by the current clustering.

Pair-wise feedback can also be *relative*. Like with similarity/dissimilarity assessments, relative pair-wise responses can be viewed as binary labels applied to pairs of objects. In this case, however, responses indicate whether $x_a \prec x_b$ or $x_a \succ x_b$ in a total order or ranking of objects. As a result, relative pair-wise feedback is asymmetric. Note that though this form of feedback is meant to convey information similar to ordinal labels, it only conveys a relative assessment of the ordering of two objects rather than the exact position of either object in the total ordering. Due to this, learning to rank methods that use this form of feedback are commonly categorized separately from those use ordinal labels [113, 68, 50, 19, 20].

As with labels, pair-wise feedback can convey mid-level attribute information. In this case, however, attributes are given in a *relative* fashion. Instead of describing the presence or absence of an attribute, a relative attribute [102, 72, 13] describes to what degree the object expresses the attribute relative to other objects. For example, an attribute can indicate that an image of Pittsburgh, Pennsylvania is "cold", but relative attributes can indicate that Pittsburgh is "colder" than Miami, Florida, but "warmer" than Antarctica. This can be viewed as ranking objects according to attributes



(a) Pair-wise



(b) Triplet

Figure 2: Directed graph depictions of relative feedback

rather than applying binary labels to them.

Two key ways pair-wise feedback differs from unary feedback are in concepts of *implied* and *conflicting* responses. Intrinsic in the nature of pair-wise feedback is the ability to identify when a response is implied through previous feedback and when a set of responses cannot all be true. For instance, a person providing similarity/dissimilarity assessments could respond that objects x_a and x_b and similar and that x_b and x_c are similar, i.e. $(x_a, x_b), (x_b, x_c) \in \mathcal{S}$. Through these responses it is implied that $(x_a, x_c) \in \mathcal{S}$, meaning we already know the answer to the query asking whether x_a and x_c are similar or dissimilar. If the person gives a response $(x_a, x_c) \in \mathcal{D}$, then, this clearly creates a contradiction with the previous responses. For labels, without other information about the objects or model assumptions (e.g. classes are linearly separable) it is impossible to tell whether the responses contain conflicts or if certain feedback is unnecessary.

We can draw these conclusions by assuming that similarity, and conversely distance, between objects is *metric* and follows the properties of metrics. Specifically in this case, we leverage the triangle equality to imply certain responses by *transitive* relationship among objects. A more illustrative example of this is looking at pair-wise ordinal feedback as a directed graph. In this case, vertices are objects and directed edges indicates the "precedes" relationship. Figure 2a illustrates an example with three objects where responses $x_a \prec x_b$ and $x_b \prec x_c$. The response $x_a \prec x_c$ is implied,

and $x_a \succ x_c$ would conflict with the other responses. The idea of forms of feedback possibly having redundant or conflicting responses is useful in comparing it to other forms, which we will discuss in Sec. 2.3.4.

2.3.3 Relative Triplet Feedback

The idea of relative pair-wise feedback can be expanded to more than two objects. When three objects are considered they form *relative triplet feedback*. Triplet feedback can be positive or negative. Positive triplet feedback is meant to communicate how the similarity between one pair of objects compares to the similarity of another pair. In English, a positive triplet query is a question of the form "Is object x_a more similar to object x_b than it is to object x_c?". Notationally, we define a positive triplet query as a tuple $(a, \{b, c\})$, where x_a is the *head* of the triplet being compared to a *pair* of objects x_b and x_c. The response to such a query is either b or c. For ease of discussion, we denote triplet feedback (the query and response together) as a triplet. For instance, the response "a is more similar to b than it is to c" implies the query $(a, \{b, c\})$ was asked, thus we can represent triplet feedback as simply (a, b, c). Note the subtle notational difference from queries in that b and c here are ordered in the feedback triple to indicate which was chosen.

Negative triplet queries are questions of the form: "Which one of these three objects, x_a, x_b, and x_c is different than the other two?", and the response is any of the three objects. Much like the positive form, negative triplet feedback gives relationships among each pair of objects. For instance if x_a, x_b, and x_c are being compared and a human chooses x_c, this can be interpreted that x_a and x_b are more similar than both x_a and x_c, and x_b and x_c. Thus, a single unit of negative feedback gives two positive triplet responses: (a, b, c) and (b, a, c).

Two other special cases of triplet feedback include quadruplets, which use queries of the form: "Which pair of objects, x_i and x_j or x_k and x_l are more similar?", and multi-triplets, which ask humans to select all objects from a set that are similar or dissimilar to each other. All of these forms of triplet feedback are closely related. Note that many works using either negative triplet feedback [59, 35, 5] or multi-triplets [155] effectively distill their forms to positive triplet feedback. Positive triplet feedback itself is a special case of quadruplet feedback in which the two pairs being compared share a common object. As a result, methods that use quadruplet feedback [74, 4, 141] can often

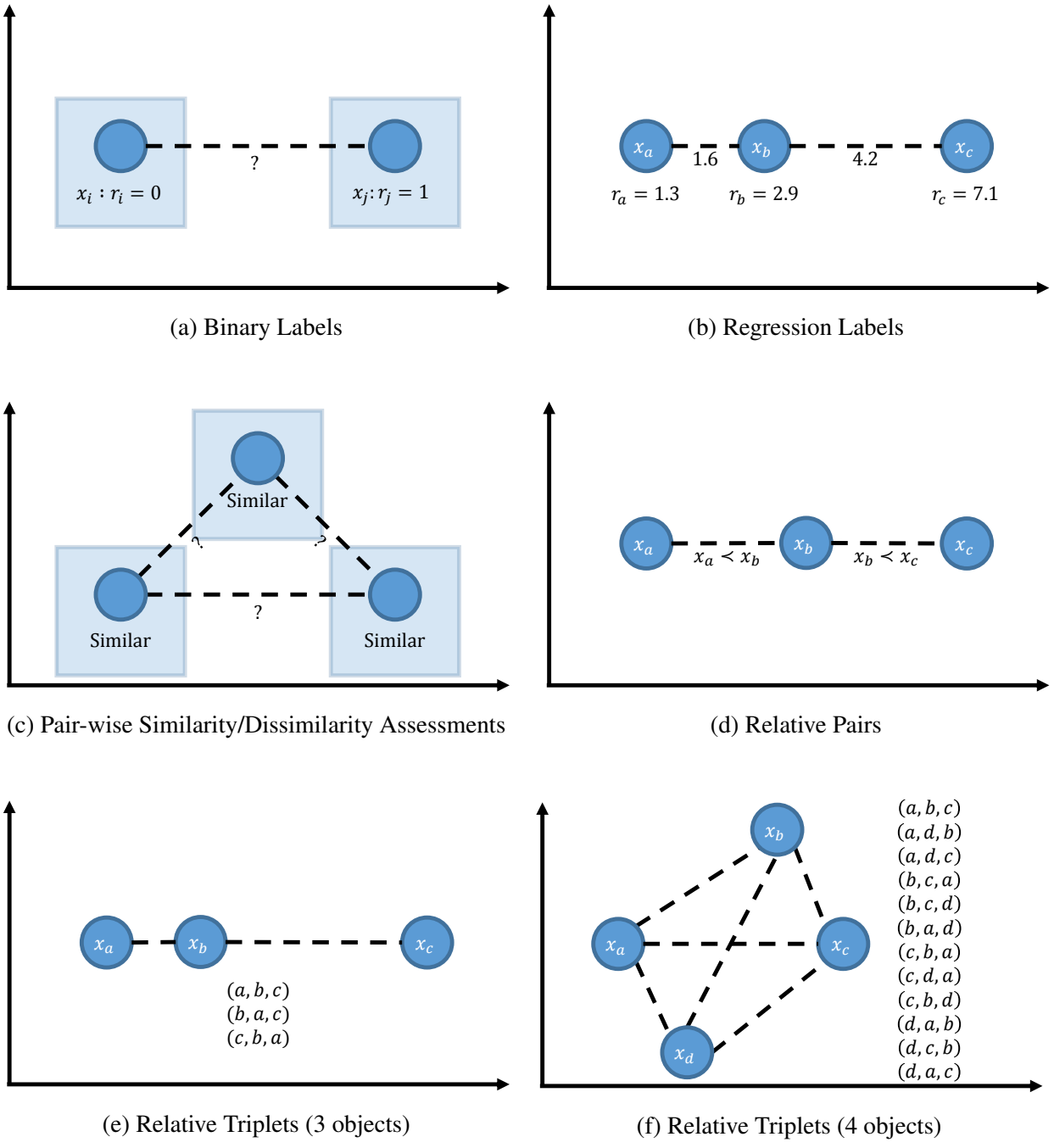
easily me applied to scenarios where positive triplet feedback is obtained. For ease of discussion going forward, we limit ourselves to positive triplet feedback (henceforth, simply "triplet feedback") and note that much of the discussion and the techniques in the remainder of this dissertation can be extended to these other cases.

Triplet feedback is powerful in that it can impart similarity information in a general sense. Almost all forms of feedback require context for humans to provide it. Class labels need names so humans know how to categorize objects. Pair-wise relative feedback requires a context in which objects are ranked, such as relevancy to a query if the objects are documents, or a named attribute if relative attributes are trying to be learned. Triplet feedback requires no context and solely relies on a person's ability to make relative assessments about how she views the objects in a general sense. As a result, triplet feedback can be used in many different applications including semi-supervised clustering [5], density estimation [141], and even to identify multiple unnamed attributes [6]. More commonly, however, triplet feedback is used to learn general models of similarity that simply capture how humans perceive objects [4, 132, 91, 143]. It it for this reason that the methods presented in this dissertation use triplet feedback to learn perceptual models of similarity.

As with relative pair-wise feedback, triplet feedback can conflict or be implied. If responses (a, b, c) and (a, c, d) are given then (a, b, d) is implied and (a, d, b) would conflict. Also like pair-wise feedback, triplet feedback can be represented as a directed graph [91]. We illustrate an example of a triplet graph in Fig. 2b. Here, a node in the graph represents a pair of objects while a directed edge represents a triplet response, or a "more similar than" relationship. Besides the interpretation of nodes and edges, the difference between pair-wise and triplet graphs is in how valid edges are defined. In a pair-wise graph, a valid edge can be drawn between any two nodes. In a triplet graph, only nodes with a common object can have an edge between them, as a unit of triplet feedback requires a single single "head" object. This implies that triplets allow not for a total ordering of the pair-wise similarity between objects, but a *partial ordering*.

2.3.4 Comparison of Forms

Clearly, different forms of feedback are meant to convey different kinds of information. Yet, the differences between the forms affect more than the information content within responses. These



(a) Binary Labels

(b) Regression Labels

(c) Pair-wise Similarity/Dissimilarity Assessments

(d) Relative Pairs

(e) Relative Triplets (3 objects)

(f) Relative Triplets (4 objects)

Figure 3: Example object embeddings using different forms of feedback

differences must be considered when obtaining feedback for practical applications. In this section, we compare and contrast each form of feedback in terms of three of the four challenge categories in

learning models of similarity. Note that computational efficiency is not a good metric in which to compare forms of feedback as it is largely dependent on the learning algorithms themselves.

2.3.4.1 Model Effectiveness A form of feedback's model effectiveness can be more succinctly described as its *expressiveness*. While certain forms are meant to convey a particular kind of information, others may be able to express more complex relationships among objects. If a task requires an intricate model of similarity, then prompting humans for simple forms of feedback may not be appropriate. To guide our discussion on expressiveness, we view forms of feedback in terms of learning a particular models of similarity called object *embeddings*. An embedding of objects is a placement of each object in a d-dimensional real space where common Euclidean notions of distance and similarity can be used. An object's placement in the space is entirely determined by the feedback given over it. As such, embeddings can faithfully reflect given feedback without biasing the model to common external factors, such as an assumed feature representation. Even so, these models are not perfect, as certain forms do not provide information regarding relationships, thus the embeddings must make certain assumptions. In practice, it is up to the embedding-learning methods themselves how to reconcile this lack of information in the learned embedding. For our comparisons, we remark when this is the case and use it to discuss the expressiveness of the models.

Figure 3 shows a number of object embeddings that reflect different sets of feedback of different forms. The most simple forms are those that express a categorical relationship among objects: class (including binary) labels, and pair-wise similarity/dissimilarity assessments. In our examples, objects expressed as similar or of the same class are positioned on exactly the same point, which we display in the figures as plates over unlabled points. Objects of different categories are embedded as different points. However, other embedddings that reflect the given feedback are possible. How pairs of objects in different categories or within the same category compare to each other is not expressed. In terms of embedding objects, this means that the distance between objects of different categories should be greater than those of the same category, but the exact distances are unknown. In fact, Fig. 3c shows a two-dimensional embedding from pair-wise similarity/dissimilarity assessments, but an accurate one dimensional embedding, as in Fig. 3a, is also possible.

All other embeddings depicted in Fig. 3 must position individual objects with nonzero distance between them to faithfully reflect their respective sets of feedback. Regression labels allow for

exact distances between objects to be known in the embedding (Fig. 3b). For the relative forms, the exact distance between objects is not known because their relationships are relative. In Fig. 3d, the feedback dictates that x_a precedes x_b in a total ordering, but information defining exactly how similar those two objects are is not given. Thus, the embedding that reflects the total ordering of objects can reflect the total ordering of objects, but must assume some distance between x_a and x_b. This is also true for relative triplet feedback. However, as more objects are sampled and feedback given, the number of constraints on pair-wise distances between objects increase. As a result, the distances between pairs of objects becomes more defined [126].

Possibly the most important factor when discussing expressiveness is the *dimensionality* of the spaces for which a form is able to convey. All embeddings can accurately reflect their respective given feedback in one dimension: Regression labels simply indicate position on the real-number line, relative pairs indicate relative position in a single-dimensional total ordering, etc. The only one that cannot be accurately captured in one dimension is the triplet responses in 3f. There does not exist a one dimensional embedding for which every triplet response on the right can be reflected in the embedding (e.g. for (a, b, c) objects a and b should be closer together than a and c). As a result, if the true notion of similarity trying to be gained is multidimensional, relative triplets are the only one of these forms that can express multidimensional relationships. This is not to say that other forms cannot be modified to convey multidimensional similarity. Multiple binary or regression labels or relative pair-wise comparisons across multiple dimensions can be used to describe multidimensional spaces, but it requires putting names to the specific labels or comparisons. For instance, multiple, different relative attributes can be found over a given set of scenes, such as "warmth" or "plant growth", but these individual attributes require names so humans have context for which comparisons can be made. Relative triplet feedback need not name the kind of similarity trying to be gained. As a result, if perceptual similarity is trying to be obtained from humans without biasing them to specific attributes, then relative triplets can be used.

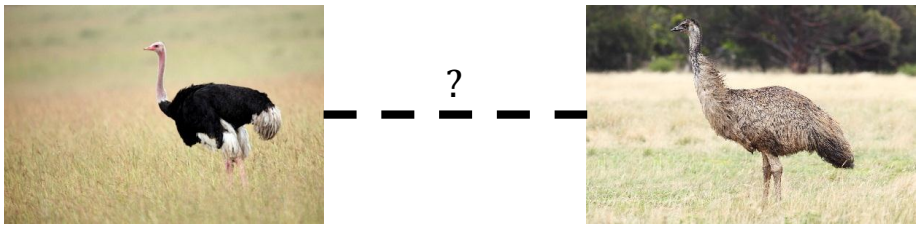
2.3.4.2 Human Efficiency Informally, human efficiency for a form of feedback is the amount of information contained in a response. If humans can entirely describe a desired model of similarity by only answering a few queries of a particular form, then that form of feedback is considered human efficient. Unfortunately, because forms convey relationships of different complexity, they

are not directly comparable. Nevertheless, we attempt to compare each form in terms of human effectiveness, while introducing some alternative forms that could be more efficient, but have high impact on human effectiveness (discussed next section).

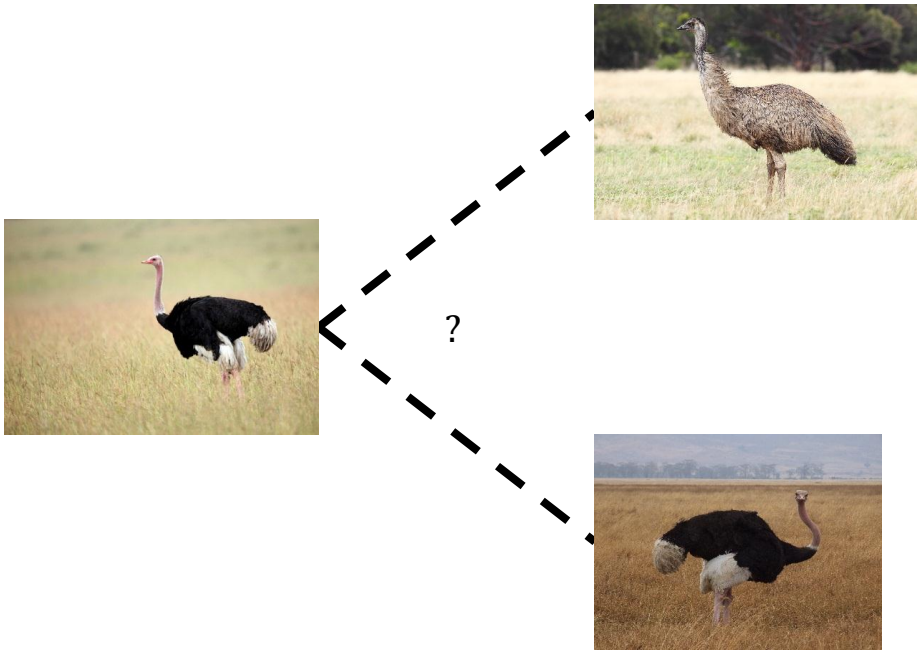
We compare each form by the total number of unique queries can be asked when procuring feedback over n objects. As a result, we consider forms over fewer objects per unit of feedback to be more human efficient. Labels are the most efficient of the forms of feedback discussed, requiring only n queries to be responded to in order to obtain a complete set of responses. Symmetric pair-wise feedback requires $\binom{n}{2}$ queries, and asymmetric pair-wise feedback requires $n!/\,(n-2)!$; both of which are $O\,(n^2)$. Finally relative, triplet feedback requires $n * \binom{n-1}{2} = O\,(n^3)$ queries (each head object can be compared to two other unique objects).

While this metric of comparison sheds some light onto the human efficiency of different forms, it does not tell the whole story. While pair-wise and triplet feedback has many more potential queries than labels, many responses can be redundant. As stated previously, some sets of pair-wise and triplet responses imply other responses without having to pose those queries to a human. As a result, the number of queries needed to be asked in order to obtain a complete set of responses can be less than the total number possible. For example, finding a total ordering using pair-wise relative comparisons is known to require only $O\,(n * \log n)$ units of feedback in the worst case [70]. Similar reasoning can be used for triplet feedback. Triplet feedback can be seen as an ordering with respect to a single object of all other objects. In other words, it is n separate orderings of $n-1$ objects. Because of this, finding a partial ordering that satisfies a complete set of triplet feedback using comparison-based sorting techniques requires only $O\,(n^2 * \log n)$ units of feedback in the worst case.

The relative forms of feedback are clearly less human efficient than labels, though they have benefits in terms of model effectiveness. This provokes the question: Are there forms that have the expressiveness of relative feedback, while being more efficient? To answer this, consider instead of comparing pairs of objects to each other as in relative triplets, simply asking exactly how different two objects are. We will call this form *quantitative pair-wise* feedback. This from can still be used to elicit multidimensional models of similarity, but requires asking only $O\,(n^2)$ queries to obtain a complete set of responses. Furthermore, quantitative pair-wise feedback can be interpreted as an exact measure of distance, eliminating any ambiguity in pair-wise distances between objects. In the



(a) Quantitative pair-wise query



(b) Relative triplet query

Figure 4: Direct quantitative query vs. relative query

next section, we discuss why prompting humans for exact, quantitative assessments about similarity can have practical issues, despite being otherwise attractive.

2.3.4.3 Human Effectiveness We define the human effectiveness of a form of feedback as the ease in which humans can answer queries of that form. If humans find queries to be easy to answer, then their responses contain more accurate information. While this particular challenge is more qualitative than quantitative, human effectiveness can be studied in a larger sense by what tasks

humans tend to find easy, and which they find difficult; A problem studied in cognitive psychology. Previous work [28] suggests that humans naturally categorize objects when they reason about them. As such, feedback that requires humans to reason about objects in a categorical sense (class labels, pairwise similarity assessments), tend to be easier for humans to respond to.

Also, humans tend to find relative assessments of objects easier to provide than exact, quantitative ones [135, 130]. As a result, relative feedback is much easier to provide than exact measure of similarity or distance. Consider, again, quantitative pair-wise feedback. In Fig. 4a, there are pictures of an emu and an ostrich. A quantitative pair-wise query would ask a human for a number, either discrete or real on a scale, of how similar the two images are. Such a query may be difficult for a human to answer because of the many factors involved in how she compares birds. In Fig. 4b, there is a depiction of a relative triplet query. A human would most likely be able to answer this query with more ease because it requires her to compare two pair-wise similarities relative to each other.

In addition, quantitative pair-wise feedback can be difficult for humans because they may not have the correct idea of the domain of objects. Assume the first query posed to a human is the one illustrated in Fig. 4a. If the human thinks the task is comparing animals, or birds, she could give a response indicating the images are very similar. However, if the domain of images is all ostriches and one emu, then she may have answered this query differently. Conversely, because relative triplet feedback can ground the response relatively, order of queries or knowledge of the domain is much less of a factor in the responses a human can provide. Therefore, responses could be more consistent. A similar argument can be made for relative pair-wise feedback versus ordinal labels. A human may find providing the exact positioning of an object in the total ordering to be challenging without knowledge of other objects. Because of this, ordinal labels may be difficult for humans to provide compared to relative pair-wise feedback.

2.3.4.4 Summary To summarize the strengths and weaknesses of each form of feedback discussed in this section:

Class Labels

+ $O(n)$ unique queries

+ Asks for categorical information, which is often easy for humans to provide

- Cannot express information about the relationship between categories or objects within the same category
- Limited to modeling categories

Ordinal Labels

+ $O(n)$ unique queries
+ Can convey ordering information
- Exact position of an object can be difficult for humans to provide without knowledge of all other objects
- Limited to conveying single dimensional relationships among objects

Pair-wise similarity/dissimilarity feedback

+ Asks for categorical information, which is often easy for humans to provide
+ Can be used to find clustering or unnamed classes of objects
- $O\left(n^2\right)$ unique queries
- Cannot express information about the relationship between categories or objects within the same category

Relative pair-wise feedback

+ Asks for relative information, which is often easy for humans to provide
+ Can convey ordering information
- $O\left(n^2\right)$ unique queries
- Limited to conveying single dimensional relationships among objects

Relative triplet feedback

+ Asks for relative information, which is often easy for humans to provide
+ Can convey multidimensional relationships among objects
- $O\left(n^3\right)$ unique queries
- Does not exactly convey pair-wise similarity, only defines it relatively

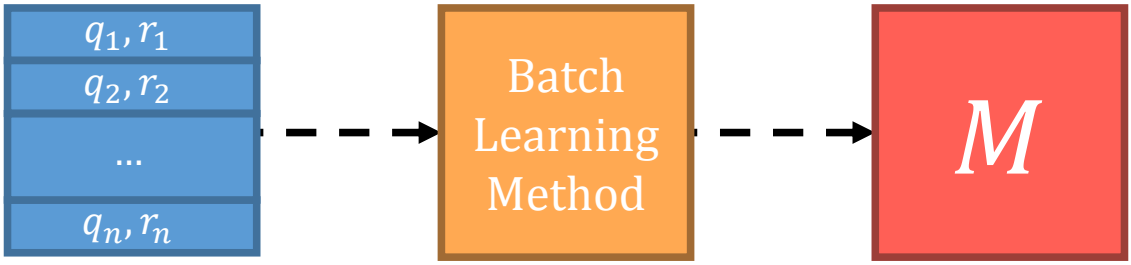
2.4 SIMILARITY-LEARNING SETTINGS

Now that we have discussed both the input (forms of feedback) and the output (models of similarity) of our problem, we can discuss the settings in which similarity-learning methods can act. Specifically, we discuss learning settings defined by two characteristics. The first concerns itself with how feedback is obtained for learning a model: all at once, or in small amounts at a time. The second separates learning settings where feedback is obtained at random, as opposed to settings where the learner is able to choose which queries humans are to respond to.

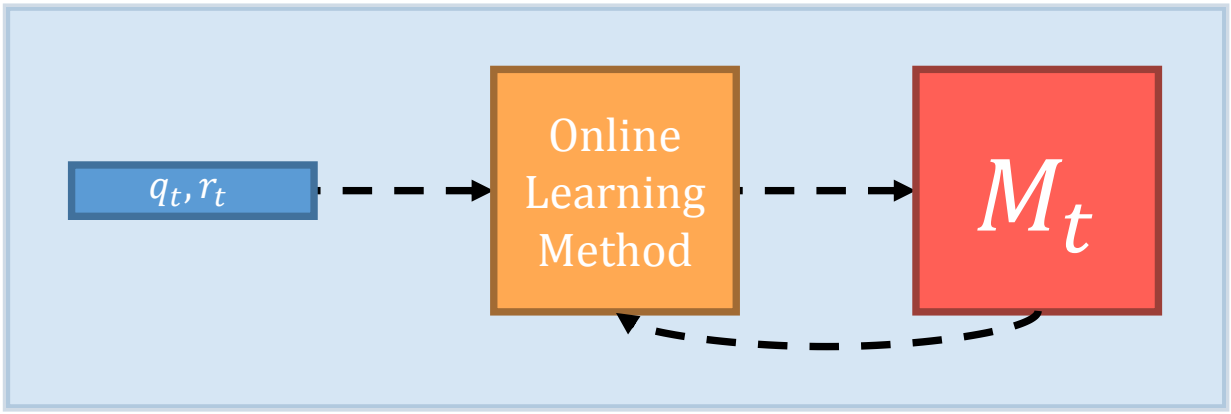
2.4.1 Batch versus Online Learning

Traditionally in machine learning, data collection is assumed to be a separate step from learning a model from the data. As a result, the majority of machine learning applications consider feedback or supervision to be obtained in *batch*. Figure 5a depicts the batch learning process. Here, it is assumed that all feedback to learn a model is available at a single point in time, namely *train time*. At train time, all feedback $\{(q_1, r_1), (q_2, r_2), ..., (q_n, r_n)\}$ is used as a *train set* $\mathcal{D} \subset \mathcal{Q} \times \mathcal{R}$ to be used as the input to a learning algorithm, which in turn produces a learned model M. Generally speaking, the model should in some way reflect the given feedback, and can then be used in the application it was made for. For instance, if the goal is to learn a function $\hat{f}$ that estimates f, then a learned model M can be a component in such a function, i.e $\hat{f}_M : \mathcal{Q} \to \mathcal{R}$. In Secs. 3.1 and 3.2 we discuss how this is done for a variety of tasks and models of similarity.

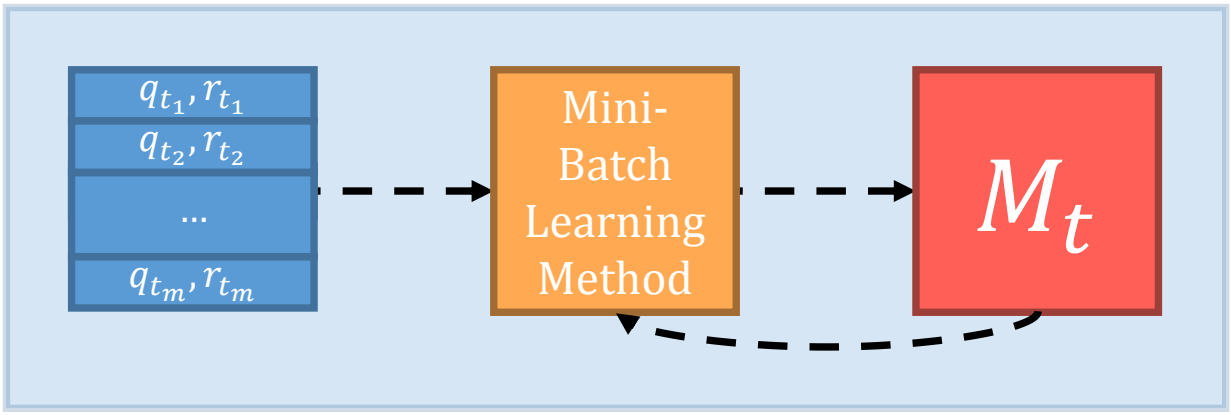
However, in many real-world problems, feedback is being obtained continuously, and models learned from feedback must be updated as feedback is obtained. This is the setting in which *online learning* methods act. We illustrate online learning in Fig. 5b. Here, the plates around components of the diagram indicate iteration. In this setting, the goal is to update a learned model as feedback is continuously being obtained. More specifically, at each time step t, a single unit of feedback (q_t, r_t) is given, and an updated model M_t is produced. Such a model should not only reflect feedback obtained at time t, but also all that which was previously obtained. The hope is that M_t is at least as accurate as using all t responses as a training set in batch learning. In the simplest case at time t, a single unit of feedback, and the model from the previous time step $t-1$ are used as input



(a) Batch Learning



(b) Online Learning



(c) Mini-Batch Learning

Figure 5: Batch, online, and mini-batch learning settings

to an online learning method. The model produced by the learning method should then contain information from the previous model as well as the most recent feedback.

Online learning is attractive for multiple reasons. Online methods tend to be simple with fast, easy to implement updates. By virtue of this, they can be utilized to quickly create up-to-date models in applications where feedback is obtained in a streaming fashion, which are many (spam filters, stock market readings, human-computer interactions, etc.). Also because of their simplicity, online learning methods tend to be amenable to analysis, resulting in many theoretical guarantees. Theoretical work typically studies online learning using *regret analysis*, which defines online learning slightly differently than we do in this work. Because this dissertation is focused more on the practical application of online methods we defer to the following references for discussion of online learning methods under the lens of regret analysis [21, 120].

Mini-batch learning can be seen as a compromise between the two extremes of batch and online learning, and is shown in Fig. 5c. In this case, feedback is continuously being obtained in small sets or "mini-batches" of m units. As such, it is very similar to the online case, but instead of individual units of feedback obtained at each time step, multiple units are provided.

2.4.1.1 Batch versus Stochastic Gradient Methods

A particularly popular class of learning methods that can be used for either batch, mini-batch, or online problem settings are *gradient descent* methods. Consider, again, the case where we wish to estimate f by learning M to be used in $\hat{f}_M$. In the *batch gradient* case, M is found by optimizing an objective over all training feedback. The objective is often formulated as a loss function that measures how well $\hat{f}_M$ fits to the training data:

$$L\left(\hat{f}_M, \mathcal{D}\right) = \sum_{(q_i, r_i) \in \mathcal{D}} l\left(\hat{f}_M\left(q_i\right), r_i\right) \tag{2.13}$$

In this case, the loss function over all training data is the sum of the loss l with respect to individual training samples of feedback. Batch gradient descent procedures minimize the loss L by performing the following updates until a convergence criterion is met:

$$M_i \leftarrow M_{i-1} - \delta_i * \nabla L\left(\hat{f}_{M_{i-1}}, \mathcal{D}\right) \tag{2.14}$$

where δ is the *learning rate* or *step size*. By performing these updates, batch gradient descent procedures are minimizing the *empirical risk* of prediction on the training set. Empirical risk is

meant to approximate the risk over the true distribution of feedback $P(\mathcal{Q}, \mathcal{R})$:

$$R\left(\hat{f}_M, L, \mathcal{Q}, \mathcal{R}\right) = \int_{q \in \mathcal{Q}} \int_{r \in \mathcal{R}} l\left(\hat{f}_M(q), r\right) P(q, r)\, dqdr \tag{2.15}$$

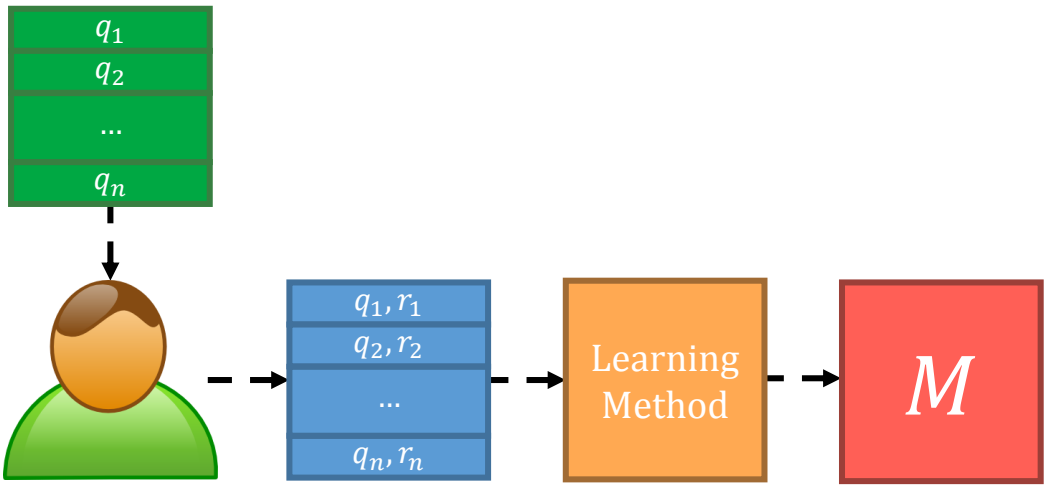
The conditions in which empirical risk is a good approximation for (2.15), known as *expected risk*, are outlined by VC-theory [144].

In contrast to batch gradient methods, *stochastic gradient* methods [110] perform updates using the loss function defined for a single unit of feedback (q_t, r_t):

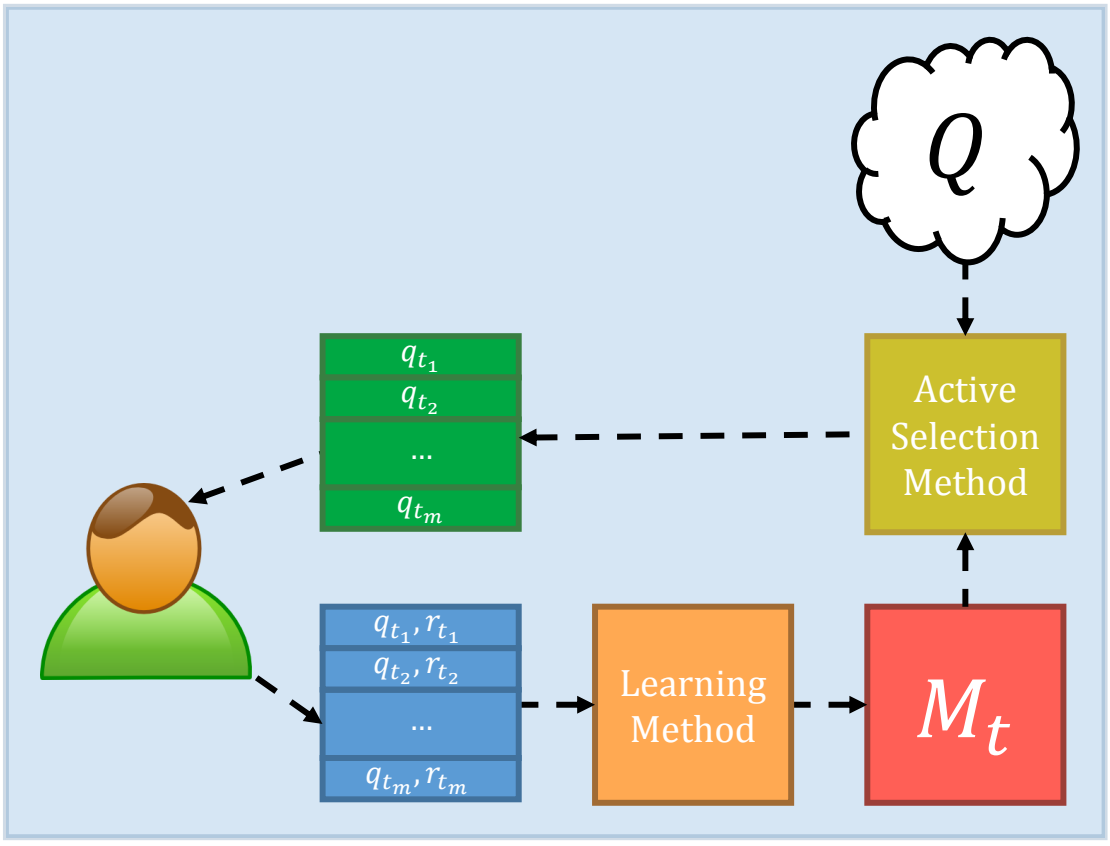
$$M_t \leftarrow M_{t-1} - \delta_t * \nabla l\left(\hat{f}_{M_{t-1}}(q_t), r_t\right) \tag{2.16}$$

For batch learning, single units of feedback are randomly chosen from the training set and used to update the model repeatedly until convergence. However, because updates are defined over single units of feedback, they can be applied as feedback is obtained in an online fashion. While this is a clear benefit over batch gradient optimization for the online learning setting, stochastic methods enjoy other benefits in general. By performing updates without reference to a particular training set, stochastic gradient methods can be viewed as updating a model by taking individual samples from the distribution $P(\mathcal{Q}, \mathcal{R})$. The average stochastic update, then, directly minimizes expected risk, not approximating it as with empirical risk. However, convergence to the minimum expected risk is highly dependent on the step size δ_t. In practice, the step size is a function of the number of steps taken. Specifically, $\delta_t \leftarrow O\left(\sqrt{t}\right)$ is a popular choice as it enjoys various convergence guarantees for objectives with different properties [15, 96, 3, 121, 7]. Part of these guarantees assume that some form of *averaging* takes place. Possibly the simplest way to perform this in practice is to revisit previously obtained feedback periodically by randomly sampling from the set of obtained feedback and updating with these samples as well as new feedback, as it is obtained. In short, one can achieve an averaging effect in practice by performing the update in (2.16) with feedback as it is obtained, but also randomly updating using samples from the pool of feedback obtained in previous steps.

Both batch and stochastic gradient methods can be applied to mini-batches of feedback, creating *mini-batch gradient descent* methods. At time t, the new mini-batch of feedback can be combined with all previous mini-batches to make a single train set on which batch methods can be run. Similarly, the mini-batch obtained at time t can be incorporated into the model by taking stochastic steps with respect to each individual unit of feedback within the mini-batch. Only recently has there been gradient methods developed specifically for the mini-batch learning setting [30, 131, 85].



(a) Random query selection



(b) Active query selection

Figure 6: Random versus active query selection

2.4.2 Random versus Active Learning

The batch/mini-batch/online setting effectively determines "how much" feedback is received at a time. These settings, however, do not describe "which" feedback is received. In most cases, which queries are posed to a human are determined before a model is learned. Because of this, most methods assume that feedback is sampled at *random* from the distribution $P(\mathcal{Q}, \mathcal{R})$. Figure 6a shows this in more detail. In batch learning, a set of predefined training queries are posed to a human, which she answers. This feedback is used to train a model. Note that the batch case is pictured, but this can easily applied to the online or mini-batch case where single or mini-batches of randomly-selected queries are iteratively posed to a human for responses.

However, the number of possible queries that can be posed to a human is high in many domains. Many of which contain redundant or unimportant information for a learning task. If the learner is able to choose which queries are "most informative" for learning from a pool of possible queries, then it could require less feedback to learn an accurate model, effectively increasing human efficiency. This is the motivation behind *pool-based active learning*. We show an example of pool-based active learning in Fig. 6b. Here, after an initial set of m queries are answered and the resulting feedback is used to train a model, an active selection method considers the current model as well as a pool of possible queries that can be posed to a human. With this information, the active selection method produces the next round of queries that can potentially be most informative in learning M_{t+1}. We say this is an example of pool-based active learning as there are many variations of this process. For instance, the active selection method may also consider which queries were already asked or the specific method that produced M_t to determine the next round of queries. Also, m can be set to one to more closely resemble the online learning case.

Where active selection methods primarily differ from one another is in the criteria they use to determine the informativeness of a query. Active selection strategies range from maintaining a committee of learned models and choosing the queries in which most models disagree in prediction [119, 33, 88], to choosing queries that maximally reduce error in expectation [112, 159, 54] to selecting queries that result in the biggest change in the model from the previous iteration in expectation [118]. The most popular active selection strategy is *uncertainty sampling* [84, 83, 136]. These methods score each query about how uncertain the current model is in its prediction of how

the human will respond to it. Consider the case where $\hat{f}_M$ has a probabilistic interpretation, namely, that it induces well-defined probabilities $P_{\hat{f}_M}(r|q)$ for all responses and queries in their respective domains. With such probabilities one can define various measures of how uncertain $\hat{f}_M$ is in a query q. Possibly the most well-known measure of uncertainty is *Shannon entropy* [124]:

$$H_{\hat{f}_M}(q) = -\sum_r P_{\hat{f}_M}(r|q) \log\left(P_{\hat{f}_M}(r|q)\right) \tag{2.17}$$

Here, the sum is over all r such that r is a valid response to query q. This assumes the likely case where there are a discrete number of valid responses r for a given query q. If responses are continuous, then the entropy of a query q is defined as:

$$H_{\hat{f}_M}(q) = -\int_r P_{\hat{f}_M}(r|q) \log\left(P_{\hat{f}_M}(r|q)\right) dr \tag{2.18}$$

Armed with such a powerful measure, active learning methods can be created for any models for which $P_{\hat{f}_M}(r|q)$ can be defined by simply scoring all queries in the pool according to (2.17), and choose the one(s) with the highest entropy given the current model. In practice, this can be difficult if there are a high number of queries or valid responses to queries. In these cases, more thoughtful methods than brute force scoring of all queries must be developed.

On a final note, pool-based active learning is only one of many active learning settings, and is reviewed here as a means to provide a basis for our active learning method in Chap. 4. For further review on other other active learning settings, as well as a more comprehensive survey of methods and their theoretical guarantees, see [117].

3.0 RELATED WORK

In this chapter, we build off of the concepts introduced in Chap. 2, and review specific previous works that are related to that which is presented in the remainder of this dissertation. We begin with a survey various methods in metric learning, highlighting those most relevant to this dissertation. Then, we transition to methods that learn kernels, namely those in Multiple Kernel Learning and Non-Parametric Kernel Learning. Finally, we discuss relevant work in online and active learning.

3.1 METRIC LEARNING

Metric learning methods learn a generalized Mahalanobis distance metric from feedback about objects. Different metric learning methods have been developed for different forms of information describing objects pair-wise similarity/dissimilarity assessments [156], relative pair-wise comparisons [116], and general linear constraints [34], just to name a few. Here, we review two methods that influenced the development of our metric learning method in Chap. 6. For more comprehensive review on metric learning methods and their applications, see [157, 75, 10].

3.1.1 Large Margin Nearest Neighbors

Large Margin Nearest Neighbor (LMNN) [151, 152, 153] metric learning, as the name implies, learns a metric from class labels for the purpose of nearest neighbor classification. LMNN distinguishes itself by being more efficient than its predecessors. Previous methods [156, 122, 52] used objectives that consider distances between all pairs of objects. The key observation that motivates LMNN is that for a kNN classifier to be accurate, only the closest objects to a query object needs to

be of the same class. With this in mind they formulated LMNN to be a semidefinite program (SDP) that learns a metric such that the k closest neighbors to every training object are of the same class. By limiting the scope of the optimization, training an LMNN metric is much faster in practice. Even with the simplified learning procedure, LMNN achieves state-of-the-art classification performance. For these reasons, LMNN has become a popular baseline in the metric learning community, and many extensions to it have since been developed [105, 101, 69].

In practice, LMNN learns $\mathbf{L}$ using the the factorized metric shown in (2.6) squared, instead of learning $\mathbf{M}$ as in (2.3). This effectively drops the PSD constraint in their optimization problem, which is computationally expensive to maintain. The main drawback is that their objective, which is convex in $\mathbf{M}$ is non-convex in $\mathbf{L}$. Indeed, this trade-off is one that many metric learning methods must face. Here though, the creators of LMNN show in their experiments that they are still able to achieve high classification accuracy despite solving a non-convex optimization problem.

To motivate their formulation, the authors of LMNN introduce the concepts of *target neighbors* and *impostors* for each object. Target neighbors to a training object are the k other training objects with the same label that are the closest to it in terms of Euclidean distance in the original feature space. Impostors are objects that do not have the same label that are closer to a training object than its target neighbors. The objective of LMNN contains two terms: a "pull" energy that pulls target neighbors toward each training object and a "push" energy that forces impostors away:

$$E_{pull}\left(\mathbf{L}\right) = \sum_{j \rightsquigarrow i} d^2_{\mathbf{L}}\left(\mathbf{x}_i, \mathbf{x}_j\right) \tag{3.1}$$

$$E_{push}\left(\mathbf{L}\right) = \sum_{i,j \rightsquigarrow i} \sum_{l} \left(1 - y_{il}\right) \left[1 + d^2_{\mathbf{L}}\left(\mathbf{x}_i, \mathbf{x}_j\right) - d^2_{\mathbf{L}}\left(\mathbf{x}_i, \mathbf{x}_l\right)\right]_+ \tag{3.2}$$

Here, $j \rightsquigarrow i$ indicates that $\mathbf{x}_j$ is a target neighbor of $\mathbf{x}_i$, y_{il} is 1 if $\mathbf{x}_i$ and $\mathbf{x}_l$ have the same class label (0 otherwise). In this case, $[\cdot]_+$ is known as the *hinge loss* function for which loss is strictly non-negative. LMNN minimizes the sum of the push and pull energy of all training objects via a gradient descent procedure. In doing so, the distance between each training object and its target neighbors is minimized (pull), and the distance between each training object and its impostors is increased until the impostors are farther away from the training point than its target neighbors (push) by a margin of one. Note that the set of impostors for an object may change as steps are taken towards a solution. Normally, LMNN would need to find the new set of impostors after every

gradient step. The authors remark that in practice, the set of impostors changes little from step to step. Because of this, they recompute the set of impostors after every couple of steps, reducing computation time significantly.

3.1.2 Metric Learning for Kernel Regression

There are decidedly fewer metric learning methods developed for regression than classification. However, Metric Learning for Kernel Regression (MLKR) [154] does, indeed, learn a metric from real-valued labels. The goal in MLKR is to learn a kernel regressor, where a predicted label $\hat{y}_i$ for an object $\mathbf{x}_i$ is defined as the weighted mean of the training objects:

$$\hat{y}_i = \frac{\sum_{j \neq i} y_j * k\left(\mathbf{x}_i, \mathbf{x}_j\right)}{\sum_{j \neq i} k\left(x_i, x_j\right)} \tag{3.3}$$

where y_j is the label for object $\mathbf{x}_j$. The sum in both the numerator and the denominator is for all training objects not including the one in which the label is predicted for. Essentially, an object's label is constructed by the labels of those most similar to itself. MLKR learns a Gaussian kernel function k, as in (2.11), where the Euclidean distance is instead a factorized Mahalanobis distance, as in (2.6), that is learned. Note, however, that the bandwidth parameter of the Gaussian kernel is simply a scaling of the distance, which is a function of $\mathbf{L}$. Because of this, the authors drop the bandwidth from the kernel and instead learn the scaling through $\mathbf{L}$.

Their optimization, which they solve via gradient descent, minimizes leave-one-out quadratic regression error over training objects:

$$E_{MLKR}\left(\mathbf{L}\right) = \sum_i \left(y_i - \hat{y}_i\right)^2 \tag{3.4}$$

Here, both the predicted and true labels are real-valued, thus error in this case is a function of how far off the predicted label is from the true label. MLKR is prohibitively computationally expensive for problems with many objects due to the fact that the kernel must be computed at each gradient step for all pairs of objects. However, the authors remark that they can take advantage of the fact that the Gaussian kernel decays rapidly towards zero. As a result, the kernel function need only be computed for a small subset of objects to closely approximate the exact solution.

3.2 KERNEL LEARNING

Mahalanobis distance metric learning methods, at their core, learn a linear transformation of objects in a feature space into a metric space. *Kernel-learning* methods learn models of similarity that can be more expressive than simple linear transformations. In this section, we discuss two kernel-learning techniques that act under different assumptions: Multiple Kernel Learning and Non-Parametric Kernel Learning. These two categories of kernel-learning methods show both the versatility of kernels and different settings in which they can be used to learn a model of similarity. We finish this section by surveying methods in a special case of Non-Parametic Kernel Learning called Relative Comparison Kernel Learning where kernels are learned using relative triplet feedback.

3.2.1 Multiple Kernel Learning

In practice, the success of a kernel method depends heavily on the choice of kernel used. If a linear kernel does not discriminate objects of different classes well, then a linear SVM will often perform poorly. Often in practice, a kernel is chosen through some form of validation. This involves applying a kernel function with a certain parameterization to training data, training a kernel method with the result, and evaluating the resulting model on a held-out validation set. After numerous kernel functions are evaluated, the one that performs the best on the validation set is used. Choosing a kernel in such a way is both time consuming and can lead to sub optimal performance. A alternate strategy is to consider a particular kernel function with a free parameter, such as the Gaussian kernel with a bandwidth, and learn the parameter through some optimally criteria [22, 23, 26]. These methods eliminate the need for manual validation, but separate the problem of finding the kernel and the application in which the kernel is used. Also, these methods tend to restrict themselves to a class of kernels with a particular form, while not considering other potential kernels.

Multiple Kernel Learning (MKL) methods [80, 8, 128, 106, 145, 53], on the other hand, attempt to solve this problem by learning an appropriate kernel while simultaneously learning the model in which it is to be used. MKL methods assume that there exist numerous predefined basis kernels $k_1, k_2, ..., k_A$ that can be applied to objects. These kernels can be entirely different kernel functions (e.g. linear kernel, polynomial kernel, Gaussian kernel, etc.), different parameterizations of kernel

functions (e.g. Gaussian kernels with different bandwidths), or kernel functions applied different subsets of the object feature space. The benefit MKL methods offer is that different combinations of a large number of basis kernels can represent a large class of kernel functions. MKL methods learn the "optimal" kernel from this class using the same criteria used to learn the model for which the kernel is used. For instance, if the kernel is meant to be used in a SVM classifier, an MKL method would learn both the parameters of an SVM while learning the optimal kernel. If the same task were to be accomplished by validation, it would require many iterations of validation.

In a sense, MKL methods eliminate problem of choosing the best single kernel, but create another one: What combination, and thus, what class of kernels should be learned? While various combinations have been proposed, the work discussed in this dissertation focuses on a *conic combination* of basis kernels, which is defined as:

$$\begin{aligned} k\left(x_i, x_j\right) &= \sum_{a=1}^{A} \mu_a k_a\left(x_i, x_j\right) \\ \text{s.t.} \quad & \boldsymbol{\mu} \in \mathbb{R}_+^A \end{aligned} \tag{3.5}$$

The parameter to be learned by an MKL method in this case is $\boldsymbol{\mu}$. Intuitively, kernels that are better suited for the task will have a higher setting of their corresponding weight μ_a. Further intuition can be gained by considering the resulting kernel mapping of the objects. Let $\phi_1, \phi_2, ..., \phi_A$ be the A feature mappings that correspond to the basis kernels $k_1, k_2, ..., k_A$. The conic combination of kernels defined in (3.5) induces the following mapping [115]:

$$\phi\left(x_i\right) = \left[\sqrt{\mu_1} * \phi_1\left(x_i\right), \sqrt{\mu_2} * \phi_2\left(x_i\right), ..., \sqrt{\mu_A} * \phi_A\left(x_i\right)\right] \tag{3.6}$$

This is simply the weighted concatenation of the inner product spaces induced by each of the basis kernels. Stated another way, by learning a conic combination, MKL methods implicitly learn an embedding of the objects in a feature space that is composed of a scaling of different features.

3.2.2 Non-Parametric Kernel Matrix Learning

For many methods, defining k is sufficient for kernelization and there is no need to explicitly define ϕ. Similarly, in scenarios where k only needs to be defined for a fixed set of objects, learning a kernel matrix $\mathbf{K}$ is sufficient. For instance, clustering is often performed over a fixed set of observed objects. Learning over a fixed set of objects is known as *transductive* learning, as opposed to *inductive* learning which learns a model that can be applied to a domain of objects. The benefit of inductive learning methods, such as Mahalanobis distance metric learning methods as well as many MKL methods, is that they can be used to perform inference on objects not explicitly trained on. Transductive methods cannot be applied to such out-of-sample objects, but because they limit themselves to a specific set of objects, tend to enjoy much more freedom in modeling assumptions.

This transductive learning setting is the one in which non-parametric kernel learning (NPKL) [78, 76, 63, 160] methods were created for. NPKL methods do not learn a kernel function k as many MKL methods do, but the elements of a kernel matrix $\mathbf{K}$ directly. While NPKL methods cannot learn a model of similarity for unobserved objects, they enjoy other benefits over techniques that learn a kernel function or a metric. MKL and metric learning methods are limited to a class of kernel functions or metrics that may not be able to model all desired relationships between objects. If the elements of the matrix $\mathbf{K}$ are learned directly, the only constraint is that the learned kernel matrix must be PSD. Indeed, learning a kernel matrix is equivalent to learning an embedding $\mathbf{X} \in \mathbb{R}^{n \times d}$ of the objects, where each row is an object embedded in a d-dimensional space, such that $\mathbf{K} = \mathbf{X}\mathbf{X}^T$. Thus, the only restriction placed on NPKL methods is that they must induce a d-dimensional embedding of objects in an inner product space. Also as a result of their relationship to embeddings, many NPKL methods can be defined as equivalent *embedding-learning* formulations.

3.2.2.1 Relative Comparison Kernel Learning Different NPKL methods are guided by different forms of information, including forms of feedback discussed in Sec. 2.3. Much of the subsequent work in this document revolves around developing NPKL methods. More specifically, in Chaps. 4 and 5 we study a special case of NPKL we call Relative Comparison Kernel Learning (RCKL), which uses relative triplet feedback to learn a kernel matrix. Let $\mathcal{T}$ be a set of triplet

Algorithm 1 A Prototypical Batch RCKL Learning Algorithm

Input: $\mathcal{T}$

1: $\mathbf{K}_0 \leftarrow \mathbf{I}$
2: **for** $i = 1, 2, 3, ...$ **until** convergence **do**
3: $\quad \mathbf{K}'_i \leftarrow \mathbf{K}_{i-1} - \delta_t \nabla L\left(\mathbf{K}_{i-1}, \mathcal{T}\right) + \lambda \mathbf{I}$
4: $\quad \mathbf{K}_i = \Pi_+\left(\mathbf{K}'_i\right)$
5: **end for**

responses, obtained from humans:

$$\mathcal{T} = \{(a, b, c) | x_a \text{ is more similar to } x_b \text{ than } x_c\} \tag{3.7}$$

The goal of RCKL methods is to learn a kernel $\mathbf{K}$, such that the relationships among objects given by the triplets are modeled by $\mathbf{K}$. More specifically:

$$\begin{aligned} \forall_{(a,b,c)\in\mathcal{T}}: \quad & d^2_{\mathbf{K}}(x_a, x_b) < d^2_{\mathbf{K}}(x_a, x_c) \\ \text{Where} \quad & d^2_{\mathbf{K}}(x_a, x_b) = \mathbf{K}^{aa} + \mathbf{K}^{bb} - 2\mathbf{K}^{ab} \end{aligned} \tag{3.8}$$

Much of the previous work in RCKL has focused on designing different loss functions that measure a kernel's ability to satisfy the constraints outlined in (3.8) for a given $\mathcal{T}$. Many RCKL methods attempt to learn a kernel by solving an optimization problem of the following form:

$$\begin{aligned} \min_{\mathbf{K}} \quad & L(\mathbf{K}, \mathcal{T}) + \lambda \text{trace}(\mathbf{K}) \\ \text{s.t.} \quad & \mathbf{K} \succeq 0, \end{aligned} \tag{3.9}$$

The first term, $L(\mathbf{K}, \mathcal{T})$, is a function of the loss that the objective incurs for $\mathbf{K}$ not satisfying triplets in $\mathcal{T}$, which typically can be written as a sum of loss functions over individual triplets:

$$L(\mathbf{K}, \mathcal{T}) = \sum_{t\in\mathcal{T}} l\left(\mathbf{K}, t\right) \tag{3.10}$$

The second term regularizes $\mathbf{K}$ by its trace. Here, the trace is used as a convex approximation of the non-convex rank function. The rank of $\mathbf{K}$ directly reflects the dimensionality of the embedding of the objects in $\mathcal{H}_{\mathbf{K}}$, the inner product space induced by $\mathbf{K}$. A low setting of the hyperparameter λ favors a more accurate embedding, while a high value prefers a lower rank kernel. The PSD

constraint ensures that $\mathbf{K}$ is a valid kernel matrix, and makes (3.9) an SDP over n^2 variables. For the remainder of this dissertation we will refer to (3.9) as traditional RCKL.

Most RCKL methods solve (3.9) via *projected gradient descent* in batch. A prototypical batch RCKL procedure is outlined in Alg. 1. At each iteration of this procedure, a gradient step is taken along in the descending direction of the objective (line 3). Then, the result of that step is projected onto the PSD cone on line 4 using the projection procedure Π_+. The computational bottleneck is in the PSD projection, which is commonly defined as follows:

$$\Pi_+ (\mathbf{K}) = \mathbf{V}_\mathbf{K} [\mathbf{\Lambda}_\mathbf{K}]_+ \mathbf{V}_\mathbf{K}^T \tag{3.11}$$

Here, $\mathbf{V}_\mathbf{K}$ is a matrix consisting of eigenvectors of $\mathbf{K}$ as rows. The matrix $\mathbf{\Lambda}_\mathbf{K}$ has the eigenvalues of $\mathbf{K}$ on the diagonal, and zeros elsewhere. Finally, $[\cdot]_+$ is taken element-wise, i.e $[\mathbf{\Lambda}_\mathbf{K}^{ii}]_+$ for $i = 1, ..., n$. Succinctly put, the projection operator Π_+ performs full eigendecomposition of $\mathbf{K}$, sets negative eigenvalues to zero, and then reconstructs $\mathbf{K}$ from the new eigenvalues and original eigenvectors. Without prior assumptions on the structure of the gradient in Alg. 1, any number of the eigenvalues of $\mathbf{K}_i'$ can be negative. Thus, full eigendecomposition is necessary for the PSD projection. Full eigendecomposition of a dense matrix is known to be a $O(n^3)$ operation [100].

Where many RCKL methods differ is in their choice for L. We consider three such methods in this work. First, *Generalized Non-Metric Multidimensional Scaling* (GNMDS) [4] uses hinge loss:

$$L_{\text{GNMDS}} (\mathbf{K}, \mathcal{T}) = \sum_{(a,b,c) \in \mathcal{T}} \left[d^2_\mathbf{K}(x_a, x_b) - d^2_\mathbf{K}(x_a, x_c) + 1 \right]_+ \tag{3.12}$$

Here, only triplets that are not satisfied by a margin of one effect the objective. Stochastic Triplet Embedding (STE) [143] proposes the following probability that a triplet is satisfied:

$$p^\mathbf{K}_{abc} = \frac{\exp(-d^2_\mathbf{K}(x_a, x_b))}{\exp(-d^2_\mathbf{K}(x_a, x_b)) + \exp(-d^2_\mathbf{K}(x_a, x_c))} \tag{3.13}$$

If this probability is high, then x_a is closer to x_b than it is to x_c. As such, they minimize the negative sum of the log-probabilities over all triplets:

$$L_{\text{STE}} (\mathbf{K}, \mathcal{T}) = -\sum_{(a,b,c) \in \mathcal{T}} \log(p^\mathbf{K}_{abc}) \tag{3.14}$$

Like STE, *Crowd Kernel Learning* (CKL) [132] defines a probability that a triplet is satisfied:

$$p^{\mathbf{K}}_{abc} = \frac{d^2_{\mathbf{K}}(x_a, x_c) + \mu}{d^2_{\mathbf{K}}(x_a, x_b) + d^2_{\mathbf{K}}(x_a, x_c) + 2\mu} \tag{3.15}$$

Where μ is a "uniqueness" parameter that ensures that no probability can be exactly one, even if the distance between two objects is zero. CKL differs from GNMDS and STE in a few key aspects. First, the creators of CKL choose to minimize the sum of the inverse probabilities defined in (3.15) for all obtained triplets. Their objective is then non-convex $\mathbf{K}$. Furthermore, the creators of CKL opt to constrain $\mathbf{K}$ to be a correlation matrix instead of regularizing it by its trace. This has a similar, but not equivalent regularization effect. This removes the last term in line 3 in Alg. 1, and changes the projection in line 4 to one that projects $\mathbf{K}'_i$ onto the set of PSD correlation matrices. Such a projection is even more expensive to perform in practice. The authors suggest a projection scheme that alternates between projecting onto the PSD cone and diagonal matrices [81]. For further comparison of all three of these RCKL methods, see [143].

RCKL methods, like other NPKL methods, can be formulated to learn object embeddings. In fact, the foundational work that introduced the problem of learning similarity from relative comparisons poses it as an embedding-learning problem [74]. As an example example, the CKL probabilities shown in (3.15) can be defined as a function of an embedding $\mathbf{X}$:

$$p^{\mathbf{X}}_{abc} = \frac{d^2_2(\mathbf{x}_a, \mathbf{x}_c) + \mu}{d^2_2(\mathbf{x}_a, \mathbf{x}_b) + d^2_2(\mathbf{x}_a, \mathbf{x}_c) + 2\mu} \tag{3.16}$$

In [132], the authors suggest to use this embedding-learning formulation as a means to avoid their costly alternating projection. For GNMDS and STE, formulating their respective loss functions in terms of an embedding makes them non-convex. Thus, the two different formulations of RCKL create a trade-off between computational efficiency and guaranteed optimality.

RCKL methods assume a particular model of similarity, namely a kernel matrix or an embedding. One may wonder how expressive this choice of model is. More specifically, one can ask the question: Can any set of triplet responses be represented by a kernel matrix? Appendix A of [91] proves the following theorem in response to this question:

Theorem 1 *For any non-conflicting set of triplet responses $\mathcal{T}$, there exists a embedding $\mathbf{X}$, or equivalently, a kernel $\mathbf{K}$ such that all triplet constraints defined in (3.8) induced by $\mathcal{T}$ are satisfied.*

They authors of [91] prove this by showing any set of non-conflicting triplets can be satisfied by an n-dimensional embedding, or, equivalently, a full-rank kernel matrix. This shows the power of RCKL methods. RCKL methods do not limit themselves to a parametric class of functions. Instead, they directly learn a representation of a given set of objects. Because of this, they have the expressiveness to produce models that faithfully reflect given triplet feedback. Note, that if there exists conflicts in a set of triplet responses, there does not exist an embedding of any dimensionality that can satisfy all responses in that set. Thus, if inconsistent feedback is given, some responses will be unsatisfied in a model learned by any RCKL method.

Because the dimensionality of a learned embedding (or equivalently the rank of a learned kernel matrix) determines the complexity of the models learned by RCKL methods, choosing the dimensionality (or how much to regularize $\mathbf{K}$ by) greatly affects the expressiveness of RCKL methods. If the dimensionality is chosen to be too small, then some triplet constraints may not be satisfied in the model. A simple example is the one depicted in Fig. 3f. The figure depicts a two-dimensional object embedding. If the dimensionality was chosen to be one, then the full set of triplet responses could not be satisfied. On the other hand, it can often be beneficial to assume a low dimensional object embedding. In [66] the authors prove the following theorem:

Theorem 2 *The number of triplet queries required to determine the embedding of n objects in d dimensions that satisfies all true triplet constraints is $\Omega\left(dn\log\left(n\right)\right)$.*

This theorem states that the number of triplet queries required to be asked in order to model all responses in an embedding is a function of the assumed dimensionality of the learned embedding. In other words, if d is small, then fewer queries are required to understand all object relationships than if d is large. In the worst case, if $d = n$, then this bound reduces to the one if comparison-based sort methods are used without knowledge of the complexity of object relationships.

3.3 ONLINE LEARNING

Learning either of the two models of similarity considered in this work, non-parametric kernel matrices and Mahalanobis distance metrics, require a PSD constraint. If the factored version of these

models are learned, common convex methods of solving for them become non-convex optimization problems. In the online learning case, the problems associated with these two alternatives are amplified. Consider a batch non-convex optimization procedure. Most standard optimization techniques guarantee locally, but not necessarily globally optimal solutions. If this is a concern, random restarts are often used and the optimization procedure is run multiple times to convergence using different starting positions. Now consider the online non-convex optimization case. Ideally, online learning methods globally minimize a loss function by using the most recent feedback and make little use of previously obtained feedback. Randomly restarting an online optimization means all previously obtained feedback must be used again. In doing so, the optimization acts much more like the batch procedure than a true online method. The other option is to optimize the convex versions of the objectives in an online fashion, and handle the PSD constraint. In stochastic gradient online learning, stochastic steps can be taken with respect to an objective, and then the solutions are projected onto the cone of PSD matrices. If done naively, this projection, like in the batch case, is an $O\left(n^3\right)$ operation that must be performed every time an update is made.

In Chap. 5 we introduce online stochastic optimization procedure for RCKL problems that maintains a PSD constraint while still being computationally efficient enough to be applied to large-scale problems. Recently, there has been work developing efficient, general-purpose, stochastic optimization procedures for SDPs [58, 87, 123]. These methods either formulate stochastic steps that either require no projection, or formulate projection procedures that are more efficient than than projecting naively. The method most similar to ours is low-rank stochastic gradient descent (LR-SGD) [25], where the authors assume stochastic updates that are rank-two:

$$M_t \leftarrow M_{t-1} - \delta_t * \left(\lambda_1 * \mathbf{v}_1\mathbf{v}_1^T + \lambda_2 * \mathbf{v}_2\mathbf{v}_2^T\right) \tag{3.17}$$

Here, $\mathbf{v}_1$ and $\mathbf{v}_2$ are the first and second eigenvectors of the gradient matrix ∇l, and λ_1 and λ_2 are their corresponding eigenvalues. With this assumption, one can bound the number of negative eigenvalues after the update to be at most two. Because of this, projection requires only finding the smallest two eigenvalues of a matrix after an update, which can be done in $O\left(n^2\right)$ time using iterative eigendecomposition methods. In our work, we show common RCKL loss functions have similar properties that allow us to bound the number of negative eigenvalues after an update. In addition, we are able to conservatively estimate when an update will result in a PSD matrix without

eigendecomposition. Because of this, we are able to create an online RCKL framework that is often able to skip the projection step entirely, resulting in constant time updates. When our framework is unsure whether the update will result in a non-PSD matrix, it is still able to project onto the PSD cone in $O\left(n^2\right)$ time.

Our framework is also inspired by *passive-aggressive* online learning [31], which was first introduced as a means to learn support vector machines in an online fashion. The intuition behind passive-aggressive learning is that in the online setting, single updates should not skew the model to reflect only the most recent feedback, but also shouldn't entirely ignore it. For problems that use a hinge-loss function as their objective, such as in SVMs, only instances that incur a positive loss affect the objective. In stochastic online learning, this means that only the units of feedback that do not satisfy a margin constraint require an update. As a result, an online algorithm could *passively* ignore feedback that is already sufficiently represented in the learned model. If a unit of feedback is obtained that does incur a loss, passive-aggressive learning methods *aggressively* step so that the new solution has a loss of zero for the most recently obtained unit of feedback.

A simple but illustrative example of passive-aggressive online learning is passive-aggressive hard-margin SVM classification. Here, class label feedback $(\mathbf{x}_t, y_t)$ is obtained in an online fashion. The goal is to find a weight vector $\mathbf{w}_t$ for an SVM classifier that is close to the solution in the previous step, but also correctly classifies feedback $(\mathbf{x}_t, y_t)$ by a margin of one. The solution at time t, then, can be found by solving the following problem:

$$\begin{aligned} \underset{\mathbf{w}_t}{\text{argmin}} \quad & \tfrac{1}{2}||\mathbf{w}_t - \mathbf{w}_{t-1}||_2^2 \\ \text{s.t.} \quad & \left[1 - y_t\left(\mathbf{w}_t \cdot \mathbf{x}_t\right)\right]_+ = 0 \end{aligned} \tag{3.18}$$

Here, the squared Euclidean norm between the weight vectors at times t and $t-1$ is used as a measure of distance between them. The constraint ensures that the SVM classifier parameterized by $\mathbf{w}_t$ classifies $\mathbf{x}_t$ correctly by a margin of one (it is assumed $y_t \in \{-1, 1\}$). The solution of (3.18) induces in the following online update:

$$\begin{aligned} \mathbf{w}_t \quad & \leftarrow \mathbf{w}_{t-1} + \delta_t y_t \mathbf{x}_t \\ \text{where} \quad & \delta_t = \frac{\left[1 - y_t\left(\mathbf{w}_t \cdot \mathbf{x}_t\right)\right]_+}{||\mathbf{x}_t||_2^2} \end{aligned} \tag{3.19}$$

Algorithm 2 Active Sequential Triplet Query Selection

Input: $\mathcal{X} = \{x_1, x_2, ..., x_n\}$

$\mathcal{R} \leftarrow \{\}$
for $a = 2, ..., n$ **do**
    for $b = 1, ..., a$ **do**
        for $c = 1, ..., a$ **do**
            if $q = (a, \{b, c\})$ is ambiguous **then**
                Ask source for response r to q
            else
                Infer r from $\mathcal{R}$
            end if
            $\mathcal{R} \leftarrow \mathcal{R} \cup r$
        end for
    end for
end for

Hence, if the SVM hinge-loss is zero for the feedback at time t, the second term in the update is zero, making the solution at time t the same as it was at time $t-1$. Otherwise, a step is taken with a learning rate that ensures the loss will be zero after the update.

3.4 ACTIVE LEARNING

In Chap. 4 we introduce a novel active RCKL method as a means to increase human efficiency in a task that uses an inherently human inefficient form of feedback. The goal of an active learning method for RCKL is to select triplet queries for which the most informative triplet responses will be given. In doing so, RCKL methods can learn a model of similarity that can reflect a complete notion of similarity that extends to responses yet to be obtained. To our knowledge only two active learning methods exist for the RCKL problems. In this section, we discuss both.

3.4.1 Active Sequential Triplet Query Selection

In [66] the authors consider the embedding-learning version of the RCKL problem. They aim to learn a d-dimensional embedding of n objects that models a large set of true triplet responses by

asking only a few triplet queries. Instead of thinking about an object embedding being in $\mathbb{R}^{n\times d}$, where each row is an embedded object, the authors equivalently think of an embedding as the concatenation of each d-dimensional representation of an object into a single vector $x \in \mathbb{R}^{nd}$. A query, then, divides $\mathbb{R}^{nd}$ into two regions: one in which the query is answered one way ((a, b, c)) and one where it is answered the other ((a, c, b)). The set of all possible queries over n objects divides $\mathbb{R}^{nd}$ into nd-cells via the intersection of the regions created by individual queries. A set of feedback induces a single nd-cell in which the feedback is satisfied. By this interpretation, the active RCKL problem can be viewed as finding the nd-cell for which every true response is satisfied by asking the fewest number of queries.

With this in mind, the authors formulate an *active sequential triplet query selection* algorithm, which we outline in Alg. 2. Each possible query over the n objects is considered by iterating over all pairs of object for a randomly selected head object. If a query is deemed *ambiguous* given the previous responses, then ask for a response from the source of feedback. Otherwise, infer the correct response to the query without asking the source of feedback and proceed to the next query. Ambiguity is determined as follows. Assume at iteration i of the algorithm responses $\mathcal{R}_{i-1} = \{r_1, r_2, ..., r_{i-1}\}$ have been either provided by a source of feedback or inferred by the algorithm. Assume without loss of generality that $q_i = (a, \{b, c\})$. The ambiguity finding procedure then determines if the nd-cells induced by $\mathcal{R}_i \cup (a, b, c)$ and $\mathcal{R}_i \cup (a, c, b)$ are feasible. If both are, then q_i is ambiguous. If only one of the two possible responses induces a feasible nd-cell, then the algorithm can infer it to be the correct response. In short, if the intersection of the nd-cell induced by previous responses and a possible response to a query is non-empty, then the response can be given and still be consistent with previous responses. The sequential algorithm forgoes asking queries for which one of the two possible responses would be inconsistent.

The motivation behind the sequential selection procedure comes from their proof of Thm. 2, where they consider an oracle sequentially providing the exact positions of each object in the embedding. The difference in their active algorithm is that instead of exact positions, a source of feedback give all responses with respect to each object as the head. Even so, experimentally, the authors show their sequential query algorithm is able to infer the responses to many queries if the objects can be embedded in $d << n$ and still reflect all feedback. The subroutine the authors use to determine if a query is ambiguous is essentially a procedure for finding an embedding. As

such, it is a non-convex optimization problem. Since only local optimality is guaranteed in solving such a problem, their subroutine could determine that a query is not ambiguous when it actually is, resulting in inferred responses that could be false. The authors note that in practice their solution for finding an embedding is most often correct, and random restarts can be used for increased accuracy.

3.4.2 Adaptive Crowd Kernel Learning

The other active triplet query selection method for RCKL is Adaptive Crowd Kernel Learning (A-CKL) [132] that is uses CKL as a basis for active selection. A-CKL considers the case where triplets are obtained in *rounds*, where a triplet query is actively chosen for each of the n objects as the head. As a result, A-CKL chooses mini-batches of n queries to be asked at a time. Consider, then, selecting a query with head x_a at round i of the algorithm. Let $\mathcal{R}_{i-1}^{a} = \{(a, b_1, c_1), (a, b_1, c_1), ..., (a, b_{i-1}, c_{i-1})\}$ be all response with head object x_a obtained before round i, and let $\mathbf{X}_{i-1}$ be the embedding learned by CKL from $\bigcup_{j=1}^{n} \mathcal{R}_{i-1}^{a_j}$. The creators of A-CKL define a posterior probability over the embeddings of x_a:

$$\tau_a(\mathbf{x}) = \pi(\mathbf{x}) * \prod_{a,b,c\,:\,(a,b,c)\in\mathcal{R}_{i-1}^{a}} p_{\mathbf{x},\mathbf{x}_b,\mathbf{x}_c}^{\mathbf{X}_{i-1}} \tag{3.20}$$

Where:

$$p_{\mathbf{x},\mathbf{x}_b,\mathbf{x}_c}^{\mathbf{X}} = \frac{d_2^2(\mathbf{x},\mathbf{x}_c)}{d_2^2(\mathbf{x},\mathbf{x}_b) + d_2^2(\mathbf{x},\mathbf{x}_c)} \tag{3.21}$$

and π is a prior distribution of object embeddings. The probability (3.21) is the same as (3.16) with the exceptions that the head object's embedding is not from the learned embedding $\mathbf{X}$, but explicitly input, and the uniqueness parameters are removed.

In A-CKL all possible triplet queries $(a, \{b, c\})$ are scored. The ones with the highest scores per head are chosen to be asked in that round. To this end, the creators of A-CKL define the probability that the source of feedback (a "crowd" in their application) will respond to $(a, \{b, c\})$ with (a, b, c) as:

$$p \propto \int_{\mathbf{x}\in\mathbb{R}^d} \tau_a(\mathbf{x}) * p_{\mathbf{x},\mathbf{x}_b,\mathbf{x}_c}^{\mathbf{X}_{i-1}} d\mathbf{x} \tag{3.22}$$

If the source of feedback were to respond with (a, b, c), the posterior probability would then become:

$$\tau_{a,b}(\mathbf{x}) = \tau_a(\mathbf{x}) * p_{\mathbf{x},\mathbf{x}_b,\mathbf{x}_c}^{\mathbf{X}_{i-1}} \tag{3.23}$$

To score this query, then, A-CKL uses *information gain* based on these distributions:

$$H\left(\tau_a\right) - \left(p * H\left(\tau_{a,b}\right) + (1-p) * H\left(\tau_{a,c}\right)\right) \tag{3.24}$$

Here, H is Shannon entropy:

$$H\left(\tau\right) = \int_{\mathbf{x}\in\mathbb{R}^d} \tau\left(\mathbf{x}\right) \log\left(\tau\left(\mathbf{x}\right)\right) d\mathbf{x} \tag{3.25}$$

The first term in (3.24) measures the uncertainty in the position of x_a before the next query is answered. The second term measures the expected uncertainty after $(a, \{b, c\})$ is answered. As such, (3.24) effectively measures the expected decrease in uncertainty of the position of x_a after $(a, \{b, c\})$ is answered. The query for which (3.24) is maximized is chosen for each object as head a at iteration i.

For A-CKL to be applied to real-world problems, many design decisions need to be made. First, the integrations for both (3.22) and (3.25) do not have analytical solutions, so they must be approximated using numerical methods such as Monte Carlo integration. Furthermore, an appropriate prior π must be chosen. The authors of [132] suggest taking the uniform prior over the current positions of all other objects. This effectively enforces an assumption that the positions of all objects besides the head objects is perfectly known, and that the head object is actually one of the other objects. This greatly simplifies the computation of the scores. Even with this assumption, scoring every query would be costly for a large n. In practice, a sample of all queries for a given head can be taken and selection can be made over this reduced pool.

4.0 SIMILARITY LEARNING FROM TRIPLET FEEDBACK WITH AUXILIARY INFORMATION

The RCKL methods discussed in Sec. 3.2.2.1 assume that no prior information is known about a domain of objects, and only the given set of triplet responses is used to learn a model of similarity over objects in that domain. Yet, in many scenarios, it is common to have *auxiliary information* associated with the objects that captures some latent aspect of the perceptual similarity space trying to be learned. In images for instance, humans must consider objects visually, and thus visual features can play a role in how humans make comparisons. Auxiliary visual features, such as those learned from convolutional neural networks on the images themselves [37], are readily available and can be used as a basis in learning similarity. If these features reflect how humans perceive the objects, then a learned similarity model based on features can capture a more complete notion of similarity from fewer responses.

In this chapter, we introduce techniques for incorporating such auxiliary information into RCKL methods. Ideally, an RCKL method that includes auxiliary features would identify general trends within the features that coincide with the perceptual similarity being conveyed through an obtained set of triplet responses. If it is able to do this, it can then include the relevant features into a learned model of similarity. Features that are irrelevant to the learning task could then be omitted from the model. Doing so would enable RCKL methods to learn more general models from fewer triplet responses by leveraging the trends in the auxiliary information that generalize to unseen triplet responses. In other words, by leveraging auxiliary information, we can increase the human efficiency of RCKL methods by supplementing triplet feedback with readily available prior information that describes the relationship among objects in a domain.

This chapter is broken down into two main sections. In Sec. 4.1, we begin by outlining how traditional RCKL methods that learn non-parametric kernel matrices can be modified to

include auxiliary information without drastically increasing their time complexity, and maintaining the convexity of their optimization problems. We then compare these new RCKL methods to similar methods in metric learning, and highlight the benefits of learning within our framework, as opposed to learning a metric. Finally, we experimentally evaluate RCKL methods that use auxiliary information to ones that do not, as well as similar metric learning methods. In Sec. 4.2 we focus on the embedding-learning formulation of Crowd Kernel Learning (CKL), and show how auxiliary information can be included in the models of similarity it produces. This formulation is probabilistic, enabling the use of information-theoretic notions of uncertainty in the embedding of objects. Because of this, we are able to formulate an active learning technique that selects triplet queries that have the potential to reduce uncertainty in the learned embedding the most. By doing so, we can further increase the human efficiency of learning a model of similarity from triplet feedback.

4.1 LEARNING A NON-PARAMETRIC KERNEL WITH AUXILIARY INFORMATION

In general, if there are few triplet responses $\mathcal{T}$ relative to the number of objects in question n, there are many different RCKL solutions. Without using information regarding how the objects relate other than $\mathcal{T}$, traditional RCKL methods may not generalize well to the many unobtained triplet responses. This problem is present in any supervised learning setting: without sufficient training data, learned models may not be general enough to model an acceptable number of object relationships. However, as noted in previous chapters, triplet feedback is inherently human inefficient, meaning this problem is intensified and many responses may be needed to completely define object relationships.

Fortunately, objects can often be described by features drawn from data. Let $\mathbf{x}_i \in \mathbb{R}^d$ be a d-dimensional feature vector for object $x_i \in \mathcal{X}$. Each feature represents a different characteristic that describes the object. For instance, if $\mathcal{X}$ is clothing items, then $\mathbf{x}_i$ can have features describing the color, size, and material. To effectively use this form of auxiliary information, an RCKL method should include features that in some sense reflect the same notion of similarity being conveyed in $\mathcal{T}$. Features that do not model $\mathcal{T}$ well, should not be included in the model. In the subsequent sections,

we outline how this form of auxiliary information can be included into RCKL methods.[1]

4.1.1 Multiple Kernel RCKL

Using features, one can construct $A \in \mathbb{Z}^+$ auxiliary kernels $\mathbf{K}_1, ..., \mathbf{K}_A \in \mathbb{R}^{n \times n}$ using standard kernel functions to model the relationship among objects. If one or more auxiliary kernels satisfy many triplets in $\mathcal{T}$, they may represent factors that influence how some of the unobtained triplets would have been answered. We wish to identify which of these predefined auxiliary kernels, built from data, model trends in $\mathcal{T}$, and then combine them in a way to satisfy triplets in $\mathcal{T}$. An approach to combine multiple different kernels popularized by MKL methods is through a weighted sum:

$$\mathbf{K}' = \sum_{a=1}^{A} \mu_a \mathbf{K}_a \quad \boldsymbol{\mu} \in \mathbb{R}^A_{\geq 0} \tag{4.1}$$

Restricting the domain of $\boldsymbol{\mu}$ as such makes $\mathbf{K}'$ a conic combination of kernels. By construction each $\mathbf{K}_a$ is PSD. Thus, (4.1) is also PSD [115]. $\mathbf{K}'$ induces the mapping $\boldsymbol{\Phi}_{\mathbf{K}'} : \mathcal{X} \to \mathbb{R}^D$ [53]:

$$\boldsymbol{\Phi}_{\mathbf{K}'}(x_i) = [\sqrt{\mu_1}\boldsymbol{\Phi}_1(x_i), ..., \sqrt{\mu_A}\boldsymbol{\Phi}_A(x_i)] \tag{4.2}$$

Here $\boldsymbol{\Phi}_a : \mathcal{X} \to \mathbb{R}^{d_a}$ is a mapping from an object into the inner product space induced by $\mathbf{K}_a \in \mathbb{R}^{n \times n}$, and $D = \sum_{a=1}^{A} d_a$. In short, $\mathbf{K}'$ induces a mapping of the objects into a space defined by the weighted concatenation of the individual feature spaces. Consider, then, the following optimization:

$$\begin{aligned} \min_{\boldsymbol{\mu}} \quad & L(\mathbf{K}', \mathcal{T}) + \lambda \|\boldsymbol{\mu}\|_1 \\ \text{s.t.} \quad & \boldsymbol{\mu} \geq 0 \end{aligned} \tag{4.3}$$

Here, L is a loss function indicating how poorly $\mathbf{K}'$ models $\mathcal{T}$. By learning the weight vector $\boldsymbol{\mu}$ through minimizing the loss L, (4.3) scales the individual concatenated feature spaces to emphasize the auxiliary kernels that model the object relationships in $\mathcal{T}$ well, and reduce the influence of those that do not.

Since the auxiliary kernels are fixed, regularizing them by their traces has no effect on their rank nor the rank of $\mathbf{K}'$. Instead, we choose to regularize $\boldsymbol{\mu}$ by its ℓ_1-norm, a technique first made popular for its use in the Least Absolute Shrinkage and Selection Operator (LASSO) [134]. For a

[1] The material presented in this section was originally published as [62].

proper setting of the hyperparameter λ, this has the effect of eliminating the contribution of kernels that do not help in reducing the error by forcing their corresponding weights to be exactly zero. Because of its relationship to multiple kernel learning, we call this formulation Multiple Kernel RCKL (RCKL-MKL). Because RCKL-MKL learns only $\boldsymbol{\mu}$, it is linear program over A variables.

By limiting the optimization to only a conic combination of the predefined auxiliary kernels, RCKL-MKL does not necessarily produce a kernel that satisfies any responses in $\mathcal{T}$. To capture the potential generalization power of using auxiliary information while retaining the ability to satisfy responses in $\mathcal{T}$, we propose to learn a combination of the auxiliary kernels and $\mathbf{K}_0$, a kernel similar to the one in traditional RCKL whose elements are learned directly. By doing this, we force traditional RCKL methods to prefer solutions similar to the auxiliary kernels, which could satisfy unobtained triplets. We call this hybrid approach Relative Comparison Kernel Learning with Auxiliary Kernels (RCKL-AK).

4.1.2 RCKL with Auxiliary Kernels

RCKL-AK learns the following kernel combination:

$$\mathbf{K}'' = \mathbf{K}_0 + \sum_{a=1}^{A} \mu_a \mathbf{K}_a \quad \boldsymbol{\mu} \in \mathbb{R}^A_{\geq 0},\ \mathbf{K}_0 \succeq 0 \tag{4.4}$$

(4.4) is a conic combination of kernel matrices that induces the mapping:

$$\boldsymbol{\Phi}_{\mathbf{K}''}(x_i) = [\boldsymbol{\Phi}_0(x_i), \sqrt{\mu_1}\boldsymbol{\Phi}_1(x_i), ..., \sqrt{\mu_A}\boldsymbol{\Phi}_A(x_i)] \tag{4.5}$$

The intuition behind this combination is that auxiliary kernels that satisfy many triplet responses are emphasized by weighing them more, and $\mathbf{K}_0$, which is learned directly, can satisfy the responses that cannot be satisfied by the conic combination of the auxiliary kernels. Consider, again, the example of learning similarity over clothing items. A human may compare clothes by characteristics such as color, size, and material, which can be represented by features used to build the auxiliary kernels. However, other factors may influence how a person compares clothes, such as designer or pattern, which may be omitted from the auxiliary kernels. In addition, the person providing feedback may have a personal sense of style that is impossible to be gained from features alone. $\mathbf{K}_0$,

and thus features induced by the mapping $\mathbf{\Phi}_0$, is learned to model factors a person uses to compare clothes that are omitted from the auxiliary kernels or cannot be modeled by extracted features.

Using (4.4), we propose to solve the following optimization problem:

$$\begin{aligned} \min_{\mathbf{K}_0, \boldsymbol{\mu}} \quad & L(\mathbf{K}'', \mathcal{T}) + \lambda_1 \text{trace}(\mathbf{K}_0) + \lambda_2 \|\boldsymbol{\mu}\|_1 \\ \text{s.t.} \quad & \mathbf{K}_0 \succeq 0, \ \boldsymbol{\mu} \geq 0 \end{aligned} \tag{4.6}$$

Here, the objective has two regularization terms: trace regulation on $\mathbf{K}_0$, and ℓ_1-norm regularization on $\boldsymbol{\mu}$. Increasing λ_1 limits the expressiveness of $\mathbf{K}_0$ by reducing its rank, while increasing λ_2 reduces the influence of the auxiliary kernels by forcing the elements of $\boldsymbol{\mu}$ towards zero. Thus, λ_1 and λ_2 represent a trade-off between finding a kernel that is more influenced by $\mathbf{K}_0$ and one more influenced by the auxiliary kernels. Like traditional RCKL, RCKL-AK is an SDP, but with $n^2 + A$ optimization variables. For practical A, RCKL-AK can be solved with minimal additional computational overhead to traditional RCKL methods, as the computational bottleneck is in projecting $\mathbf{K}_0$ onto the PSD cone.

One desirable property of (4.6) is that under certain conditions, it is a convex optimization:

Proposition 1 *If L is convex in both $\mathbf{K}_0$ and $\boldsymbol{\mu}$, then (4.6) is a convex optimization problem.*

Proposition 1 is proven in Sec. A.1. While Prop. 1 may seem simple, it allows us to leverage traditional RCKL methods that contain error functions that are convex in $\mathbf{K}_0$ and $\boldsymbol{\mu}$ in order to find globally optimal solutions using convex optimization techniques. For instance, if we use the STE loss function in our framework, it becomes:

$$L_{\text{STE}}(\mathbf{K}'', \mathcal{T}) = -\sum_{(a,b,c)\in\mathcal{T}} \log(p_{abc}^{\mathbf{K}''}) \tag{4.7}$$

Where, $p_{abc}^{\mathbf{K}}$ is defined in (3.13). We call the resulting method STE-AK. With this we can state:

Proposition 2 (4.7) *is convex in both $\mathbf{K}_0$ and $\boldsymbol{\mu}$*

Proposition 2 is proven in A.2. By Props. 1 and 2, STE-AK is a convex optimization problem. Another option is the GNMDS loss function. When used in the RCKL-AK framework it becomes:

$$L_{\text{GNMDS}}(\mathbf{K}'', \mathcal{T}) = \sum_{(a,b,c)\in\mathcal{T}} \left[d_{\mathbf{K}''}^2(x_a, x_b) - d_{\mathbf{K}''}^2(x_a, x_c) + 1 \right]_+ \tag{4.8}$$

Algorithm 3 RCKL-AK Projected Gradient Descent

Input:

$\mathcal{X} = \{x_1, ..., x_n\}$,
$\mathcal{T} = \{(a, b, c) \,|x_a \text{ is more similar to } x_b \text{ than } x_c\}$,
$\mathbf{K}_1, ..., \mathbf{K}_A \in \mathbb{R}^{n\times n}$, $\lambda_1 \in \mathbb{R}^+$, $\lambda_2 \in \mathbb{R}^+$, $\delta \in \mathbb{R}^+$

Output:

$\mathbf{K}'' \in \mathbb{R}^{n\times n}$

1: $i \leftarrow 0$
2: $\mathbf{K}_0^0 \leftarrow \mathbf{I}^{n\times n}$
3: $\mu_1^0, ..., \mu_A^0 \leftarrow \frac{1}{A}$
4: $\mathbf{K}'' \leftarrow \mathbf{K}_0^0 + \sum_{a=1}^{A} \mu_a^0 \mathbf{K}_a$
5: **repeat**
6: $\quad \mathbf{K}_0^{i+1} \leftarrow \mathbf{K}_0^i - \delta * \left(\nabla_{\mathbf{K}_0^i} L(\mathbf{K}'', \mathcal{T}\right) + \lambda_1 * \mathbf{I}^{n\times n})$
7: $\quad \boldsymbol{\mu}^{i+1} \leftarrow \boldsymbol{\mu}^i - \delta * \left(\nabla_{\boldsymbol{\mu}^i} L(\mathbf{K}'', \mathcal{T}) + \boldsymbol{\lambda}_2\right)$
8: $\quad \mathbf{K}_0^{i+1} \leftarrow \Pi_+ \left(\mathbf{K}_0^{i+1}\right)$
9: $\quad \boldsymbol{\mu}^{i+1} \leftarrow \Pi_{\geq 0} \left(\boldsymbol{\mu}^{i+1}\right)$
10: $\quad \mathbf{K}'' \leftarrow \mathbf{K}_0^{i+1} + \sum_{a=1}^{A} \mu_a^{i+1} \mathbf{K}_a$
11: $\quad i \leftarrow i + 1$
12: **until** convergence

We call our method with this error function GNMDS-AK, which is also a convex optimization problem, due to Prop. 1 and the following:

Proposition 3 (4.8) *is convex in both* $\mathbf{K}_0$ *and* $\boldsymbol{\mu}$.

4.1.2.1 Projected Gradient Descent for RCKL-AK Methods

We propose to solve methods created within our framework via the projected gradient descent algorithm outlined in Alg. 3. After initialization, the algorithm repeats the following steps until convergence:

1. **Line 6:** Take a gradient step for $\mathbf{K}_0$ (trace regularization included)
2. **Line 7:** Take a gradient step for $\boldsymbol{\mu}$ (ℓ_1-norm regularization included)
3. **Line 8:** Project $\mathbf{K}_0$ onto the positive semidefinite cone
4. **Line 9:** Project the elements of $\boldsymbol{\mu}$ to be non-negative
5. **Line 10:** Update $\mathbf{K}''$

Projection is onto the PSD cone is performed by (3.11), as is done in traditional RCKL methods. Projection of the elements of $\boldsymbol{\mu}$ to be non-negative is simply done by assigning all negative elements to be zero. The ℓ_1-norm regularization in Alg. 3 is performed by adding $\boldsymbol{\lambda}_2 = \lambda_2 * \mathbf{1}^A$ to the gradient (Line 7). Since $\boldsymbol{\mu}$ is constrained to be non-negative, the subgradient of the ℓ_1-norm function needs only to be over the non-negative orthant, thus $\boldsymbol{\lambda}_2$ is an acceptable subgradient. Moreover, since we then project the elements of $\boldsymbol{\mu}$ to be non-negative, we get the desired effect of the ℓ_1-norm regularization: the reduction of some elements to be exactly zero.

4.1.3 Relationship to Metric Learning

Relative comparisons have also been considered metric learning [116, 34, 65]. Most relevant to the work in this chapter are two recent methods that learn a Mahalanobis distance metric with multiple kernels: Metric Learning with Multiple Kernels (ML-MKL) [149] and Multiple Kernel Partial Order Embedding (MKPOE) [91], the latter focusing exclusively on relative distance constraints similar to those considered in RCKL. The kernel learned by RCKL-AK induces a mapping that is fundamentally different than those learned by these metric learning techniques. To show this, consider the distance metric shown in Sec. 4.2 of [91] and Equation 5 of [149]:

$$d^2_{\mathbf{M},\boldsymbol{\mu}}(x_i, x_j) = \sum_{a=1}^{A} \left(\mathbf{K}^i_a - \mathbf{K}^j_a\right) \mu_a \mathbf{M} \sum_{b=1}^{A} \mu_b \left(\mathbf{K}^i_b - \mathbf{K}^j_b\right)^T \tag{4.9}$$

This can be rewritten in terms of the feature mappings induced by each of the auxiliary kernels:

$$d^2_{\mathbf{M},\boldsymbol{\mu}}(x_i, x_j) = \left(\boldsymbol{\Phi}_{\boldsymbol{\mu}}(x_i) - \boldsymbol{\Phi}_{\boldsymbol{\mu}}(x_j)\right) \boldsymbol{\Phi}_{\boldsymbol{\mu}}(\mathbf{X}) \, \mathbf{M} \boldsymbol{\Phi}_{\boldsymbol{\mu}}(\mathbf{X})^T \left(\boldsymbol{\Phi}_{\boldsymbol{\mu}}(x_i) - \boldsymbol{\Phi}_{\boldsymbol{\mu}}(x_j)\right)^T \tag{4.10}$$

Where:

$$\boldsymbol{\Phi}_{\boldsymbol{\mu}}(x) = [\sqrt{\mu_1}\boldsymbol{\Phi}_1(x), ..., \sqrt{\mu_A}\boldsymbol{\Phi}_A(x)] \tag{4.11}$$

$\mathbf{X} = \left[\mathbf{x}_1^T, \mathbf{x}_2^T, ..., \mathbf{x}_n^T\right]^T$, and $\boldsymbol{\Phi}_{\boldsymbol{\mu}}(\mathbf{X}) = \left[\boldsymbol{\Phi}_{\boldsymbol{\mu}}(\mathbf{x}_1)^T, \boldsymbol{\Phi}_{\boldsymbol{\mu}}(\mathbf{x}_2)^T, ..., \boldsymbol{\Phi}_{\boldsymbol{\mu}}(\mathbf{x}_n)^T\right]^T$. By factoring the matrix $\mathbf{M} = \mathbf{L}\mathbf{L}^T$ and defining $\boldsymbol{\Omega} = \boldsymbol{\Phi}_{\boldsymbol{\mu}}(\mathbf{X})\,\mathbf{L}$ we can distribute $\boldsymbol{\Omega}$ through (4.10), similar to what is done in (2.6):

$$\begin{aligned} d^2_{\mathbf{M},\boldsymbol{\mu}}(x_i, x_j) &= \left(\boldsymbol{\Phi}_{\boldsymbol{\mu}}(x_i) - \boldsymbol{\Phi}_{\boldsymbol{\mu}}(x_j)\right) \boldsymbol{\Omega}\boldsymbol{\Omega}^T \left(\boldsymbol{\Phi}_{\boldsymbol{\mu}}(x_i) - \boldsymbol{\Phi}_{\boldsymbol{\mu}}(x_j)\right)^T \\ &= \left(\boldsymbol{\Phi}_{\boldsymbol{\mu}}(x_i)\,\boldsymbol{\Omega} - \boldsymbol{\Phi}_{\boldsymbol{\mu}}(x_j)\,\boldsymbol{\Omega}\right) \left(\boldsymbol{\Phi}_{\boldsymbol{\mu}}(x_i)\,\boldsymbol{\Omega} - \boldsymbol{\Phi}_{\boldsymbol{\mu}}(x_j)\,\boldsymbol{\Omega}\right)^T \\ &= d^2_2\left(\boldsymbol{\Phi}_{\boldsymbol{\mu}}(x_i)\,\boldsymbol{\Omega}, \boldsymbol{\Phi}_{\boldsymbol{\mu}}(x_j)\,\boldsymbol{\Omega}\right) \end{aligned} \tag{4.12}$$

In this form and by (4.11), we can see that learning $\mathbf{M}$ in (4.9) is equivalent to learning a squared Euclidean distance of points transformed by the following linear transformation:

$$\mathbf{\Phi}_{\boldsymbol{\mu},\mathbf{\Omega}}(x) = \mathbf{\Phi}_{\boldsymbol{\mu}}(x)\,\mathbf{\Omega} = [\sqrt{\mu_1}\mathbf{\Phi}_1(x), ..., \sqrt{\mu_A}\mathbf{\Phi}_A(x)]\,\mathbf{\Omega} \tag{4.13}$$

Here $\mathbf{\Omega} \in \mathbb{R}^{mxD}$ produces a new feature space by transforming the feature spaces induced by the auxiliary kernels. Without $\mathbf{\Omega}$, (4.13) learns a mapping similar to (4.2). The matrix $\mathbf{\Omega}$ plays a role similar to the one $\mathbf{K}_0$ plays in RCKL-AK: it is learned to satisfy triplets that the auxiliary kernels alone cannot. Instead of linearly transforming the auxiliary kernel feature spaces, RCKL-AK implicitly learns new features that are concatenated onto the concatenated auxiliary kernel feature spaces. In doing so, RCKL-AK is not limited to a class of linear transformations of the original features, but effectively learns additional features directly.

In both works, the authors propose non-convex optimizations to solve for their metrics, as well as, different convex relaxations. A critical issue with the convex solutions is that they employ SDPs over $n^2 * A$ (MKPOE-Full) and $n^2 * A^2$ (NR-ML-MKL) optimization variables, respectively. For moderately sized problems these methods are impractical. To resolve this issue, [91] proposes a method that imposes diagonal structure on the learned metric, reducing the number of optimization variables to $n * A$ (MKPOE-Diag), but in the process, greatly limits the structure of the metric. RCKL-AK is a convex SDP with $n^2 + A$ optimization variables that does not impose strict structure on the learned kernel. Unfortunately, by learning the unique kernel $\mathbf{K}_0$ directly and not the mapping $\mathbf{\Phi}_0$ or a generating function of $\mathbf{K}''$, our method cannot be directly applied to out-of-sample objects. However, the auxiliary kernel portion of models produced by RCKL-AK can be used to partially incorporate new objects, and the model can be updated if feedback over new objects is obtained.

4.1.4 Experiments

In order to show that RCKL-AK can learn kernels that generalize well from few obtained triplet responses, we perform two experiments: one using synthetic data, and one using real-world data. More specifically, in both experiments, we train STE and GNMDS in their traditional RCKL formulations (RCKL-T), but also in Multiple Kernel Learning RCKL (RCKL-MKL), and RCKL with Auxiliary Kernels (RCKL-AK) variants. In addition, we evaluate non-convex and convex

variants of MKPOE. We wish to see how the number of obtained triplet responses to train each model affect the ability of the models to generalize. As such, our experiments test each method's ability to learn models that satisfy held-out test triplet responses as a function of an increasing number of training responses. More specifically, we evaluate a model M with respect to a set of triplet responses $\mathcal{T}$ using *triplet query prediction error* (TQPE):

$$\text{TQPE}\left(M, \mathcal{T}\right) = \frac{\left|\left\{t \in \mathcal{T} : t \text{ is satisfied by } M\right\}\right|}{\left|\mathcal{T}\right|} \tag{4.14}$$

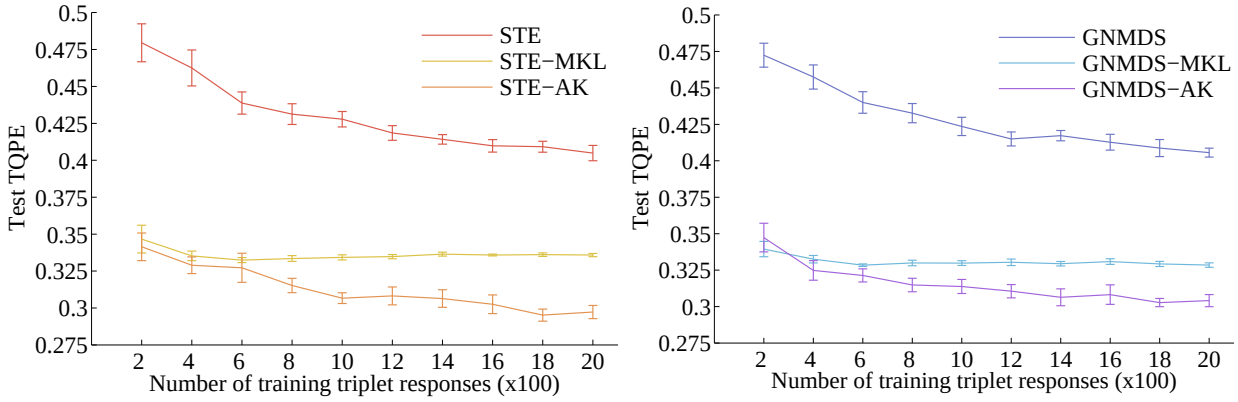
In short, TQPE measures the number of responses in a set that are satisfied by a model, normalized by the total number of responses in that set. For the kernel-learning methods, (3.8) defines when a kernel satisfies a triplet response. For the MKPOE methods, we consider a triplet (a, b, c) to be satisfied if $d_{\mathbf{M}}\left(x_a, x_b\right) < d_{\mathbf{M}}\left(x_a, x_c\right)$, where $d_{\mathbf{M}}$ is the distance function defined by the metric. The STE and GNMDS implementations used are from [143], which are made publicly available on the authors' websites. The MKL and AK versions were extended from these. MKPOE implementations were provided to us by their original authors. All auxiliary kernels are normalized to unit trace, and all hyperparameters were validated via line or grid search using validation sets.

4.1.4.1 Synthetic Data To generate synthetic data we began by randomly generating 100 points in seven, independent, two-dimensional feature spaces where both dimensions were over the interval $[0, 1]$. Then, we created seven linear kernels, $\mathbf{K}_0, ..., \mathbf{K}_6$ from these seven spaces. We combined four of the seven kernels:

$$\mathbf{K}^* = \frac{1}{2}\mathbf{K}_0 + \frac{1}{4}\mathbf{K}_1 + \frac{1}{6}\mathbf{K}_2 + \frac{1}{12}\mathbf{K}_3 \tag{4.15}$$

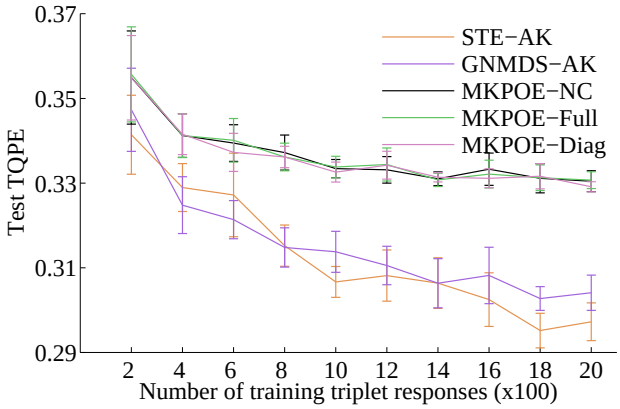
We then used $\mathbf{K}^*$ as the ground truth to answer all possible triplet queries over the 100 objects. Following the experimental setup in [132], we divided the resulting triplet responses into 100 response "rounds". A round is a set of responses where each object appears once as the head a being compared to randomly chosen objects b and c. From the pool of rounds, 20 were chosen to be the training set, 10 were chosen to be the validation set, and the remaining rounds were the test set. This was repeated ten times to create ten different trials.

Next, each object in all seven feature spaces was perturbed with randomly generated Gaussian noise. From these new spaces, we created seven new linear kernels $\hat{\mathbf{K}}_0, ..., \hat{\mathbf{K}}_6$, of which $\hat{\mathbf{K}}_1, ..., \hat{\mathbf{K}}_6$



(a) STE Methods

(b) GNMDS Methods

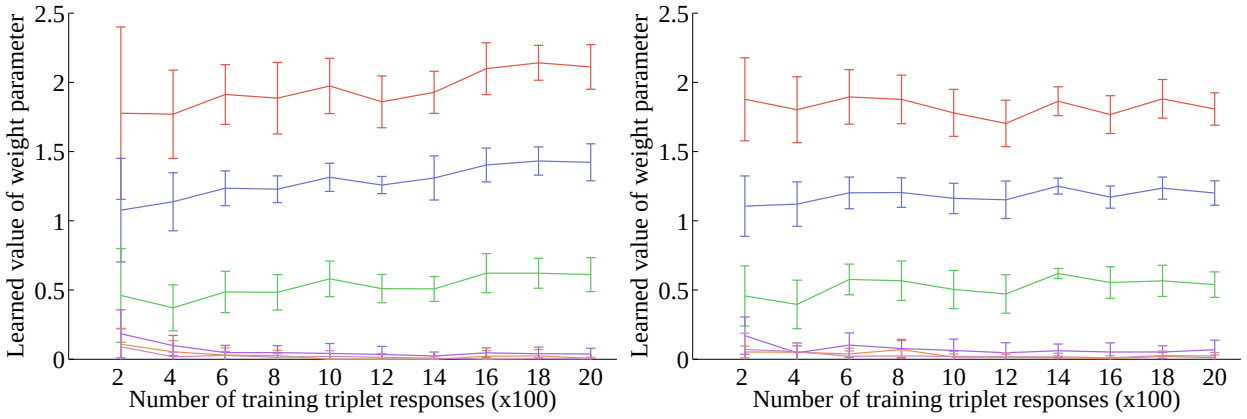


(c) MKPOE and AK Methods

Figure 7: RCKL-AK synthetic data experiments: Test TQPE vs. number of training responses (ten trials, 95% CI)

were used as the auxiliary kernels in the experiment. Here, $\hat{\mathbf{K}}_1, ..., \hat{\mathbf{K}}_3$ are kernels that represent attributes that influence how the ground truth makes comparisons. $\hat{\mathbf{K}}_4, ..., \hat{\mathbf{K}}_6$ contain information that is not considered when making comparisons, and $\mathbf{K}_0$ represents intuition about the objects that was not or cannot be input as an auxiliary kernel.

Intuitively, the models produced by each method should generalize better as more responses are used in training. To show this, and to compare the methods to one another, we performed the following experiment. For each trial, the 20 training rounds are divided into ten subsets, each



(a) STE-AK (b) GNMDS-AK

Figure 8: RCKL-AK synthetic data experiments: mean values of weight parameters (μ_1 (red), μ_2 (blue), μ_3 (green), μ_4 (teal), μ_5 (pink), μ_6 (purple))

containing two training rounds. Starting with one of the subsets, each model is trained, setting the hyperparameters through cross-validation on the validation set, and evaluated on the test set. Then, another subset is added to the training set. We repeat this process until all ten subsets are included. We evaluate all methods every time the training set increases in size. For all of the following figures, error bars represent a 95% confidence interval.

Discussion: Figure 7 shows the mean test TQPE versus the number of triplet responses in the training set for all ten trials and all methods tested. Figures 7a and 7b show the performance of the STE and GNMDS methods, respectively. Both RCKL-MKL methods improve performance initially, but achieve their approximate peak performance at around 600 training responses and fail to improve as more were added. This supports the claim that RCKL-MKL is overly limited by only being able to combine auxiliary kernels. Both traditional RCKL methods perform worse than all other methods. Without the side information provided by the auxiliary kernels, they cannot generalize to test responses given few training responses.

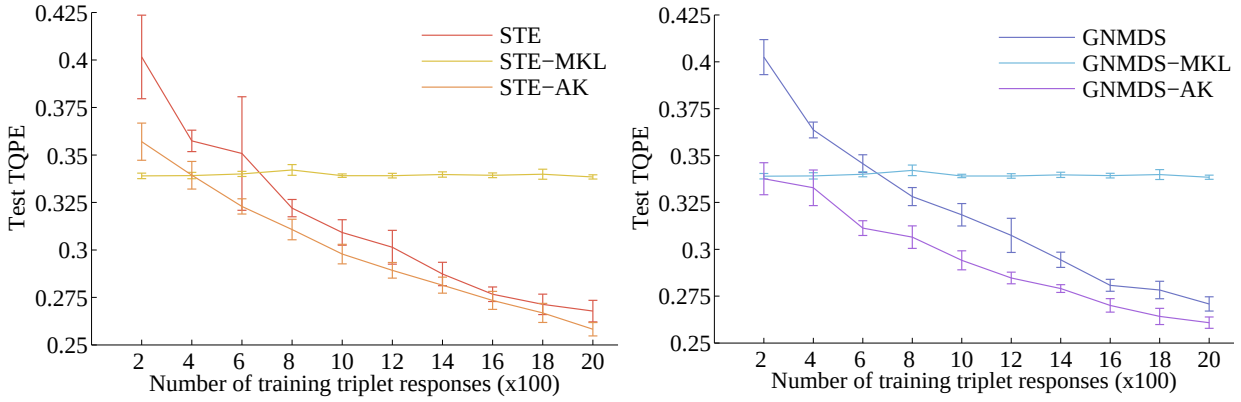
We believe this experiment demonstrates the utility of both $\mathbf{K}_0$ and the auxiliary kernels in RCKL-AK. With very few training responses, the RCKL-AK methods relied on the auxiliary kernels, thus the performance is similar to the RCKL-MKL methods. As more responses are given,

the RCKL-AK methods used $\mathbf{K}_0$ to satisfy the responses that a conic combination of the auxiliary kernels could not. Further evidence for this is shown by the fact that the rank of $\mathbf{K}_0$ increased as the number of training triplets increased. For example, for STE-AK, the mean rank of $\mathbf{K}_0$ was 85.6, 94.2, and 96.2 for 200, 400, and 600 responses in the training set, respectively. In other words, the optimal settings of λ_1 and λ_2 made $\mathbf{K}_0$ more expressive as the number of responses increased.

Figure 7c shows the same STE-AK and GNMDS-AK error plots as in 7a and 7b, but also includes three variations of MKPOE: A non-convex formulation (MKPOE-NC), and two convex formulations (MKPOE-Full and MKPOE-Diag). All metric learning methods perform very similarly, yet worse than the RCKL-AK methods. This shows the limitation of learning a linear function of the inputs without being able to learn features directly as the RCKL-AK methods can.

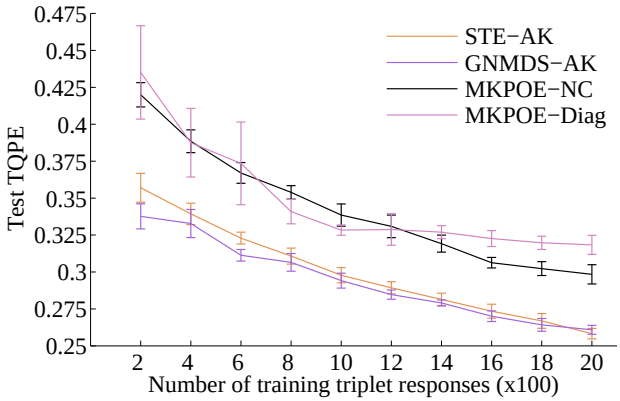
Ideally, the RCKL-AK methods should eliminate $\hat{\mathbf{K}}_4$, $\hat{\mathbf{K}}_5$, and $\hat{\mathbf{K}}_6$ from the model by reducing their corresponding weights μ_4, μ_5, and μ_6 to exactly zero. Figure 8 shows the values of the $\boldsymbol{\mu}$ parameter for STE-AK and GNMDS-AK as the number of training triplet responses increase. Each line in the figure is a different learned weight. Both RCKL-AK methods correctly identify the three auxiliary kernels from which the ground truth kernel was created by setting their corresponding weight parameters to be non-zero. In addition, they assigned weights to the kernels roughly proportional to the ground truth (i.e. μ_1 is set higher than μ_2, etc.). The three noise kernels were assigned very low, and often zero weights. The RCKL-MKL methods learned similar values for the elements of $\boldsymbol{\mu}$ than those in Fig. 8. Since RCKL-MKL learned the relative importance of the auxiliary kernels with only few responses, it had achieved approximately its peak performance and could not improve further with the addition of more responses to train on.

4.1.4.2 Music Artist Data We also performed an experiment using comparisons among popular music artists. The *aset400* data set [40] contains 16,385 triplet responses over 412 music artists gathered from a web survey, and [90] provides five kernels built from various features describing each artist and their music. Two of the kernels were built from text descriptions of the artists, and three were built by extracting acoustic features from songs by each artist. This data set provides a challenge absent in the synthetic data: not all artists appear in the same number of responses. In fact, some artists never appear as the head of a triplet response at all. As a result, this data set represents a setting where feedback was gathered non-uniformly amongst the objects. In light of this, instead



(a) STE Methods

(b) GNMDS Methods



(c) MKPOE and AK Methods

Figure 9: RCKL-AK *aset400* data experiment: Test TQPE vs. number of training responses (ten trials, 90% CI)

of training the models in rounds, we randomly chose 2000 triplet responses as the development set; the rest were used as the test set. As before, we broke the development set into ten subsets. Each subset was progressively added to the working set for training, validation, and testing. Ten percent of the working set was used for validation and 90 percent was used for training. The experiment was performed ten times on different randomly chosen train/validation/test splits.

Discussion: The results, shown in Fig. 9, are similar to those for the synthetic data with a few key differences. First, MKPOE-Full could not be included in this experiment due to its impractically long run-time for an experiment of this size. Second, the RCKL-MKL methods did not perform

as well, relative to the traditional RCKL methods. This could be attributed to the fact that the auxiliary kernels here did not reflect the triplet responses as well as those in the synthetic data. Only one kernel was consistently used in every iteration (the kernel built from artist tags). The rest were either given little weight or completely removed from the model. As with the synthetic data, with 200 and 400 training responses the RCKL-AK methods performed as well as their respective RCKL-MKL counterparts, but as more training responses were added, the RCKL-AK methods began to perform much better. The traditional RCKL methods were more competitive in this experiment than in the synthetic experiments, but were outperformed significantly by RCKL-AK much of the time. This, again, could be because the auxiliary kernels were less useful than with the synthetic data. Both MKPOE methods perform similarly. Though, they seem to suffer even more than the RCKL-AK methods from the lack of many meaningful auxiliary kernels, and over all experiments, have statistically significantly higher test error than both RCKL-AK methods.

4.2 ACTIVELY LEARNING AN OBJECT EMBEDDING WITH AUXILIARY INFORMATION

In the previous section, we demonstrated that using auxiliary information can improve the human efficiency of RCKL methods, resulting in more complete, effective models that require less feedback to learn. However, no consideration was made as to *what* feedback is obtained, and it was assumed that random queries were asked to the source of feedback. Previous work in active learning has shown that if queries are chosen with some foresight, then the human efficiency of learning methods can increase. Consider the case of learning perceptual similarity via crowdsourced triplet feedback. Because crowdsourcing has real costs in both time and money, it is beneficial to reduce the number of dispatched tasks in order to learn a perceptual model of similarity.

In this section, we introduce a novel active learning method for learning similarity from triplet responses with auxiliary information. Our active learning method chooses triplet queries for which our similarity learning method is able to quickly determine what auxiliary information is most relevant to the learning task, but also queries that are informative for learning similarity that cannot be modeled by the auxiliary information. As in the previous section, we learn such a model from

triplet responses, but instead of learning a non-parametric kernel, we create an embedding-learning formulation that can utilize auxiliary information for the sake of computational efficiency. In our embedding-learning method, one portion of the embedding is a parametric combination of auxiliary features, while the other portion is non-parametric and is learned directly from triplet responses to resolve remaining similarity details, similar to $\mathbf{K}_0$ in RCKL-AK methods. Because our formulation is probabilistic in nature, we are able to use information-theoretic notions of uncertainty in the solutions our method produces. Using this, we develop an active learning method to choose queries that reduce this uncertainty in expectation. In doing so, our active learning method finds queries which mutually benefit both the parametric component – quickly finding the relevant feature space whose generalization matches human judgment – as well as the non-parametric component. [2]

4.2.1 A Probabilistic Embedding-Learning Formulation with Auxiliary Features

For our similarity-learning method, we assume that the underlying perceptual similarity space from which $\mathcal{T}$ is drawn can be modeled by embedding the objects in a $\hat{d} + d$-dimensional space. To begin, consider the case without auxiliary information, where the goal is to learn a representation of n objects in the form of an embedding $\hat{\mathbf{X}} = \{\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, ..., \hat{\mathbf{x}}_n\} \subset \mathbb{R}^{\hat{d}}$. This is the goal of Crowd Kernel Learning (CKL). CKL uses a probability that a triplet is satisfied, defined in (3.16), to learn an embedding. This probability can be interpreted as the likelihood of a triplet response given an embedding of the objects:

$$p\left((a,b,c)\,|\hat{\mathbf{X}}\right) = \frac{\mu + d_2^2\left(\hat{\mathbf{x}}_a, \hat{\mathbf{x}}_c\right)}{2\mu + d_2^2\left(\hat{\mathbf{x}}_a, \hat{\mathbf{x}}_c\right) + d_2^2\left(\hat{\mathbf{x}}_a, \hat{\mathbf{x}}_b\right)} \tag{4.16}$$

To find an embedding that models a set of responses $\mathcal{T}$ well, one can maximize the joint log-likelihood over $\mathcal{T}$:

$$\max_{\hat{\mathbf{x}}} \sum_{(a,b,c)\in\mathcal{T}} \log\left(p\left((a,b,c)\,|\hat{\mathbf{X}}\right)\right) \tag{4.17}$$

The model produced by solving (4.17) is *non-parametric* in that each object is represented by a unique point in the embedding space without rigid structure. Such a model enjoys the same level of expressiveness as learning a non-parametric kernel matrix using similar RCKL formulations.

[2] The material presented in this section is currently under review.

A limitation of (4.17), much like RCKL methods that learn a kernel, is that in the presence of few obtained triplet responses, it may not produce a model that generalizes well to the numerous triplet responses yet to be provided. One way to address this is to include side information about the objects. For this, we consider side information in the form of features that characterize each object in some meaningful manner. Let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\} \subset \mathbb{R}^d$ be *auxiliary feature* vectors for each object. These features may represent important characteristics in the underlying perceptual similarity space of objects. If they do, then incorporating them into a learned embedding of the objects may allow it to not only model responses in $\mathcal{T}$, but also generalize to triplet responses yet to be obtained, creating a more complete model of similarity. To include this side information, we propose to learn a model comprised of a *parametric* component to be learned over the auxiliary features in conjunction with a non-parametric embedding. We define the parametric component of our model to be $\mathbf{X}' = \{\mathbf{x}'_1, \mathbf{x}'_2, ..., \mathbf{x}'_n\} \subset \mathbb{R}^d$, where:

$$\mathbf{x}'_i = \left[\mathbf{w}^1\mathbf{x}_i^1, \mathbf{w}^2\mathbf{x}_i^2, .., \mathbf{w}^d\mathbf{x}_i^d\right] \tag{4.18}$$

Here, $\mathbf{w} \in \mathbb{R}^d$ is a learned weight parameter. Let $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, ...\mathbf{y}_n\} \subset \mathbb{R}^{\hat{d}+d}$ be the combination of the parametric and non-parametric components, where each object $\mathbf{y}_i$ is represented by a concatenation of vectors:

$$\mathbf{y}_i = \left[\mathbf{w}^1\mathbf{x}_i^1, \mathbf{w}^2\mathbf{x}_i^2, .., \mathbf{w}^d\mathbf{x}_i^d, \hat{\mathbf{x}}_i^1, \hat{\mathbf{x}}_i^2, ..., \hat{\mathbf{x}}_i^{\hat{d}}\right] \tag{4.19}$$

The first d elements of $\mathbf{y}_i$ are the auxiliary features $\mathbf{x}_i$, each weighed by an element of the learned parameter $\mathbf{w}$, i.e. $\mathbf{w}_i \circ \mathbf{x}_i$. A proper setting of $\mathbf{w}$ would emphasize auxiliary features that adhere to the general trend in obtained triplet responses, while reducing the influence of features that do not. The last $\hat{d}$ elements of $\mathbf{y}_i$, $\hat{\mathbf{x}}_i$, are free parameters to be learned in a non-parametric fashion similar to traditional CKL.

To learn $\mathbf{Y}$ we propose a probabilistic formulation depicted graphically in Fig. 10. We assume that each of the n objects have a true embedding $\mathbf{y}$ in a human-perceptual metric space. We treat $\mathbf{y}$ as an intermediate variable that has two components: $\mathbf{x}' \in \mathbb{R}^d$ and $\hat{\mathbf{x}} \in \mathbb{R}^{\hat{d}}$. A subset of the elements of $\mathbf{y}$, i.e a subset of the characteristics in the true perceptual similarity space, are modeled exclusively by $\mathbf{x}'$. The parametric component $\mathbf{x}'$ is itself an intermediate variable that is defined by $\mathbf{w}$ and $\mathbf{x}$. The non-parametric component $\hat{\mathbf{x}}$ subsequently models the remaining perceptual characteristics

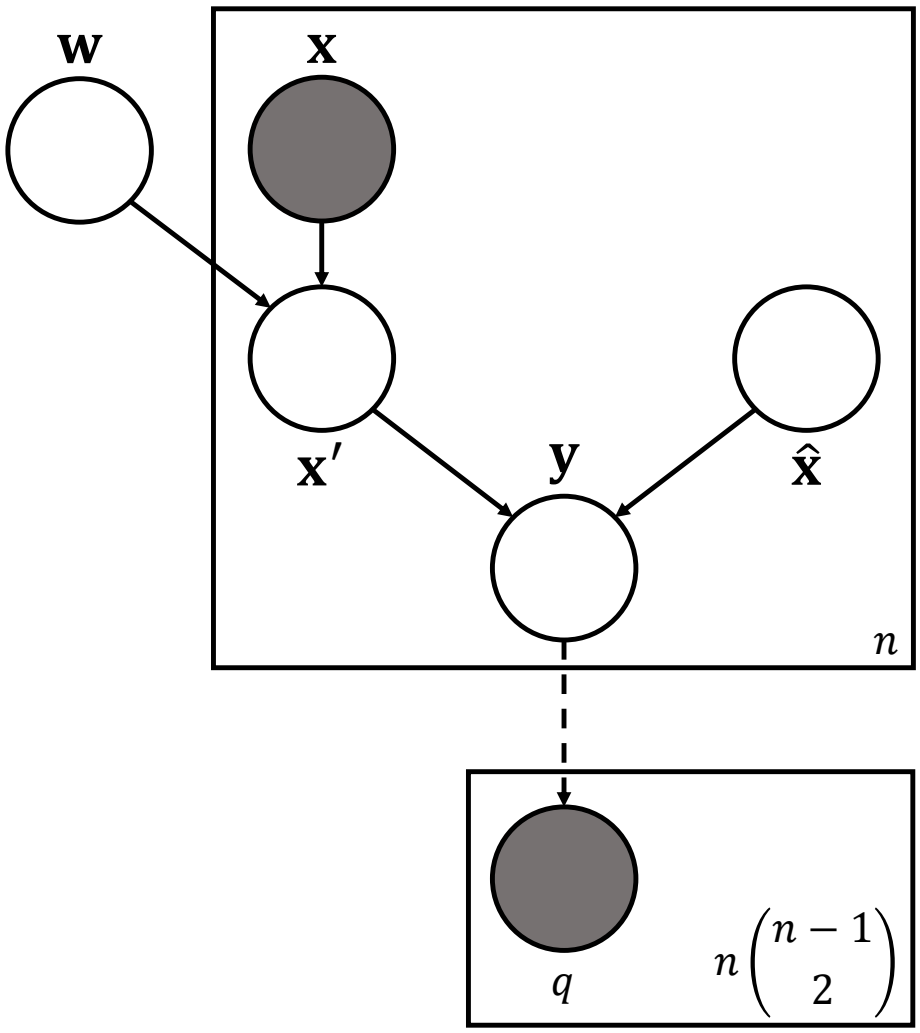


Figure 10: Plate diagram depicting TACKL variable relationships

not captured by $\mathbf{x}'$. The likelihood of a triplet query over the n objects is then dependent on the individual embeddings of objects. Which specific queries are dependent on a single object embedding is shown in Fig. 11 (x_1 and x_2 indicate any two unique objects not including a).

Because $\mathbf{X}$ is given, the only model parameters needed to learn an embedding $\mathbf{Y}$ are $\mathbf{w}$ and $\hat{\mathbf{X}}$. We choose to do so by maximizing their log-joint posterior given the input variables. We can decompose the log-joint posterior into two terms:

$$\log\left(p\left(\mathbf{w},\hat{\mathbf{X}}|\mathbf{X},\mathcal{T}\right)\right) = \log\left(p\left(\mathbf{w}|\mathbf{X},\mathcal{T}\right)\right) + \log\left(p\left(\hat{\mathbf{X}}|\mathbf{w},\mathbf{X},\mathcal{T}\right)\right) \tag{4.20}$$

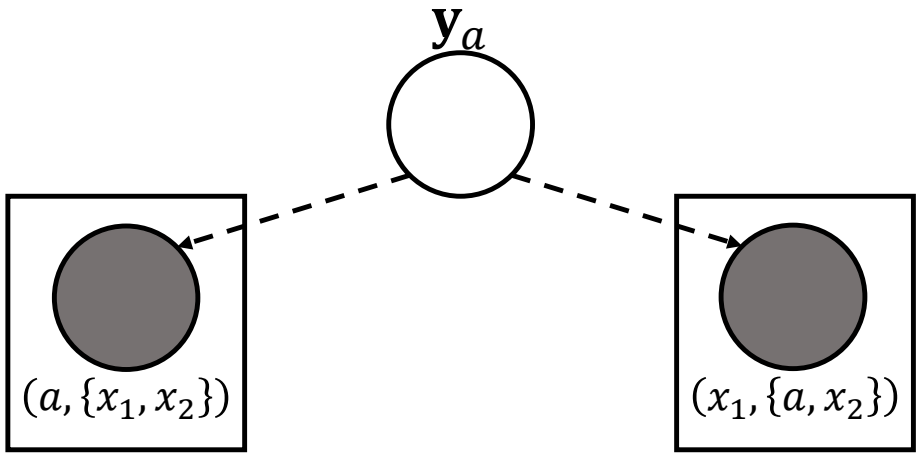


Figure 11: Plate diagram depicting triplet query/object dependencies

This decomposition reflects the intuition behind the two components in our model. In the parametric component, $\mathbf{X}$ contains features that model a portion of characteristics in $\mathbf{Y}$, and $\mathbf{w}$ weighs them by importance. The non-parametric component learned directly ($\hat{\mathbf{X}}$) models *only* the characteristics not modeled by the parametric component, thus making it dependent on $\mathbf{X}$ and $\mathbf{w}$. Intuitively, the parametric component leverages auxiliary features to model as much of the perceptual space as possible, while the non-parametric component effectively fills in the remaining gaps.

Following this same intuition, we choose to learn $\mathbf{w}$ and $\hat{\mathbf{X}}$ in two stages. First, we find $\mathbf{w}$ by maximizing the log-posterior of $\mathbf{w}$ given the input variables. Then, we fix $\mathbf{w}$ to be the solution in the first step and maximize the log-posterior of $\hat{\mathbf{X}}$ given all other variables. To learn $\mathbf{w}$, note that the log-posterior of $\mathbf{w}$ can be further decomposed:

$$\begin{aligned} \log\left(p\left(\mathbf{w}|\mathbf{X}, \mathcal{T}\right)\right) &\propto \log\left(p\left(\mathcal{T}|\mathbf{w}, \mathbf{X}\right)\right) + \log\left(p\left(\mathbf{w}, \mathbf{X}\right)\right) \\ &= \sum_{(a,b,c)\in\mathcal{T}} \log\left(p\left((a,b,c)\,|\mathbf{w}, \mathbf{X}\right)\right) + \log\left(p\left(\mathbf{w}, \mathbf{X}\right)\right) \end{aligned} \tag{4.21}$$

By maximizing the log-posterior of $\mathbf{w}$, we maximize the sum of the log-likelihood of $\mathcal{T}$ and the log-joint prior probability of $\mathbf{w}$ and $\mathbf{X}$. The log-likelihood of the triplet responses is defined by the sum of log-likelihoods of single responses:

$$p\left((a,b,c)\,|\mathbf{w}, \mathbf{X}\right) = p\left((a,b,c)\,|\mathbf{X}'\right) = \frac{\mu + d_2^2\left(\mathbf{x}'_a, \mathbf{x}'_c\right)}{2\mu + d_2^2\left(\mathbf{x}'_a, \mathbf{x}'_c\right) + d_2^2\left(\mathbf{x}'_a, \mathbf{x}'_b\right)} \tag{4.22}$$

These likelihoods are similar to (4.16) with objects represented as a weighing of their auxiliary features. We assume $\mathbf{w}$ and $\mathbf{X}$ are independent, making $\log\left(p\left(\mathbf{w},\mathbf{X}\right)\right) = \log\left(p\left(\mathbf{w}\right)\right) + \log\left(p\left(\mathbf{X}\right)\right)$. Since $\mathbf{X}$ is given, it's log-prior probability need not be considered when maximizing with respect to $\mathbf{w}$. Finally, we choose the prior on $\mathbf{w}$ to be uniform over $\mathbb{R}_+^d$, effectively restricting $\mathbf{w}$ to be positive, making our maximization problem:

$$\begin{aligned} \max_{\mathbf{w}} \quad & \sum_{(a,b,c)\in\mathcal{T}} \log\left(p\left((a,b,c)\,|\mathbf{w},\mathbf{X}\right)\right) \\ \text{s.t.} \quad & \mathbf{w} \geq 0. \end{aligned} \tag{4.23}$$

which we solve via projected gradient descent.

As with $\mathbf{w}$, the log-posterior of $\hat{\mathbf{X}}$ can be decomposed into the sum of the log-likelihood and log-prior. In this case, the likelihoods over individual triplets are defined as:

$$p\left((a,b,c)\,|\mathbf{w},\mathbf{X},\hat{\mathbf{X}}\right) = p\left((a,b,c)\,|\mathbf{Y}\right) = \frac{\mu + d_2^2\left(\mathbf{y}_a,\mathbf{y}_c\right)}{2\mu + d_2^2\left(\mathbf{y}_a,\mathbf{y}_c\right) + d_2^2\left(\mathbf{y}_a,\mathbf{y}_b\right)} \tag{4.24}$$

We assume independence between the variables in the log-prior, thus we can rewrite it as:

$$\log\left(p\left(\mathbf{w},\mathbf{X},\hat{\mathbf{X}}\right)\right) = \log\left(p\left(\mathbf{w}\right)\right) + \log\left(p\left(\mathbf{X}\right)\right) + \log\left(p\left(\hat{\mathbf{X}}\right)\right) \tag{4.25}$$

Here, $\mathbf{w}$ and $\mathbf{X}$ are given, making their priors inconsequential in maximizing with respect to $\hat{\mathbf{X}}$. Furthermore,, we choose the prior on $\hat{\mathbf{X}}$ to be uniform over $\mathbb{R}^{\hat{d}}$. Thus, to learn the non-parametric portion of our model, we solve the following maximization problem via gradient descent:

$$\max_{\hat{\mathbf{X}}} \sum_{(a,b,c)\in\mathcal{T}} \log\left(p\left((a,b,c)\,|\mathbf{w},\mathbf{X},\hat{\mathbf{X}}\right)\right) \tag{4.26}$$

In solving (4.23) we learn the parametric portion of the model to fit to the given triplet responses. Then, by solving (4.26) with fixed $\mathbf{w}$, we learn the non-parametric component that complements the parametric component. Because we use a **T**wo step learning procedure that utilizes **A**uxiliary features, based on **CKL**, we call our method TACKL.

In our formulation, we assume d and $\hat{d}$ are sufficiently small to conform to the common assumption that perceptual similarity is low-dimensional. For high-dimensional auxiliary features we can apply standard dimensionality reduction techniques, such as PCA, and process this low-dimensional representation. Another option is to process the high-dimensional data directly, employing regularization techniques which promote feature sparsity such as ℓ_1-norm regularization [134]. Such

regularization terms may take the place of our prior probabilities; however, since different types of priors are not the emphasis of our approach, we leave these directions as future work.

Formulating similarity learning from triplet responses in terms of probability distributions allows us to reason about triplet queries in an information theoretic context. Specifically, we can define concepts such as entropy and information gain. In the next section, we show how we use these concepts to actively select the most informative queries to ask humans to learn a more complete model with fewer responses.

4.2.2 Active Embedding-Learning with Auxiliary Features

The general strategy we employ for our active learning scheme is to reduce the uncertainty our model has in the position of each object individually. As a result, our method works in rounds where n queries are chosen, each with a different head object x_a, and actively selected pairs of objects x_b and x_c. Then, for a round t, the entire round of queries is posed to a source of feedback, whose responses are then used to learn $\mathbf{w}^{t+1}$ and $\hat{\mathbf{X}}^{t+1}$. We measure uncertainty in a head x_a at round t by the *entropy* of its learned variables:

$$\begin{aligned} H^t\left(\mathbf{w}, \hat{\mathbf{x}}_a\right) &= \int_{\mathbf{w}} p\left(\mathbf{w}|\mathbf{X}, \mathcal{T}_a^t\right) \log\left(p\left(\mathbf{w}|\mathbf{X}, \mathcal{T}_a^t\right)\right) \\ &\quad * \int_{\hat{\mathbf{x}}_a} p\left(\hat{\mathbf{X}}|\mathbf{w}, \mathbf{X}, \mathcal{T}_a^t\right) \log\left(p\left(\hat{\mathbf{X}}|\mathbf{w}, \mathbf{X}, \mathcal{T}_a^t\right)\right) d\hat{\mathbf{x}}_a d\mathbf{w} \end{aligned} \tag{4.27}$$

Here, $\mathcal{T}_a^t$ is all obtained triplet responses at round t with head x_a. In round $t+1$ we wish to select the query for which its response in expectation, when added to $\mathcal{T}_a^t$, will reduce $H^t\left(\mathbf{w}, \hat{\mathbf{x}}_a\right)$ the most. Thus, we choose the triplet query $(a, \{b, c\})$ with highest expected *information gain*:

$$H^t\left(\mathbf{w}, \hat{\mathbf{x}}_a\right) - p_{abc}^t * H_{abc}^t\left(\mathbf{w}, \hat{\mathbf{x}}_a\right) - \left(1 - p_{abc}^t\right) * H_{acb}^t\left(\mathbf{w}, \hat{\mathbf{x}}_a\right) \tag{4.28}$$

where $H_{abc}^t\left(\mathbf{w}, \hat{\mathbf{x}}_a\right)$ is $H^t\left(\mathbf{w}, \hat{\mathbf{x}}_a\right)$ evaluated over $\mathcal{T}_a^t \cup (a, b, c) = \mathcal{T}_{abc}^t$, and p_{abc}^t is the probability that a human will respond (a, b, c) when prompted with $(a, \{b, c\})$:

$$p_{abc}^t = \int_{\mathbf{w}} p\left(\mathbf{w}|\mathbf{X}, \mathcal{T}_{abc}^t\right) \int_{\hat{\mathbf{x}}_a} p\left(\hat{\mathbf{X}}|\mathbf{w}, \mathbf{X}, \mathcal{T}_{abc}^t\right) d\hat{\mathbf{x}}_a d\mathbf{w} \tag{4.29}$$

Algorithm 4 Active TACKL

Input: $\mathbf{X} \subset \mathbb{R}^d$

$\mathcal{T}^0 \leftarrow \emptyset$
for round $t = 0, 1, 2, ...$ **do**
  $\mathcal{Q}^t \leftarrow \emptyset$
  for head $a = 1, 2, 3..., n$ **do**
    if $t == 0$ **then**
      $b' \leftarrow$ Random draw from $\{1, ...n\} - a$
      $c' \leftarrow$ Random draw from $\{1, ...n\} - \{a, b\}$
    else
      for all $\{b, c\} \subset \{1, ..., n\} : b, c \neq a$ **do**
        $score\,(a, \{b, c\}) \leftarrow$ (4.30) using $\hat{\mathbf{X}}^t$ and $\mathcal{T}^t$
      end for
      $\{b', c'\} \leftarrow \text{argmin}_{b,c}\, score\,(a, \{b, c\})$
    end if
    $\mathcal{Q}^t \leftarrow \mathcal{Q}^t \cup (a, \{b', c'\})$
  end for
  $\mathcal{T}^{t+1} \leftarrow \mathcal{T}^t \cup responses\,((\mathcal{Q}^t)$
  $\mathbf{w}^{t+1} \leftarrow$ solution of (4.23) using $\mathcal{T}^{t+1}$ and $\mathbf{X}$
  $\hat{\mathbf{X}}^{t+1} \leftarrow$ solution of (4.26) using $\mathcal{T}^{t+1}$, $\mathbf{X}$, and $\mathbf{w}^{t+1}$
end for
return $\left\{\mathbf{w}^{t+1}, \hat{\mathbf{X}}^{t+1}\right\}$

For a single head, the first term in the expected information gain is constant. Thus, our active learning scheme scores pairs of objects given a head using expected query entropy:

$$p^t_{abc} * H^t_{abc}\,(\mathbf{w}, \hat{\mathbf{x}}_a) + \left(1 - p^t_{abc}\right) * H^t_{acb}\,(\mathbf{w}, \hat{\mathbf{x}}_a) \tag{4.30}$$

and chooses the one with the lowest score. Note, that for finding the entropy, we fix the non-parametric components of all objects besides x_a to be their estimate at round t, making entropy a function of only $\hat{\mathbf{x}}_a$ and $\mathbf{w}$. A similar assumption is used for A-CKL (Sec. 3.4.2). If the prior on their entirely non-parametric model is chosen to be uniform over the current positions of objects, then their scoring criteria amounts to finding information gain if x_a is randomly moved to the exact position of one of the other objects. This intuition holds for the non-parametric component of TACKL. However, the parametric component in our model couples all objects through $\mathbf{w}$. Thus, when scoring a query, the entropy calculation is based on randomly repositioning x_a via $\mathbf{w}$ and

$\hat{\mathbf{x}}_a$, and all other objects through $\mathbf{w}$. In practice, we perform Monte Carlo integration in (4.30) by uniformly sampling $\mathbf{w}$ and $\hat{\mathbf{X}}$ over domains restricted by the magnitudes of their estimates.

Algorithm 4 outlines the entire process of actively learning perceptual similarity using our methods. The first round (round 0) of query selection is random to establish a basis for subsequent rounds. For all rounds $t > 0$, for each head, all pairs of objects are scored. For learning similarity over a large number of objects, scoring every triplet is prohibitively expensive. In practice, we randomly choose a subset of pairs to score. The query that results in the lowest expected entropy for each head is chosen and added to the set of queries for round t. Then, these queries are posed to the source of feedback for responses, which are added to the running pool. The model parameters are updated from the pool of responses, consequently driving the next round of active selection.

Querying for triplets in rounds has a practical application. Often to obtain large quantities of human feedback *crowdsourcing* technologies are used. For crowdsourcing, human inference tasks are typically deployed in batches so many different people can work on tasks at once. By selecting n at a time, one could deploy batches of triplet queries, receive responses, learn a model, and then select n more to repeat the process. In the next section, we evaluate TACKL and our active variant A-TACKL on both synthetic data as well as real data gained from crowdsourcing triplet responses.

4.2.3 Experiments

To evaluate TACKL we compare it experimentally to CKL, using both random query selection and their active schemes, A-TACKL and A-CKL. The goal is to show the advantage our method gains from using auxiliary information for both random and active query selection schemes. As with the RCKL-AK experiments, we measure the accuracy of a learned perceptual model by Triplet Query Prediction Error (TQPE). We evaluate each method after each round of query selection to show error as a function of training set size. In the interest of fairness, we allow the active learning methods to score the same number of randomly selected pairs per head. Also, we fix the number of samples in the Monte Carlo integration for A-CKL to be $n\sqrt{n}$, and for $\mathbf{w}$ and $\hat{\mathbf{X}}$ in A-TACKL to be $\sqrt{n}$ and n. As a result, both active learning methods take the same number of samples. CKL and TACKL are given the same randomly selected responses each round, and all methods are provided the same random initial round of responses. We set $\hat{d} \leftarrow d$ for the TACKL methods, $\hat{d} \leftarrow 5$ for the

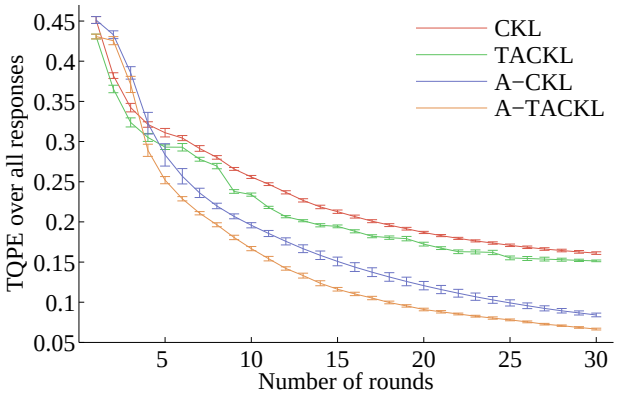


Figure 12: TACKL synthetic data experiments (ten trials, 90% CI)

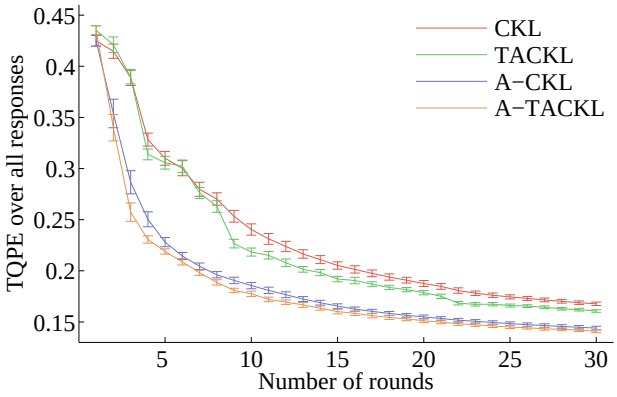


Figure 13: TACKL Yummly data experiments (20 trials, 90% CI)

CKL methods (6 for synthetic experiment), and $\mu \leftarrow 10^{-4}$ for all methods, as slight variations in their settings did not significantly impact any method.

4.2.3.1 Synthetic Data The first experiment highlights the performance of TACKL in a controlled and idealized synthetic setting. We generated 250 objects in six dimensions as the ground truth perceptual space $\mathbf{X}^*$, each dimension being drawn from a different distribution: half from uniform distributions, and half from mixtures of normal distributions. From $\mathbf{X}^*$ we generate an exhaustive set of all triplet responses. All queries by the methods are answered by this pool, and the entire pool is used to evaluate TQPE. As auxiliary features, the TACKL methods are given three dimensions from the ground truth as well as three more randomly generated noise features. This input coincides with our model assumptions: potentially only some auxiliary features represent dimensions in the underlying perceptual similarity space.

Discussion: Figure 12 shows the mean TQPE versus number of rounds from running experiments for the ten trials of different random initialization. As with all subsequent figures in this chapter, error bars represent a 90% confidence interval. Note that both active learning methods perform poorly in the first few rounds. With such few triplets in $\mathcal{T}_a^t$, the information gain scores are not meaningful and having variety among pairs seems to allow the random selection methods to learn more accurate models. However, A-TACKL recovers from this cold start by round four and

subsequently has significantly less error than all other methods. Overall, the TACKL methods are more accurate than their CKL equivalents because they are given auxiliary information that allows for better generalization.

4.2.3.2 Yummly Food Data To test TACKL on real responses obtained from humans, we performed two tests. First, we used a data set of triplet responses gathered over food images sourced from Yummly recipes using Amazon Mechanical Turk (AMT) [155]. Of the 100 images, 73 have annotations from Yummly measuring to what degree each food image has six taste properties: salty, savory, sour, bitter, sweet, and spicy. We used these as auxiliary features in our experiment. To obtain triplet responses, the authors showed AMT users various-sized grids of food images and were asked to select the images that "taste most similar". Selected images were deemed the x_a and x_b objects and the rest were deemed x_c for responses (a, b, c). Because their goal was to evaluate grid-size versus accuracy of feedback, a certain amount of noise is present in the responses. For the 73 images with auxiliary features, the data set contains 66,497 unique responses, which is roughly 36% responses to all possible queries. To reconcile the lack of a full set of responses to all queries, the active selection methods in our experiment simply chose to ask the highest scoring query for which there was a response. We evaluate error for each method on the full set of responses.

Discussion: The results of our experiments on the Yummly data are shown in Fig. 13. As with the synthetic experiments, the TACKL methods have consistently less error than their CKL counterparts, often by a significant margin. In this case, however, the active methods do not suffer from cold start. This may be due to the fact that the number of objects and responses to evaluate each model is much less than in the synthetic responses. As a result, the active methods can hone in on a satisfactory model to reason about with much fewer responses. Also in this experiment, the active methods outperform the random methods by a much larger margin in the middle rounds, indicating that the active methods especially benefited from asking informative queries in early rounds. If just the taste features themselves are used as a similarity model, a TQPE of 0.4562 is achieved. Despite this, the TACKL methods are able to perform well and find the relevant portions of taste, giving large weight to the "sweet" feature and frequently assigning zero weight to "sour" and "spicy". Indeed, the data contains many varieties of sweet (pastries, fruits) and not sweet (rice, pasta) foods, while few of the foods could be considered sour or spicy.

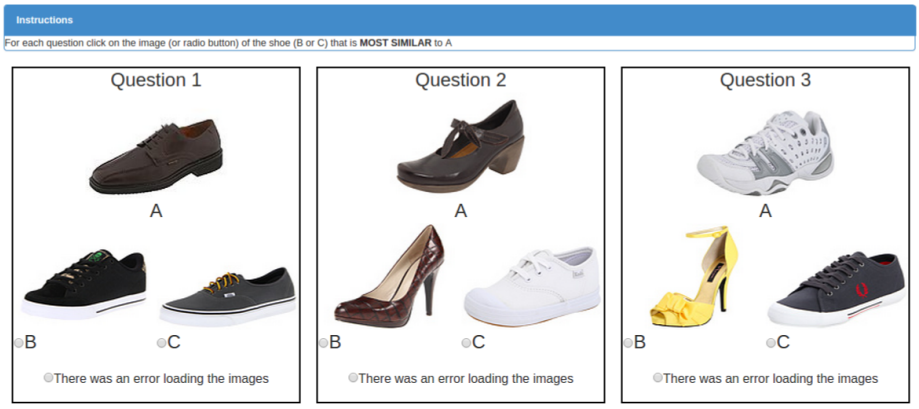
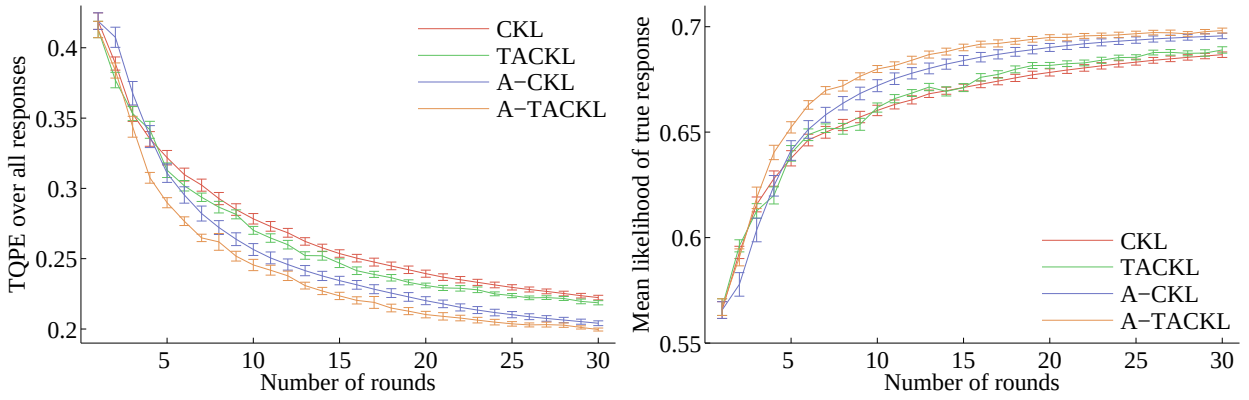


Figure 14: Example task deployed via AMT to collect Zappos triplet data (three triplet queries per task)

4.2.3.3 Zappos Shoes Data To fully evaluate the performance of our active learning approach and to provide a baseline for comparisons with future active triplet learning schemes, we sought to create a data set containing responses to all possible triplet queries over a nontrivial number of objects. By doing so, we could truly evaluate the active triplet query selection methods by allowing them to choose any query and obtain a response from an actual human. To our knowledge, there does not currently exist such a data set that is publicly available.

To this end, we chose a representative subset of 85 images of shoes from the Zappos50K data set [158] by performing k-means clustering on the images represented by their AlexNet [73] features. These images were used in the deployment of an AMT task consisting of all 296,310 triplet queries among images. A screen-shot of the deployed task is shown in Fig. 14. Each query was dispatched to three different AMT users, and the ground truth response was determined by majority vote. Approximately 1,020 different users provided responses to queries. Among them, 82% of the queries had full agreement among users, and 18% had 66% agreement. For auxiliary features, we used the AlexNet feature representations of each image projected to three dimensions using Principal Components Analysis.

Discussion: Figure 15 contains graphs depicting the results of our experiments on the Zappos data. Figure 15a shows the mean TQPE of the learned models as in previous experiments. Not



(a) TQPE over all responses vs. number of rounds

(b) Mean likelihood of true response vs. number of rounds

Figure 15: TACKL Zappos data experiments (ten trials, 90% CI)

shown in the figure is that the full AlexNet features by themselves achieved error of 0.3882. Most methods surpassed this after two rounds. The difference between the active and random methods is much smaller. This could be attributed to the slight amount of noise in the large pool of responses, which affects both learning model parameters and evaluation of the error. As with the Yummly data experiments, A-TACKL does not suffer from cold start. However, this is not the case for A-CKL. A-TACKL is able to rely on the auxiliary information when given few triplet responses and learn a model for which active learning benefits. Overall, the TACKL methods outperform their CKL counterparts.

Figure 15b shows the mean likelihood of all triples responses for all learned models. This metric conveys the *margin* by which the models adhere to all responses. While TACKL has consistently lower error than CKL, it is much less stable in terms of likelihood. This implies that TACKL can consistently raise the likelihood of more correct responses above 0.5, but it sometimes does so without raising it far beyond 0.5. Nevertheless, the TACKL methods perform relatively well according to this metric. By the time A-TACKL is given 30 responses, on average, the distance between x_a and x_c is more than twice than the distance between x_a and x_b for all responses (a, b, c).

Finally, Fig. 16 shows one of the embeddings learned by A-TACKL after 30 rounds, projected to two dimensions. While the characteristics being displayed by this embedding are largely up to

Figure 16: Example two-dimensional embedding learned by A-TACKL on Zappos data

interpretation, some trends seem to be present. First, certain clusters of shoes seem to have formed. High heels, athletic shoes, flats, and men's dress shoes all form loose clusters. Also, the shoes in the bottom half seem to be more formal than the top half. Note that much of the information is lost by projecting from six dimensions to two. If the goal is to learn a visualization of perceptual similarity, than a two-dimensional embedding can be learned. In our experience with embedding-learning methods, when visualizing embeddings in 2D, explainable characteristics are more obvious when a

two-dimensional embedding is learned directly.

4.3 SUMMARY

In this chapter, we focused on the task of improving human efficiency in learning models of similarity from triplet feedback. We did this in two ways. First, we introduced a novel RCKL framework that allowed traditional RCKL methods to include auxiliary information in the form of kernels built from features characterizing each object. When some of these features represent trends in the triplet feedback, using their corresponding kernels in the process of learning a non-parametric kernel matrix can allow the resulting similarity model to convey a more general, effective notion of similarity between objects. Our framework, RCKL-AK, is able to identify and incorporate relevant auxiliary information to include in the model without incurring significant overhead in learning a kernel. Experimentally, we show on synthetic and real data sets that RCKL-AK is able to learn kernels that satisfy more out of sample triplet responses than traditional RCKL methods, as well as recent metric learning methods that use multiple kernels as a basis for learning.

To further increase human efficiency in learning a model of similarity from triplet feedback, we also developed a novel active learning method. Here, we focused on the embedding-learning formulation of CKL. Using CKL as a basis, we derive a novel method to learn an embedding of objects that adheres to triplet feedback. Our method, called TACKL, learns an embedding with two components: a parametric component consisting of a weighted concatenation of auxiliary features, and a non-parametric component to model true object relationships not captured by the parametric component, similar in spirit to RCKL-AK. TACKL is formulated probabilistically, allowing us to create an active learning scheme based on information-theoretic notions of which queries are most informative. In our experimental evaluation, we show that by leveraging auxiliary features and actively selecting triplet queries, TACKL can learn more complete and accurate models of perceptual similarity than methods that do not use side information and select queries at random.

5.0 ONLINE SIMILARITY LEARNING FROM TRIPLET RESPONSES

To this point, we have discussed RCKL methods in terms of learning a kernel by solving a semidefinite program (SDP) in batch from a set of given triplet responses. However, in numerous practical applications, a batch approach is not appropriate due to the online and dynamic nature of how feedback is obtained. Consider the example of a users shopping on an online store. If the product a user wants to buy is out of stock, the store may have an automated system in place that suggests suitable replacement items. When a user considers replacements and chooses one, she is implicitly providing feedback that can be used to model how humans view the relationships among items. Thus, as users interact with the store, feedback is constantly being given. More specifically in this case, the user is expressing that the item she selected is in some sense more similar to the out of stock item than the other suggestions. This natural interaction with the online store can be understood as triplet feedback, and RCKL methods can be used to learn models that reflect how users view the relationships among products. Using a learned model, the store can suggest better replacements, deliver more relevant product information, or provide advertisements informed by product relationships. In this example, as well as others, it is important to quickly update the model of similarity so that systems using it will have the most up-to-date information.

Such applications motivate the need for an *efficient* and *online* method for learning similarity among a large number of objects from triplet feedback. Batch RCKL methods scale poorly for large object collections 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. For large n, this is prohibitively slow for the aforementioned online applications. In this chapter, we introduce a novel online RCKL framework called **E**fficient online **R**elative comparison **K**ernel **LE**arning (ERKLE) that achieves computational 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 [4, 143]. 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 [31], 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. [1]

5.1 EFFICIENT ONLINE RELATIVE COMPARISON KERNEL LEARNING (ERKLE)

In our online RCKL problem setting, at each time step j a triplet response $t_j = (a, b, c)$ is obtained from a source. The goal is to produce a kernel matrix $\mathbf{K}_j$ that satisfies not only t_j, but all triplet responses obtained in previous time steps, as well. One could apply any of the batch RCKL methods outlined in Sec. 3.2.2.1 to this setting by simply running their optimization procedures over all feedback $\mathcal{T}_j = \{t_i : i = 1, ...j\}$ every time a new triplet response is obtained. The main issue with applying batch RCKL methods in such a way is in ensuring the kernel matrices produced are PSD. To do this, traditional batch RCKL methods project intermediate solutions after each gradient step in their optimizations back onto the PSD cone (Π_+ on line 4 of Alg. 1). In the absence of any prior knowledge of its structure, a batch gradient step can produce intermediate solutions with an unknown number of negative eigenvalues. Because of this, projection is commonly performed using the procedure outlined in (3.11), which performs full eigendecomposition after a gradient step, an $O\left(n^3\right)$ time procedure. In the online learning setting, this projection may be needed to be performed multiple times each time new feedback is obtained. This renders batch methods computationally prohibitive for learning the similarity of a large number of objects in an online manner for many real-world tasks.

[1] The material presented in this chapter was originally published as [60].

5.1.1 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 response. As shown in (3.10), many RCKL loss functions L naturally decompose into the sum over losses l defined on individual units of obtained feedback (triplet responses in our case). From this decomposition, ERKLE first performs the following stochastic gradient step:

$$\mathbf{K}'_j \leftarrow \mathbf{K}_{j-1} - \delta_j \nabla l\left(\mathbf{K}_{j-1}, t_j\right) \tag{5.1}$$

where triplets $t_1, ..., t_{j-1}$ have been observed, and $\mathbf{K}_{j-1}$ is the online solution after observing the $j-1$ triplet. Note that our online formulation does not include trace regularization. Although this may impact our method in generalizing to unseen triplets, our experiments show that our online formulation achieves good generalization through carefully constructed, data-dependent step sizes δ_j, as detailed in Sec. 5.1.3.

5.1.2 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_+ 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* $\mathbf{G}_t$ over a triplet $t = (a, b, c)$, where the entries are defined as:

$$\mathbf{G}_t^{ij} = \begin{cases} -2 & \text{if } i = a, j = b \text{ or } i = b, j = a \\ 2 & \text{if } i = a, j = c \text{ or } i = c, j = a \\ 1 & \text{if } i = b, j = b \\ -1 & \text{if } i = c, j = c \\ 0 & \text{otherwise.} \end{cases} \tag{5.2}$$

Now, consider the following choice for the stochastic step:

$$\nabla l\left(\mathbf{K}, t\right) = f\left(\mathbf{K}, t\right) \mathbf{G}_t \tag{5.3}$$

Algorithm 5 Efficient PSD Projection

procedure $\Pi_+^1(\mathbf{K})$
    Find $\lambda_\downarrow$ and $\mathbf{v}_\downarrow$ from $\mathbf{K}$
    if $\lambda_\downarrow < 0$ **then**
        return $\mathbf{K} - \lambda_\downarrow \mathbf{v}_\downarrow \mathbf{v}_\downarrow^T$
    else
        return $\mathbf{K}$
    end if
end procedure

where f is a real-valued function. With (5.3) as the gradient in (5.1), $\mathbf{K}_{j-1}$ 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:

$$f\left(\mathbf{K}, t = (a, b, c)\right) = 1 - p_{abc}^{\mathbf{K}} \tag{5.4}$$

Where $p_{abc}^{\mathbf{K}}$ is the STE probability defined in (3.13). Similarly, by defining f to be:

$$f\left(\mathbf{K}, t = (a, b, c)\right) = \begin{cases} 1 & \text{if } d_{\mathbf{K}}^2(a, b) + 1 < d_{\mathbf{K}}^2(a, c) \\ 0 & \text{otherwise} \end{cases} \tag{5.5}$$

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 5 outlines the procedure for efficient projection in ERKLE. Here, $\lambda_\downarrow$ and $\mathbf{v}_\downarrow$ are the smallest eigenvalue and eigenvector of matrix $\mathbf{K}$. Finding $\lambda_\downarrow$ and $\mathbf{v}_\downarrow$ can be done in $O\left(n^2\right)$ time using iterative eigendecomposition methods such as using a single power iteration [94]. As such, Alg. 5 has a time complexity of $O\left(n^2\right)$. To show that Alg. 5 does indeed perform the correct projection, we prove the following theorem:

Algorithm 6 Efficient online Relative comparison Kernel LEarning (ERKLE)

1: $\mathbf{K}_0 \leftarrow \mathbf{I}$
2: $\hat{\lambda}_0^0 \leftarrow 1$
3: **for** $j = 1, 2, ...$ **do**
4: $\quad \gamma_j \leftarrow \delta_j f\left(\mathbf{K}_{j-1}, t_j\right)$
5: $\quad \mathbf{K}'_j \leftarrow \mathbf{K}_{j-1} - \gamma_j \mathbf{G}_{t_j}$
6: $\quad \hat{\lambda}_j^0 \leftarrow \hat{\lambda}_j^0 - 3\gamma_j$
7: $\quad$ **if** $\hat{\lambda}_j^0 < 0$ **then**
8: $\quad\quad \mathbf{K}_j \leftarrow \Pi_+^1\left(\mathbf{K}'_j\right)$
9: $\quad\quad \hat{\lambda}_j^0 \leftarrow \max\left(0, \lambda_\downarrow\right)$
10: $\quad$ **end if**
11: **end for**

Theorem 3 *Algorithm 5 results in a PSD matrix $\mathbf{K}_j$ that is closest to $\mathbf{K}'_j$ in terms of Frobenius distance.*

PROOF Let $\mathbf{K}_0 \in S_+^n$ (e.g. 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 $\gamma_j = \delta_j f\left(\mathbf{K}_{j-1}, t_j\right)$ be the *magnitude* of an update. By (5.3), the update in Equation (5.1) can be written as $\mathbf{K}_{j-1} - \gamma_j \mathbf{G}_{t_j}$. The only nonzero eigenvalues of $-\gamma_j \mathbf{G}_{t_j}$ are $\lambda_1 = 3\gamma_j$ and $\lambda_2 = -3\gamma_j$. It follows from Weyl's inequality that the matrix $\mathbf{K}'_j = \mathbf{K}_{j-1} - \gamma_j \mathbf{G}_{t_j}$ has *at most* one negative eigenvalue. If $\mathbf{K}'_j$ has no negative eigenvalues, then it is PSD (line 6 of Alg. (5)). If $\mathbf{K}'_j$ has one negative eigenvalue, line 4 of Alg. 5 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 [25].

The important implication of Thm. 3 is that ERKLE can incorporate a triplet into a kernel in $O(n^2)$ time by performing the efficient projection outlined in Alg. 5. Furthermore, if a step is sufficiently small, then no projection is needed at all. Let λ_j^0 be the smallest eigenvalue of $\mathbf{K}_j$. By Weyl's inequality, if $\lambda_j^0 - 3\gamma_j \geq 0$, 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 $\hat{\lambda}_j^0$. Initially, $\hat{\lambda}_0^0 \leftarrow \lambda_0^0$. It is updated each iteration with it's lower bound $\hat{\lambda}_j^0 \leftarrow \hat{\lambda}_{j-1}^0 - 3\gamma_j$. If $\hat{\lambda}_j^0 < 0$, then Alg. 5 is used to project onto the PSD cone and $\hat{\lambda}_j^0 \leftarrow \max\left(0, \lambda_\downarrow\right)$. 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 6 outlines the ERKLE algorithm. Here, we assume that our kernel is initialized to identity, but other initializations of the kernel and its smallest eigenvalue can be used. The for loop iterates for each time a new response t_j is obtained. First, the magnitude is found and used with the canonical gradient matrix to update the kernel. Next, the conservative estimate of the smallest eigenvalue is updated. If that estimate is negative, only then does ERKLE perform the efficient projection procedure to ensure the solution this iteration is PSD. The estimate is then updated to be the true smallest eigenvalue when projecting. Thus, after each iteration the kernel $\mathbf{K}_j$ is updated to model t_j and is PSD.

5.1.3 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 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 [15] 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 [95] suggests a more aggressive setting of $\delta_j = 1/\sqrt{j}$.

For our problem, however, we prefer to treat triplet responses equally: current responses should not have more influence than preceding responses. On the other hand, we do not wish to over-fit to the most recently obtained feedback. It is this observation that motivates our development of a Passive-Aggressive (PA) variant of our online RCKL framework. In the RCKL setting, the general idea of passive-aggressive learning is that if the previous solution $\mathbf{K}_{j-1}$ satisfies a newly obtained triplet response $t_j = (a, b, c)$ 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.

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 (5.5) 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 $t_j = (a, b, c)$. As such, we wish to find a δ_j that takes an aggressive step. We do this by solving the following optimization problem:

$$\begin{aligned} \min_{\delta_j} \quad & \delta_j^2 \\ \text{s.t.} \quad & d^2_{\mathbf{K}'_j}(a, b) + 1 \leq d^2_{\mathbf{K}'_j}(a, c), \delta_j \geq 0 \end{aligned} \tag{5.6}$$

By (5.3) and (5.5), the first constraint can be rewritten as:

$$d^2_{\mathbf{K}_{j-1}}(a, b) - d^2_{\mathbf{K}_{j-1}}(a, c) - 10\delta_j + 1 \leq 0 \tag{5.7}$$

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 (5.7), making the positive constraint on δ_j redundant. Also, the smallest δ_j that satisfies (5.7) 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 $\mathcal{L}(\delta_j, \alpha)$:

$$\delta_j^2 + \alpha \left(d^2_{\mathbf{K}_{j-1}}(a, b) - d^2_{\mathbf{K}_{j-1}}(a, c) - 10\delta_j + 1 \right) \tag{5.8}$$

Taking the partial derivative of (5.8) with respect to δ_j, setting it to 0, and solving for δ_j results in $\delta_j = 5\alpha$. Substituting this back into (5.8) makes the Lagrangian:

$$-25\alpha^2 + \alpha \left(d^2_{\mathbf{K}_{j-1}}(a, b) - d^2_{\mathbf{K}_{j-1}}(a, c) + 1 \right) \tag{5.9}$$

Taking the partial derivative of (5.9) 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:

$$\delta_j = \frac{d^2_{\mathbf{K}_{j-1}}(a, b) - d^2_{\mathbf{K}_{j-1}}(a, c) + 1}{10} \tag{5.10}$$

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

$$\begin{aligned} \min_{\delta_j} \quad & \delta_j^2 \\ \text{s.t.} \quad & p_{abc}^{\mathbf{K}'_j} \geq P, \delta_j \geq 0 \end{aligned} \tag{5.11}$$

Algorithm 7 ERKLE with Multiple Passes

Input: β : # of triplet responses stepped over each pass

$\mathbf{K}_0 \leftarrow \mathbf{I}$
for $j = 1, 2, ...$ **do**
  $\mathbf{K}'_j \leftarrow \mathbf{K}_{j-1} - \delta_j \nabla l\left(\mathbf{K}_{j-1}, t_j\right)$
  $\mathbf{K}_j \leftarrow \Pi^1_{S_+}\left(\mathbf{K}'_j\right)$
  if $j > 2\beta$ **then**
    for $k = 1, 2, ..., \beta - 1$ **do**
      Randomly select t' from $\{t_1, t_2, ..., t_j\}$
      $\mathbf{K}'_j \leftarrow \mathbf{K}_j - \delta_{j+k} \nabla l\left(\mathbf{K}_{j-1}, t'\right)$
      $\mathbf{K}_j \leftarrow \Pi^1_{S_+}\left(\mathbf{K}'_j\right)$
    end for
  end if
end for

In (5.11) 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 (5.11), we derive the following step size:

$$\delta_j = \frac{d^2_{\mathbf{K}_{j-1}}(a, b) - d^2_{\mathbf{K}_{j-1}}(a, c) + \kappa}{10} \tag{5.12}$$

where $\kappa = \log(P) - \log(1 - P)$. The full derivation is given in Appendix B. Both derivations reveal that passive-aggressive updates using STE and GNMDS are similar. Setting $P = \frac{e}{1+e}$ in (5.12) recovers the GNMDS passive-aggressive step in (5.10), and changing the margin in (5.10) recovers different settings of P.

Note that using (5.10) 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.

5.1.4 ERKLE with Multiple Passes

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 as a form of averaging [14, 109, 150]. For our problem setting, this indicates that ERKLE may benefit from revisiting triplet responses 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 triplet responses from the set of previously obtained responses.

Algorithm 7 outlines our multiple-pass version of ERKLE. We denote the number of "passes" ERKLE performs each time a new triplet response is observed as β. For brevity, we omit the portion of the algorithm that maintains and checks the estimate of the smallest eigenvalue, but it can be inserted into lines 4 and 9 to skip unnecessary projections. Here, after a sufficient number of triplets have been obtained (in our experiments, we chose 2β), $\beta - 1$ triplet responses are selected every iteration from all previously observed responses (for a total of β updates per iteration). These triplet responses are stepped over as done with original ERKLE algorithm. For our random selection used in our experiments, we simply selected uniformly at random with replacement from the obtained responses. More sophisticated random selection procedures may be used in order ensure triplet responses obtained initially do not get selected drastically more times than those obtained later. For instance, when a triplet response gets chosen on line 7, one could reduce the probability of that response being chosen subsequently. Using this simple approach is sufficient for ERKLE to maintain high accuracy while still ensuring computational efficiency.

5.2 EXPERIMENTS

In this section, we evaluate ERKLE by comparing it to batch RCKL methods. While batch methods can be applied directly in the online learning setting, they more commonly use mini-batches when dealing with continuously streaming data. In the mini-batch RCKL setting, every time a new batch of m triplet responses are received, batch RCKL is trained over all responses obtained so far (e.g.

if $m = 100$ after two mini-batches are received, then the batch methods are trained using 200 responses). Thus, after all responses 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 triplet responses over popular music artists is used to evaluate how the methods perform in a real-world setting with moderate response noise. Finally, ERKLE and batch RCKL methods are evaluated on a data set of responses over scene images, which consist of a small number of responses, thus focusing on the performance of these methods with very little feedback.

For these experiments, we wish to evaluate the model effectiveness of the solutions each method finds as a function of the amount of feedback obtained. For RCKL, this is measured by how the learned kernels generalize to held out triplet responses, as responses are obtained. This is important in real-world applications where the goal is to accurately model many or even all the relationships among objects, not just the observed ones. Because of this, we again use triplet query prediction error (TQPE) (4.14), as it effectively measures how well each method can utilize the obtained (training) responses to generalize to unseen (test) responses.

Unless otherwise noted, the experiments were run with 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 triplet responses). 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 responses.

We used the batch STE, GNMDS, and CKL (Crowd Kernel Learning [132]) MATLAB implementations specified by [143] 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 λ hyperparameter for STE and GNMDS, and μ for CKL was chosen to be the best performing setting over ten varying options. The timed experiments were performed on an Intel Core i5-4670K CPU

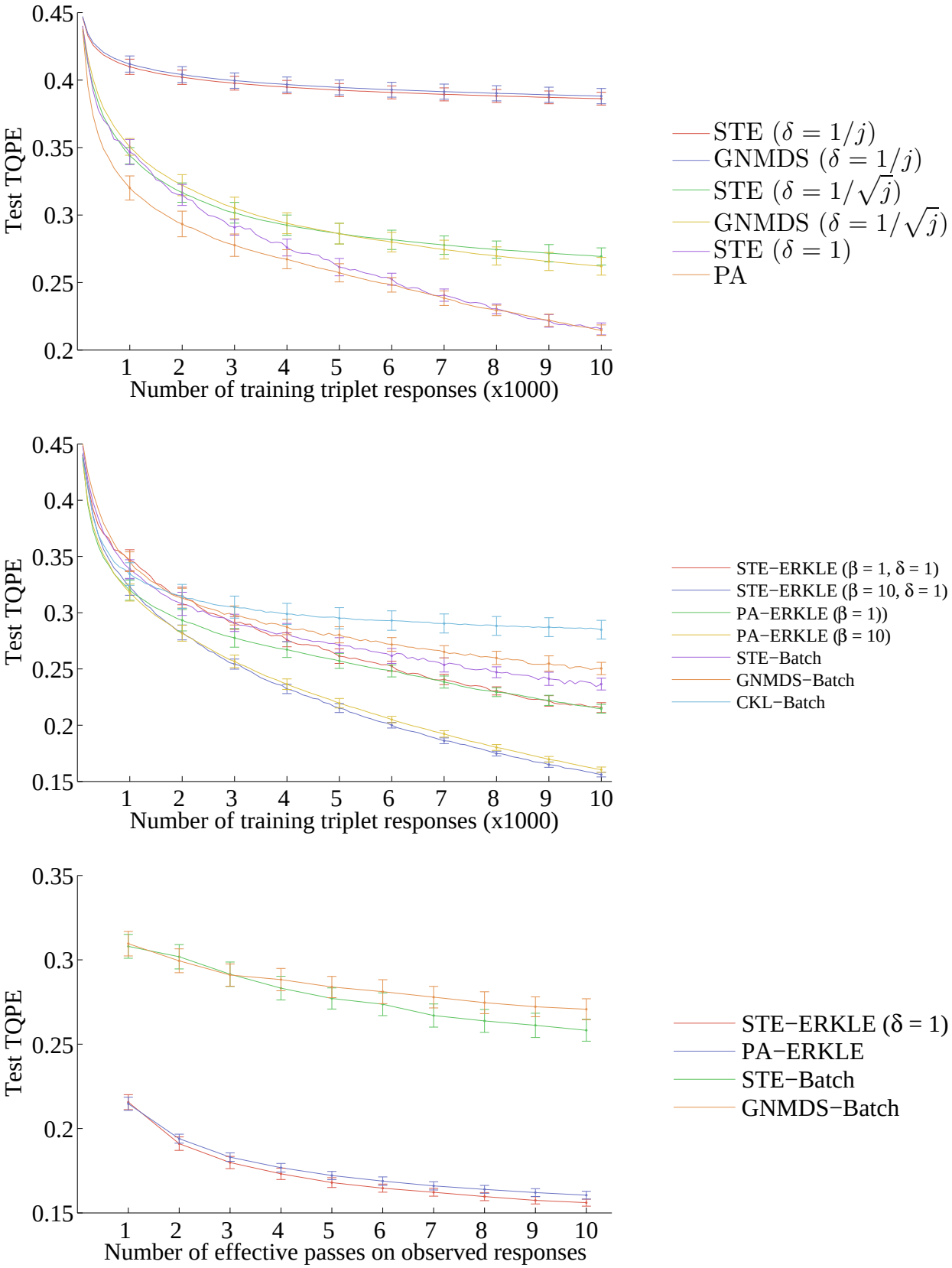


Figure 17: ERKLE small-scale synthetic data experiments (ten trials, 95% CI)

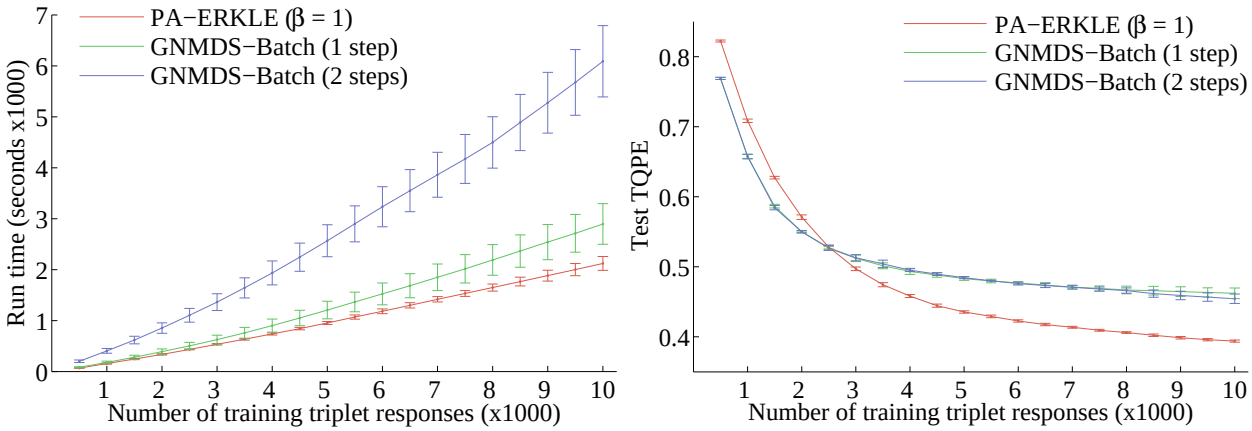
@ 3.4 GHz with 16 GB of RAM on a single thread. Each experiment was performed with ten trials, each with different, randomly chosen test, train and validation sets. The error bars in the graphs represent the 95% confidence interval.

5.2.1 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 triplet queries, resulting in 485,100 triplet responses. 10,000 responses were used as the train set and the rest were used as the test set.

Discussion: Figure 17 shows three graphs depicting the results of our experiment on small-scale synthetic data. The top figure shows the effect that the learning rate parameter δ_j has on the performance of ERKLE as more triplet responses 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 responses. STE-ERKLE with a constant learning rate and PA-ERKLE take steps solely based on how well the current solution satisfies the observed responses, and vastly outperform the alternative learning rates based on number of observations. This result indicates that reducing the influence of a response because it was observed later has an adverse effect on the ability of a learned kernel to generalize to unobserved responses.

The middle figure shows the performance of STE-ERKLE (with δ_j set to 1), and PA-ERKLE compared to three batch RCKL methods. The batch hyperparameters were chosen by selecting the best settings 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 triplet responses 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.



(a) Run time (seconds) vs. # of observed responses

(b) Test TQPE vs. # of observed responses

Figure 18: ERKLE large-scale synthetic data experiments (five trials, 95% CI)

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 triplet responses with the step size δ_j validated on the test set for the batch methods. The bottom figure 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.

5.2.2 Large-Scale Synthetic Data

Next, we evaluated how PA-ERKLE compared to batch GNMDS in terms of practical 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 five trials, 10,000 randomly generated triplet responses 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 responses due to time constraints. The batch hyperparameters and the step size δ_i in their optimizations were chosen as the settings that best performed on the test set.

Discussion: Figure 18a 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 λ and

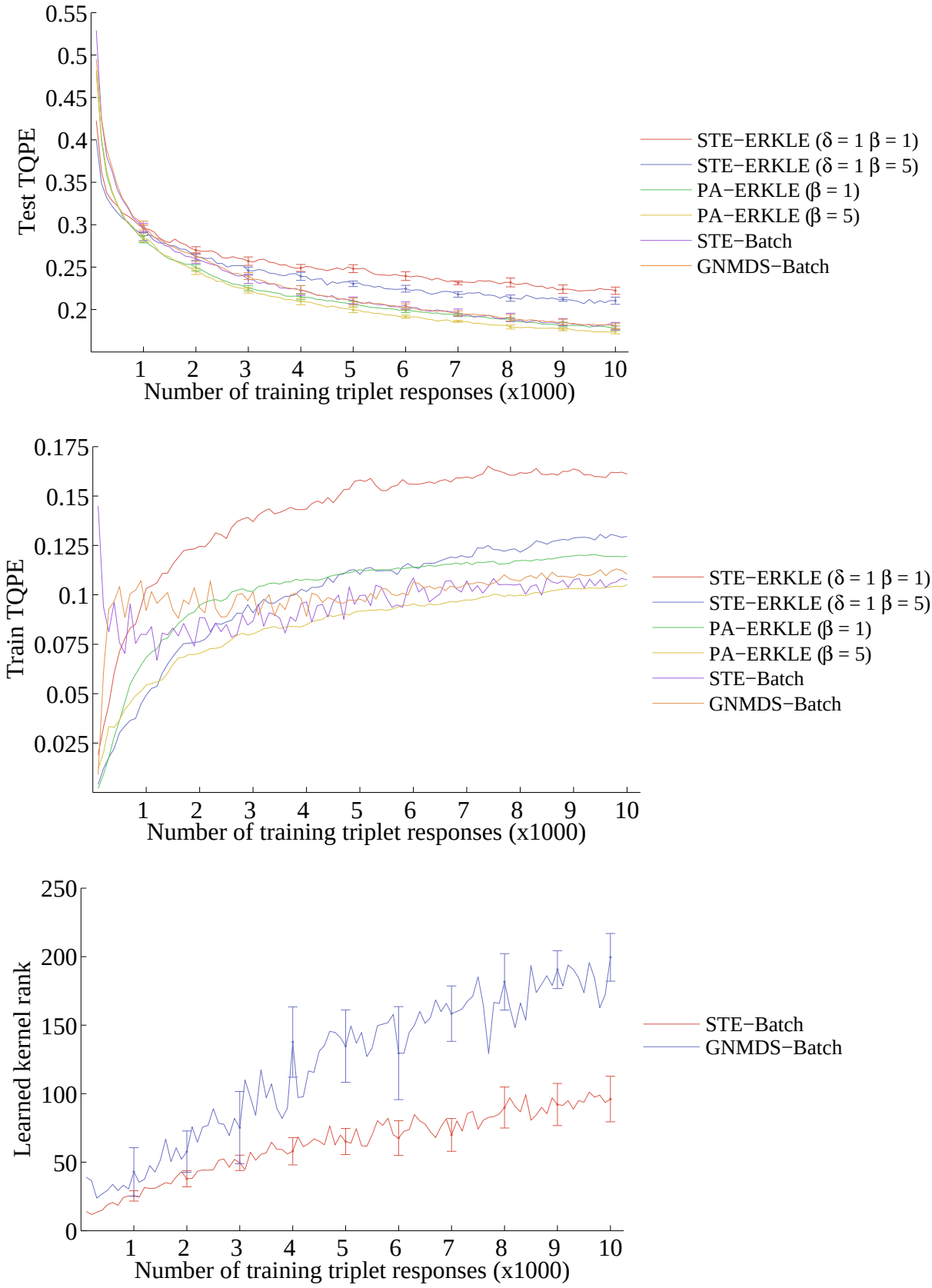


Figure 19: Results from experiments on the aset400 data set (ten trials)

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 five 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 18b depicts the test errors of each method. Initially, the batch methods perform better, but at around 2,500 triplet responses, 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 in terms of generalization and having faster run time.

5.2.3 Music Artist Similarity

For the last two experiments we performed evaluations on real-world data sets. First, we performed an experiment using triplet responses among popular music artists gathered from a web survey. The *aset400* data set [40] contains 16,385 responses over 412 artists. We randomly chose 10,000 responses as the train set, 1,000 as the validation set for the batch hyperparameters, 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 responses, thus methods used in the evaluation must deal with noise within the data.

Discussion: Figure 19 shows the results of the *aset400* data experiments. The top figure shows how well 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 triplet responses. The probability $p_t^{\mathbf{K}}$ used in STE-ERKLE decays rapidly. Thus, responses that are in agreement with previously obtained responses do not influence the learned kernel greatly. However, a conflicting responses 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 response. Because of this, PA-ERKLE performs as well as the batch methods and often better when multiple passes are taken.

The middle figure shows the training errors of each method. We use normalized training error

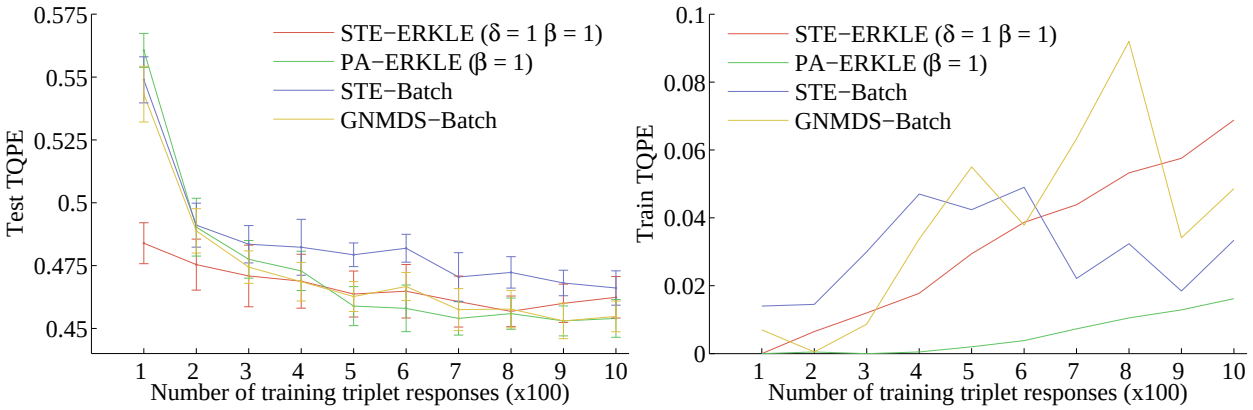
as an objective-independent measure of how well each method fits to the observed triplet responses. The STE-ERKLE models are greatly effected by the presence of conflicts in that they do not learn a kernel that fits to a large number of the observed responses. PA-ERKLE, on the other hand, is able to fit better to the set of observed responses.

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 the bottom figure in Fig. 19 we plot the ranks of the kernels learned by the batch methods. In our experiments, the range of potential λ 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 triplet responses, but as the number of responses 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 responses to the necessary extent.

5.2.4 Outdoor Scene Similarity

Our final experiment used triplet responses over 200 randomly chosen images of scenes from the Outdoor Scene Recognition (OSR) data set [99]. Triplet queries were posed to 20 people via an online system. After an initial 1200 randomly chosen queries were answered , 20 "rounds" of 200 queries were chosen to be answered according to the adaptive selection criterion in [132], resulting in 3,600 total responses. For each trial of this experiment, 1,000 responses were randomly chosen as the test set, 1,000 as the train set, and 600 as the validation set for the batch hyperparameters. This experiment is especially challenging for two reasons. First, this is the smallest experiment in terms of responses, highlighting how the methods perform with little feedback. In addition, the adaptive selection algorithm chooses queries with the highest information gain, hence, the responses are intentionally chosen to give disparate information about how the objects relate.

Discussion: Figure 20a 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 triplet responses relative to all other methods. PA-ERKLE is comparable or outperforms



(a) Test error vs. # of observed responses

(b) Train error vs. # observed responses

Figure 20: Results from experiments on the OSR data set (ten trials)

its batch counterpart in GNMDS-Batch, given enough responses (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 responses.

5.3 SUMMARY

In this chapter, we introduced a novel method to learn a PSD kernel matrix modeling the relationships among objects from relative triplet comparisons given in an online fashion. This task, motivated by real-world applications where feedback is obtained continuously, cannot practically be performed by traditional batch RCKL methods due to their PSD projection procedures having time complexity of $O\left(n^3\right)$. 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 $O\left(n^2\right)$. We show how passive-aggressive online learning benefits our method in terms of generalizing to unseen triplet responses, 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. Thus, our method, ERKLE, is able to achieve increased computational efficiency while maintaining the model effectiveness of batch methods in the online RCKL setting.

6.0 METRIC LEARNING FROM AUXILIARY CONFIDENCE LABELS WITH APPLICATION TO CLINICAL DECISION MODELING

While learning a perceptual model of similarity has practical use in many applications, certain applications require a model tailored to the specific task trying to be accomplished. In this chapter, we focus on an application in clinical medicine. Consider the application of developing a *Clinical Decision Support System* (CDSS), a computer system that uses data to aid clinicians in making treatment decisions. In their more basic forms, CDSSs simply act as a portal for clinicians to access relevant information. However, CDSSs are often developed to perform much more sophisticated tasks such as providing suggested treatment options or warning of dangerous drug interactions. For a CDSS to accomplish such inference tasks, it requires a meaningful model of how previously observed patients relate to new patients. To build such a model, data regarding previous patients and task-specific supervision on those patients is required. Fortunately, *Electronic Health Records* (EHRs) are being adopted by more and more health care providers [67, 24]. EHRs provide data that uniquely characterizes different patients in an easily accessible form. For supervision, clinicians themselves can provide quality feedback if explicitly prompted for it. By combining these two sources of information, an insightful inference model of patients can be built using supervised learning techniques.

Much of the previous work in creating patient models from supervision leverage standard classification methods [55, 39, 41, 129]. Here, feedback appears in the form of class labels (e.g. the patient is at risk for a condition or not), and the learned inference models output a predicted class label when given an unseen patient. For CDSSs, these predictions can be used to alert clinicians of important information that supports decision making. However, there are practical issues with standard classification models for use in CDSSs. First, it is vital that a clinician is able to understand how a CDSS comes to conclusions [12]. Otherwise, the clinician may not trust the model due to lack

of clear reasoning. Many standard classification methods focus solely on maximizing some measure of classification accuracy without any focus on learning a model that can be easily interpreted by humans. Consequently, clinicians may not be able to understand why they are being alerted, even the classifier is accurate. For this task in particular, the model effectiveness of a learning method not only means that classifiers it produces should be accurate, but also that they should be interpretable.

Another practical concern lies in the cost of obtaining sufficient feedback to learn an accurate classification model. Because the expertise of a clinician is valuable, the cost of obtaining clinical feedback is substantially more than obtaining feedback from the layman. Compounding this cost is that clinicians must spend a substantial amount of time to consider multiple, interacting factors before providing feedback. If standard classification methods are to be used, clinicians would be prompted for only a single class label after considering a single patient. However, class labels convey only a simple notion of how patients relate, despite the fact that clinicians have more in-depth knowledge about the patient that they could provide. Thus, often a large amount of labeled instances needed to learn accurate classifiers for more complex inference tasks. All of these factors together motivate the need for a more human efficient way of learning a classifier.

In consideration of these issues, we propose a novel metric learning method called **C**onfidence-b**A**sed **ME**tric **L**earning (CAMEL). CAMEL was designed with a specific emphasis on learning human-interpretable models of patients. Our method produces sparse models that use only the relevant patient information when modeling disease. As a result, clinicians can easily identify which features are used when making inferences. Also, metrics produced by CAMEL naturally induce confidence scores that indicate how confident the model is when making inferences. Clinicians can take these scores into consideration when making important treatment decisions. Finally, our method learns a parametric metric that projects patients into multidimensional space of disease where each dimension in the learned metric space is as a separate "factor" in how the model reasons about patients. This contrasts with many standard classification methods that project data objects to a single dimension. Clinicians can view what features influence these factors and interpret how the model is making inferences.

To reduce the cost of obtaining expert supervision, we formulate a version of CAMEL that can leverage *confidence labels* that indicate how sure a labeler is in a given class label. Our use of confidence labels is motivated by the fact that when tasked with providing supervision, clinicians

spend their time mostly on considering the patient EHRs themselves. Once they learn what is needed to produce a class label for a patient, providing a confidence label requires a relatively short amount of additional time and effort. If this confidence information provides additional, useful insight into how patients relate to classes, then our method that utilizes confidence labels, CAMEL-CL, is able to learn more accurate classification models with fewer labeled patients. In doing so, CAMEL-CL can be used to reduce the effort required from the labeler, and in turn, decrease the cost of obtaining expert feedback. [1]

6.1 PREVIOUS WORK USING AUXILIARY CONFIDENCE LABELS

Two previous works have considered auxiliary labels similar to the confidence labeled considered in our problem setting. Both use *probabilistic labels* that indicate how likely an object belongs to a class. The work that introduced this problem [97, 98] formulated various methods to solve it. One of the more successful of these was one that leverages the popular Support Vector Machine (SVM) framework. Their method learns linear classifier by solving an optimization problem that balances two energies: standard SVM classification hinge loss and a function that encourages the model to rank the objects by their probabilistic labels. By including the ranking energy, their method can produce an SVM in which objects with high probabilistic labels are farther from the decision hyperplane than those with low probabilistic labels. While our method uses a similar ranking energy, it is based on an intuitive, explicitly defined confidence score. In addition, their learned model is a linear transformation into a single dimension, and our model is a more expressive multidimensional metric. Finally, our method induces sparsity in order to improve accuracy and enhance interpretability, while their method learns a dense model. Both [97] and [98] contain experimental evaluation on clinical data sets. However, they provide no analysis into what can be interpreted from their model, opting only to show results pertaining to accuracy of inference. Because this method is most similar to our work in both methodology and application, we compare our methods to SVM-Combo from [98] in Sec. 4.1.4.

Later work [104] considers the probabilistic labels simply as regression labels, and applies

[1] The material presented in the chapter was originally published as [61].

Gaussian Process Regression [108]. For inference, they apply a threshold at 0.5 to predicted regression labels to determine class membership. The authors were able to bound the error of their probabilistic predictions as a function of a parameter meant to model noise in the given probabilistic labels. In their experiments, they evaluate their method on synthetic and real-world data sets. Unfortunately, none of these clearly matches our specific problem setting. Their real-world experiment derives probabilistic labels from movie ratings for which there is no clear class label associated with movie instances.

6.2 METHODOLOGY

We begin by more formally defining our problem setting. Let $\mathcal{D} = \{(\mathbf{x}_1, y_1, c_1), ..., (\mathbf{x}_n, y_n, c_n) \in (\mathcal{X}, \mathcal{Y}, \mathcal{C})^n\}$ be a set of observed data. Let $\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^m\}$ be a collection of n data objects represented by m-dimensional real vectors. Let $\mathcal{Y} = \{y_1, ..., y_n \in \{0, 1\}\}$ and $\mathcal{C} = \{c_1, ..., c_n \in [0, 1]\}$ be binary and confidence labels, respectively, that correspond to data objects. In our problem setting, each $\mathbf{x}_i$ is a patient instance represented by features drawn from EHR data. The corresponding y_i is a class label gathered from a clinician (e.g. a positive or negative diagnosis), and c_i is a confidence label indicating how confident she is in y_i. In this work, we consider binary class labels and confidences in $[0, 1]$, though much of the subsequent can easily be extended to other settings.

We wish to learn a metric parameter $\mathbf{L}$ as in (2.6) from observed patient data so that the resulting metric can be used to accurately predict the class labels of unobserved patient instances. To this end, we begin by defining a measure of similarity between objects, given $\mathbf{L}$:

$$k_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-d^2_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_j)\right) \tag{6.1}$$

Equation (6.1) is an application of the Gaussian kernel function as it appears in (2.11). Traditionally, the Gaussian kernel function is parameterized by a bandwidth that influences how quickly similarities decay towards zero. This parameter greatly affects the performance of the methods in which the Gaussian kernel is used, and needs to be validated for use in traditional classification methods. In (6.1), the bandwidth parameter is absorbed into the learned parameter $\mathbf{L}$. As a result, our method

not only learns a transformation, but also the bandwidth of a Gaussian kernel, similar to what is done in Metric Learning for Kernel Regression.

With this measure of similarity, we can define the relationship a patient instance has with others. Most importantly, we can define how similar an patient is to those with class label y:

$$S_{\mathbf{L}}^{y}\left(\mathbf{x}_i\right) = \frac{1}{|\mathcal{X}_{\mathbf{x}_i}^{y}|} \sum_{\forall \mathbf{x}_j \in \mathcal{X}_{\mathbf{x}_i}^{y}} k_{\mathbf{L}}\left(\mathbf{x}_i, \mathbf{x}_j\right) \tag{6.2}$$

Here, $\mathcal{X}_{\mathbf{x}_i}^{y} = \{\mathbf{x}_j \in \mathcal{X} : y_j = y \wedge \mathbf{x}_j \neq \mathbf{x}_i\}$. In essence, (6.2) is the mean similarity $\mathbf{x}_i$ has with all observed objects with label y, excluding itself. This *similarity score* measures how similar a patient is to observed patient instances of a single class. However, this score is independent of a object's relationship to other classes. For this, we formulate a *confidence score*:

$$C_{\mathbf{L}}^{y}\left(\mathbf{x}_i\right) = \frac{S_{\mathbf{L}}^{y}\left(\mathbf{x}_i\right)}{S_{\mathbf{L}}^{y}\left(\mathbf{x}_i\right) + S_{\mathbf{L}}^{\tilde{y}}\left(\mathbf{x}_i\right)} \tag{6.3}$$

In the binary class case, $\tilde{y}$ is zero if y is one, and one if y is zero (the complement of y). In the multi-class case, $\tilde{y}$ is all class labels other than y. Equation (6.3) can be interpreted as a class conditional probability that an object is a member of a class given a metric parameter $\mathbf{L}$. As such, confidence scores define both a criteria for learning an $\mathbf{L}$ that fits to observed data, and a way to predict class labels on unobserved patient instances once $\mathbf{L}$ is learned. Like other classifiers that define conditional probability functions, inference can be done by putting a threshold on the confidence score of an unobserved instance (e.g. if $C_{\mathbf{L}}^{1}\left(\mathbf{x}_i\right) > 0.4$, then the predicted label $\hat{y}_i$ is one, otherwise, it is zero). In addition, area under the Receiver Operating Characteristic curve (AUROC) can be found using confidence scores on patient instances.

In order to find a metric that models class membership of observed patient instances well, one could maximize $C_{\mathbf{L}}^{y_i}\left(\mathbf{x}_i\right)$ directly for all observed patients. However, doing so leads to a difficult non-convex optimization problem. To avoid this source of non-convexity, we maximize an approximation of the confidence scores for the observed data:

$$\max_{\mathbf{L}} \sum_{i=1}^{n} S_{\mathbf{L}}^{y_i}\left(\mathbf{x}_i\right) - S_{\mathbf{L}}^{\tilde{y}_i}\left(\mathbf{x}_i\right) \tag{6.4}$$

In (6.4) we maximize the similarity each observed data object has with its observed class label, while minimizing the the similarity it has with the opposite class. The objective "pulls" all observed

objects of the same class towards each other, and "pushes" all objects of different classes away, similar to Large Margin Nearest Neighbors. In doing so, (6.4) increases the numerator of (6.3), while decreasing a term the denominator, and by doing so, approximates learning a metric where the confidence scores of the observed data is high.

Unfortunately, (6.4) can result in solutions where observed objects of the same class are projected to nearly the same point in the metric space, while observed objects of different classes are infinitely far apart. This leads to models that can drastically over-fit to the observed data. To combat this, we include ℓ-1 norm regularization into the objective:

$$\min_{\mathbf{L}} \sum_{i=1}^{n} \left(S_{\mathbf{L}}^{\tilde{y}_i} \left(\mathbf{x}_i \right) - S_{\mathbf{L}}^{y_i} \left(\mathbf{x}_i \right) \right) + \lambda \left\| \mathbf{L} \right\|_1 \tag{6.5}$$

Note that (6.5) is a minimization problem, where (6.4) is a maximization problem. We simply multiplied the objective in (6.4) by -1 to turn it into an equivalent minimization problem before adding the regularization term. Here, the ℓ-1 norm is taken element-wise on $\mathbf{L}$, that is $\left\| \mathbf{L} \right\|_1 = \sum_{i=1}^{m'} \sum_{j=1}^{m} |\mathbf{L}^{i,j}|$. Higher settings for the hyperparameter λ force elements of $\mathbf{L}$ to exactly zero, preferring sparse solutions to more dense ones. More sparse solutions may then be found that fit less to the observed data, thus reducing the risk of over-fitting. This also has a more practical benefit. Tens, hundreds, even thousands of features can be extracted from EHR data. It can be difficult to tell, a a priori, which are useful to model patient relationships. If many features are used to represent patients (m is large) and the learned model is dense, then a clinician has to consider many, potentially irrelevant, features to understand how the model is making decisions. If $\mathbf{L}$ consists of a large number of zeros, then the learned metric only utilizes a few features, giving the clinician a concise model to interpret. We call our gradient descent method to solve (6.5) CAMEL.

6.2.1 Incorporating Confidence Labels

CAMEL attempts to maximize the correct class confidence score for each training patient using only class labels. As stated previously, we wish to also use confidence labels provided by clinicians to reduce the overall cost of obtaining expert supervision. The most obvious way of incorporating confidence labels would be to ensure that the confidence score for an observed patient matches the confidence label the clinician provides. However, in practice, confidence labels tend to contain a

great deal of noise. It has been shown that humans tend to find it difficult to accurately produce exact numerical assessments on objects and are much better suited to provide simpler forms of feedback such as class labels or relative comparisons [135, 130]. Because of this, bolstering CAMEL with the exact values of the confidence labels can introduce unwanted noise.

Instead, we choose to simplify the labels by assuming that, while the exact value of a confidence label is noisy, its value compared to others of the same class is not (or at least reasonably less noisy). For instance, if a clinician gives labels $c_a = 0.65$ and $c_b = 0.95$ we only take that to mean that the labeler is more confident in y_b than y_a with no consideration into by how much. By this assumption, we create a ranking $\mathcal{R}_\mathcal{C}$ of patients such that $(\mathbf{x}_a, \mathbf{x}_b) \in \mathcal{R}_\mathcal{C}$ if and only if $c_a > c_b$ and $y_a = y_b$. From $\mathcal{R}_\mathcal{C}$ we can induce a set of constraints to be imposed on (6.5) that allows us to incorporate the information contained in the confidence labels:

$$\begin{aligned} \min_{\mathbf{L}} \quad & \sum_{i=1}^{n} \left(S_{\mathbf{L}}^{\tilde{y}_i}(\mathbf{x}_i) - S_{\mathbf{L}}^{y_i}(\mathbf{x}_i) \right) + \lambda \|\mathbf{L}\|_1 \\ \text{s.t} \quad & \forall_{(\mathbf{x}_a, \mathbf{x}_b) \in \mathcal{R}_\mathcal{C}} \left(S_{\mathbf{L}}^{y_a}(\mathbf{x}_a) - S_{\mathbf{L}}^{\tilde{y}_a}(\mathbf{x}_a) \right) > \left(S_{\mathbf{L}}^{y_b}(\mathbf{x}_b) - S_{\mathbf{L}}^{\tilde{y}_b}(\mathbf{x}_b) \right) \end{aligned} \tag{6.6}$$

The added constraints ensure that the approximated confidence score for $\mathbf{x}_a$ is greater than $\mathbf{x}_b$ for all $(\mathbf{x}_a, \mathbf{x}_b) \in \mathcal{R}_\mathcal{C}$. In other words, it tries to ensure the confidence scores adhere to $\mathcal{R}_\mathcal{C}$. In practice, it is unlikely that all of the constraints can be satisfied, so we opt to solve a similar, unconstrained optimization:

$$\begin{aligned} \min_{\mathbf{L}} \quad & \sum_{i=1}^{n} \left(S_{\mathbf{L}}^{\tilde{y}_i}(\mathbf{x}_i) - S_{\mathbf{L}}^{y_i}(\mathbf{x}_i) \right) + \lambda_1 \|\mathbf{L}\|_1 \\ & + \lambda_2 \sum_{(\mathbf{x}_a, \mathbf{x}_b) \in \mathcal{R}_\mathcal{C}} [S_{\mathbf{L}}^{\tilde{y}_a}(\mathbf{x}_a) - S_{\mathbf{L}}^{y_a}(\mathbf{x}_a) - S_{\mathbf{L}}^{\tilde{y}_b}(\mathbf{x}_b) + S_{\mathbf{L}}^{y_b}(\mathbf{x}_b)]_+ \end{aligned} \tag{6.7}$$

The last term in (6.7) is zero when the corresponding constraint in (6.6) is satisfied and positive when it is not. In short, if an observed object should have a higher confidence score than others because the labeler is more confident in its class label, that object's contribution to the objective is increased by a factor of λ_2 for each observed object its confidence score should be higher than. If an observed object's confidence score is too high, its contribution is similarly decreased by a factor of λ_2.

By introducing the ranking term into the objective, we also introduce an additional hyperparameter λ_2. Much like higher values of λ_1 increase the influence of regularization in the objective,

Expert	# Pos	# Neg	Mean CL Pos (STD)	Mean CL Neg (STD)
1	76	473	0.583 (0.151)	0.294 (0.089)
2	143	428	0.656 (0.377)	0.114 (0.202)

Table 1: Summary statistics for expert-labeled HIT data

higher values of λ_2 put a heavier emphasis on ordering the confidence scores according to $\mathcal{R}_{\mathcal{C}}$. As such, care must be taken to properly set the hyperparameters to balance fit to the class labels, fit to the confidence labels, and sparsity. We call our gradient descent method for solving (6.7) CAMEL-CL.

6.3 EXPERIMENTS

To evaluate CAMEL and CAMEL-CL, we performed experiments on real-world clinical data, the results of which we discuss in this section. We begin by first describing the data set used. Then, we outline how the experiments were performed. Next, we present and discuss quantitative results comparing both CAMEL methods to related, current methods. Finally, we qualitatively analyze the models learned by CAMEL by looking at how it uses patient data to make inferences.

6.3.1 Data Set Description

Our experiments were performed on data extracted from the Post-Surgical Cardiac Patient (PCP) Database in combination with supervision provided by clinicians indicating whether a patient is at risk for Heparin-Induced Thrombocytopenia (HIT). A lengthy description of this data can be found in [98] and further specifics can be found in [57, 142, 56]. Here, we provide a shorter summary of our view of the data. From the PCP Database 4,486 unique EHRs were chosen. From these over 51,000 patient-state instances were extracted using 24-hour segmentation. Uniformly random sampling from the pool of patient instances would result in an overwhelmingly disproportionate number of negative labels for HIT. Because there was a finite budget in obtaining supervision,

Name	Supervision	Hyperparameters	Implementation	Brief Description
Orig	None	None	Our own	Gaussian kernel (bandwidth = 1) applied to original feature space
SVM	Class Labels	Weight on hinge-loss	LIBLINEAR [43]	Standard linear SVM classification
LMNN	Class Labels	Number of nearest neighbors	From author's website	Metric learning method for nearest neighbor classification
CAMEL	Class Labels	λ	Our own	Our algorithm for solving (6.5)
LASSO	Confidence Labels	Weight on ℓ-1 norm	MATLAB Stats & ML	ℓ-1 norm regularized linear regression
MLKR	Confidence Labels	Dimension of metric projection	From author's website	Metric learning for kernel regression
SVM -Combo	Class & Confidence Labels	Weight on classification loss and weight ranking loss	Provided by author	Standard linear SVM classification with penalty that enforces ranking of patients by confidence labels
CAMEL -CL	Class & Confidence Labels	λ_1 and λ_2	Our own	Our algorithm for solving (6.7)

Table 2: Methods used in CAMEL experimental evaluation

a stratification procedure was used to bias sampling towards patients that could be at risk for HIT. Using this procedure, patient instances were chosen to be labeled by three experts in clinical pharmacology.

The experts were asked two questions for each patient instance: "How strongly does the clinical evidence indicate that the patient is at risk of HIT?" and "Assume you have received an HIT alert for this patient. To what extent you agree/disagree with the alert?". For the first question, the experts were prompted for a number between 0 and 100, which we normalize to $[0, 1]$ and use as confidence labels. For the second question, the experts were prompted to for one of four ordinal categories ranging from "strongly agree" to "strongly disagree" from which we derive binary class labels between the "agree" and "disagree" categories. Both our experiments and the results reported in [98] indicate that the confidence labels provided by the third expert are prohibitively noisy. To more properly showcase the potential utility of confidence labeling, we omit lengthy discussion of results from experiments using the third expert's supervision, and note that all methods that utilize the third expert's confidence labels alone perform poorly. Furthermore, the two methods that utilize both class labels and confidence labels gain no benefit from the confidence labels. Statistics summarizing the two other experts' feedback used in our evaluations (number of positively labeled patients, mean confidence labels, etc.) can be found in Table 1.

From the EHRs of the selected patient instances, 50 features were extracted to form feature vectors characterizing each patient. These features measure both trends and static measurements in

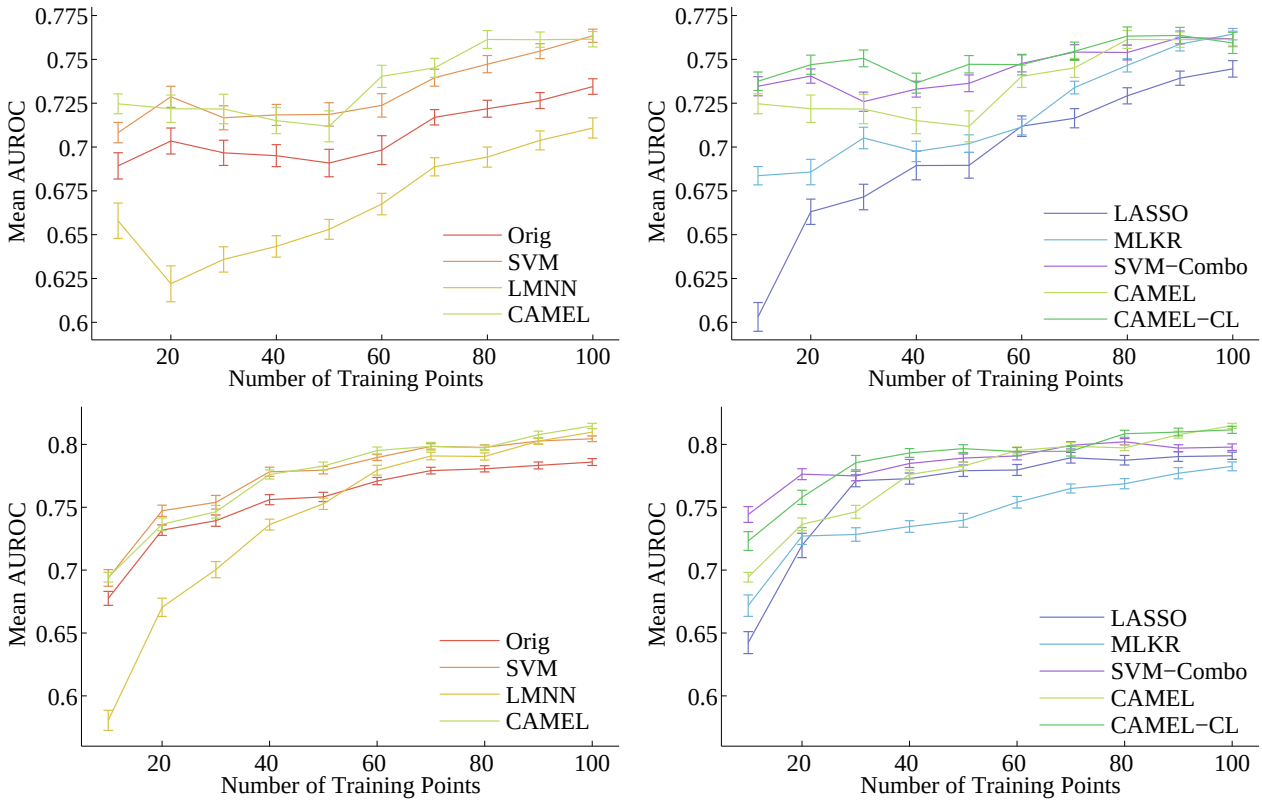


Figure 21: CAMEL HIT data experiments: number of training instances vs. AUROC on test set (Row 1 = Expert 1, Row 2 = Expert 2, ten trials, 95% CI)

one of five *attributes*: Heparin administration record (features 1-4), hemoglobin count (5-18), white blood cell count (19-31), platelet count (32-45), and major heart surgeries (46-49). For example, platelet count features include: "latest platelet value taken", "difference between last two platelet values taken", and "overall trend in platelet values".

6.3.2 Experimental Methodology

The methods used in this evaluation are listed in Table 2. These methods were chosen as a sample of current techniques that learn linear models or metrics from one or both forms of supervision considered in this work. The "Orig" method uses no supervision and provides us with a rudimentary baseline in our evaluation. All methods can produce confidence scores: The classification models can be interpreted to have uncertainty measures in class predictions, and the real-valued predictions

from the regression models can be taken as confidences. Thus, we evaluate the accuracy of each method using the AUROC of predictions on a held-out test set.

We performed separate but identical experiments on each experts' supervision. From the pool of selected patient instances we randomly selected 100 patients to be the train set, and split the remaining patients randomly into evenly-sized test and validation sets. This was done 20 times to form 20 trials. For each trial, an increasing number of the 100 training points were used to train the models in the evaluation (10, 20,...100). We did this to assess each method as a function of the amount of obtained supervision. For each training partition, hyperparameter settings for each method were chosen to be those that maximized AUROC on the validation set.

6.3.3 Discussion

Figure 21 shows plots of the AUROC values on the test set as function of the number of training points for all methods used in our experiments. The error bars represent a 95% confidence interval. The top two plots are for Expert 1's supervision, and the bottom two are for Expert 2's. The left plots show results for methods that do not use confidence labels and the right plots are methods that do (with CAMEL in both for comparison between the plots). Overall, the results were similar for both clinicians. All methods achieved better results using Expert 1's labels when given few training points. Although, as more training points were added, all methods were able to improve more using Expert 2's supervision.

In terms of classification accuracy, CAMEL and CAMEL-CL performed as well or better than many of the competing methods, especially as more training instances were used. LMNN and MLKR learn metrics like CAMEL, but do not include regularization in their optimizations. Because of this, they are prone to over-fit to train sets, resulting in poor generalization, especially when there are few training instances. LASSO includes sparsity-inducing regularization, but learns a simple single-dimensional, linear model. Also, like MLKR, LASSO fits directly to the confidence labels. Because the exact values of the confidence labels contain a great deal of noise, LASSO and MLKR are unable to learn models that produce accurate confidence values on the test instances.

The most competitive models to CAMEL and CAMEL-CL, using the same supervision, were SVM and SVM-Combo, respectively. The SVM methods learn dense, single-dimensional models,

	Expert 1		Expert 2	
	Sparsity	Rank	Sparsity	Rank
CAMEL	0.794(0.241)	34.195(18.884)	0.754(0.237)	38.734(16.311)
CAMEL-CL	0.896(0.286)	20.080(18.921)	0.883(0.203)	21.755(18.862)

Table 3: CAMEL HIT data experiments: Mean (STD) sparsity statistics over all experiments

so they provide a meaningful basis of comparison to our methods. CAMEL performed at least as well and sometimes significantly better than SVM. The same is true for CAMEL-CL and SVM-Combo, though SVM-Combo outperformed CAMEL-CL for 10 and 20 training instances using Expert 2's supervision. With only this exception, CAMEL and CAMEL-CL were able to achieve AUROCs as high or higher than the SVM methods, given the same supervision. These results indicate that to our sparse, multidimensional models are able to more accurately predict whether a patient was at risk for HIT, than methods that learn dense and/or single-dimensional models.

This evaluation not only allowed us to compare our CAMEL methods to competing methods, but also enabled us to see the effect the confidence labels had on CAMEL. The inclusion of confidence labels only improved the AUROC of CAMEL for both experts, never hindered it. Using Expert 1's supervision, CAMEL-CL was able to achieve an AUROC with 10 training instances that could not be matched by CAMEL until it received 60. Furthermore, the AUROC of the CAMEL-CL model trained on 10 instances was statistically as high as any model trained on any number of instances with a 95% confidence. This indicates that to learn an accurate predictive model, CAMEL-CL requires substantially fewer labeled instances.

While prediction accuracy is important for a patient model, for it to be useful in a CDSS it must also be interpretable. In the remainder of this section, we assess the metrics produced by CAMEL and CAMEL-CL in terms of human-interpretability. In our experiments $\mathbf{L}$ is a 49 by 49 matrix that transforms the patient instances from their original space to a multidimensional metric space by linearly combining their features. Each row is a transformation to a different dimension in the metric space, and each column represents an individual feature's contribution to the model. In essence, each element of $\mathbf{L}$ corresponds to one of 49 weights on one of the features (e.g. the

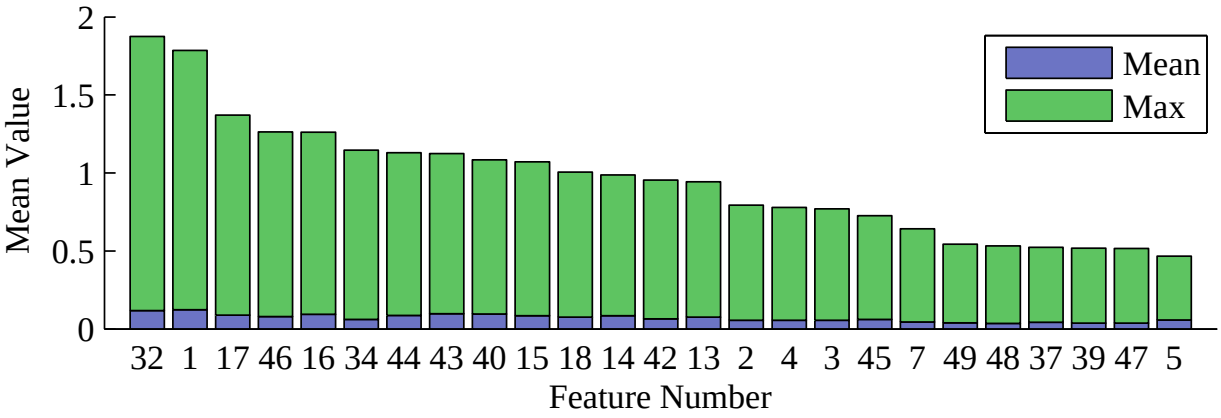


Figure 22: CAMEL HIT data experiments: feature weight statistics

(CAMEL-CL, Expert 2, ten trials)

49 values in column 1 weigh *Heparin on* feature). If not sufficiently sparse, $\mathbf{L}$ could be difficult to interpret, as the model would use many of the features, multiple times, in numerous different combinations. A sparse $\mathbf{L}$ that produces accurate inferences would use only a subset of the features in few, useful combinations, which could be easier to interpret than many, complex combinations.

Table 3 shows the mean "sparsity" of the metric parameter $\mathbf{L}$ produced by CAMEL and CAMEL-CL for all experiments. We define our sparsity statistic as the number of zero-valued elements of $\mathbf{L}$ divided by the total number of elements. A higher value means that the fewer features are being used fewer times in the model. We can see that models learned by CAMEL contain a very large number of zero-valued elements, but CAMEL-CL is able to be even more selective in choosing features by leveraging the confidence labels. Also in Tab. 3, we include the mean row rank of the $\mathbf{L}$ matrices. The row rank of a matrix is the number of linearly independent rows. In our models, a lower rank indicates there is a more simple, lower-dimensional space that describes how the CAMEL models are making inferences. The table shows that our methods are able to project the 49 dimensional patient instances into a lower-dimensional metric space in which accurate inferences can be made. The low-rank property of $\mathbf{L}$ can be attributed to the fact that strict ℓ-1 norm regularization often made many of the rows contain all zeros. For some trials, our methods produced an $\mathbf{L}$ with as many as 45 rows that contained solely zeros. While sparsity indicates the models are simple, it does not reveal how the features are being used. More specifically, for a model to be interpretable, a clinician

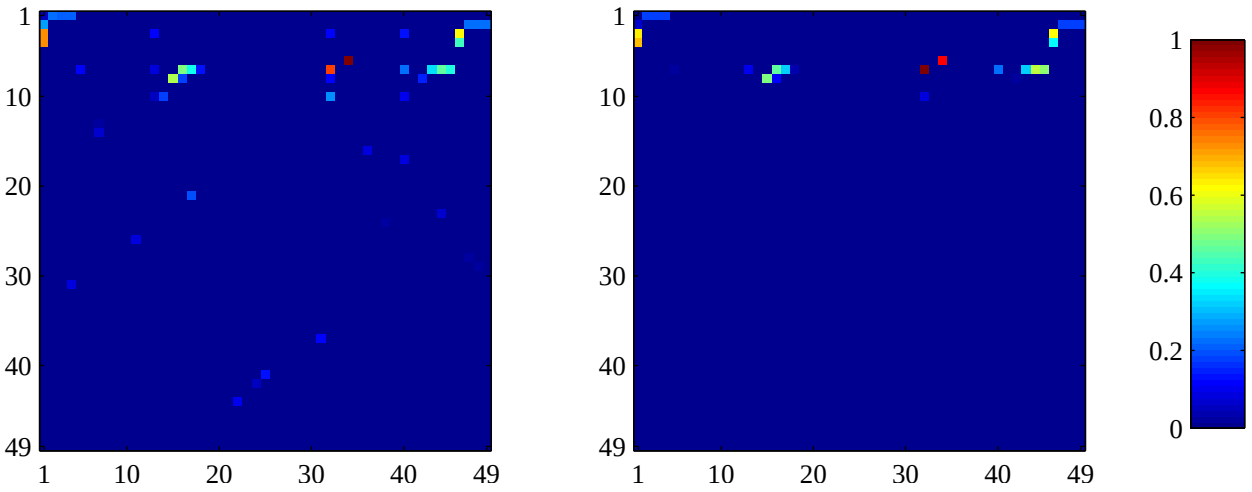


Figure 23: CAMEL HIT data experiments: normalized absolute values of $\mathbf{L}$ entries (left = CAMEL, right = CAMEL-CL)

should be able to tell which features are being used, how much, and whether they are important on their own, or in tandem with others. Figure 22 shows the mean and maximum absolute weight put on the top 25 features in the models produced by CAMEL-CL, averaged over all experiments done using Expert 2's supervision. In short, Fig. 22 displays the relative importance CAMEL-CL put on each feature. The clear top two features chosen by CAMEL-CL were "last platelet value taken" (32) and "Heparin on" (1). Clearly, whether a patient was given Heparin should influence whether they are at risk for HIT. Thrombocytopenia is indeed the deficiency of platelets in blood [**?**], thus the most recent value of platelet count intuitively should indicate risk of HIT. Other top features include "difference between the last and first hemoglobin level taken" (17), and "time since last major heart procedure" (46). A downward trend in hemoglobin level could indicate bleeding, leading to low platelet counts, making feature 17 a potential indicator of HIT. The time from last heart procedure could also be important as it indirectly measures how long the patient was on heparin. Note that the top four features all come from different attributes/lab values. This indicates that CAMEL-CL chooses which feature in a group is most informative and emphasizes it the most, as to not include redundant information. Also note that no feature measuring white blood cell count was featured prominently in the model. This model choice is supported by the convention that white blood cell

count is not commonly-used to indicate HIT.

Figure 23 displays two heat maps using the normalized absolute values of $\mathbf{L}$. Deep blue indicates a zero value, while deep red indicates the highest absolute value. The two maps depict $\mathbf{L}$ for CAMEL (left) and CAMEL-CL (right) in one trial using the same 100 training instances. Each row is a projection (weighing of the features) into a single dimension in the metric space. Thus, each row defines a different "factor" in which the metrics compare patients. The left heat map shows that the model produced by CAMEL is very sparse; it has mostly zero-valued elements (deep blue), a small number of small-valued elements (light blue), and an even smaller number of larger-valued elements (green, yellow, and red). In total, this matrix has 25 rows composed entirely of zeros (i.e. the induced metric space is $\mathbf{L}$ is 24 dimensional). However, many of the non-zero rows contribute very little to the overall model, as they contain only few, low-valued weights. Most likely, these rows simply add noise, and detract from the interpretability of the model by unnecessarily increasing the complexity. The heat map displaying the metric learned by CAMEL-CL, on the other hand, has many more zero-valued elements. Most of the small-valued elements in the CAMEL model were pushed to exactly zero in the CAMEL-CL model. In fact, every element of $\mathbf{L}$ learned by CAMEL-CL after the tenth row contains a zero, resulting a simpler, rank ten matrix. Because CAMEL-CL metric was given confidence labels in addition to the class labels, it was able to more accurately determine the few feature combinations that modeled patients well.

6.4 SUMMARY

In this chapter, we introduced a method called **C**onfidence-based **ME**tric **L**earning (CAMEL) that produces sparse, multidimensional, classification models that can perform inference on patient Electronic Health Records (EHRs) for use in Clinical Decision Support Systems (CDSSs). For an inference model to be effective in a CDSS, it must be both accurate and able to convey more information than simple predictions to clinicians. CAMEL was designed specifically with these qualities in mind. In order to combat the necessarily high cost of obtaining expert clinical supervision needed to learn an accurate model, we formulated a version CAMEL that can incorporate auxiliary confidence labels. In our experiments, we showed that CAMEL can produce models at least as

accurate as others we tested. CAMEL bolstered with confidence labels can produce models as accurate as any tested with using as few as 10% of the training instances as the other models, thus showing the human efficiency of our method. The qualitative analysis that followed highlighted the fact that CAMEL produces models that include few important "factors" composed of small subsets of the EHR features. Because CAMEL induces sparsity, it is able produce simple, concise patient models, potentially enabling clinicians to more clearly interpret how it makes decisions.

7.0 CONCLUSION AND FUTURE WORK

In this dissertation, we address the following thesis:

By defining and explicitly attending to the practical challenges in learning models of similarity from human feedback, more efficient learning methods can be developed to learn more effective models that can be used in machine learning and data mining methods.

To support this, in Chap. 1 we defined four challenge categories that similarity-learning methods face when learning from human feedback. In Chap. 2, we surveyed models of similarity that can be used as a basis for many machine learning methods. In addition, we reviewed common forms of human feedback, comparing them according to the previously defined challenge categories. In Chap. 3 we reviewed previous work in learning models of similarity. Among other topics, we focused on a problem called Relative Comparison Kernel Learning (RCKL). RCKL methods learn multidimensional models of similarity over a given set of objects from relative triplet feedback. These methods tend to be both model and human effective, but computationally and human inefficient. We also reviewed metric learning, highlighting two methods that learn metrics for classification and regression. The subsequent chapters describe the main contributions of this dissertation: addressing the challenges in RCKL, as well as those in a metric learning application.

7.1 CONTRIBUTIONS

In Chap. 4 we addressed the inherent human inefficiency of RCKL. Though relative triplet feedback is expressive enough to model multidimensional perceptual spaces, methods that learn

similarity from triplet feedback often require a large amount of feedback in order to produce models that capture a complete notion of how objects relate. Traditional RCKL methods assume no information is given about the objects for which similarity is to be learned, besides relative triplet feedback. In practice, objects often have uniquely identifying characteristics that can be captured by features drawn from data. If some or all of these features model trends in a given set of triplet responses, then they could also model key latent factors in how humans perceive object relationships. In turn, if features relevant to learning perceptual similarity are included into the RCKL learning process, the models they produce could capture object relationships not explicitly given, thus, modeling a more complete notion of similarity from fewer triplet responses.

To utilize auxiliary features describing objects, we developed an RCKL framework called RCKL with Auxiliary Kernels (RCKL-AK). Our framework extends traditional RCKL methods that learn non-parametric kernel matrices to include multiple *auxiliary kernels* as input. These kernel matrices can be built using standard kernel functions on the auxiliary features. RCKL-AK can incorporate auxiliary kernels into traditional RCKL methods with little additional computational overhead, and preserve the convex properties of the optimization problems some RCKL methods solve. We include regularization terms that allow RCKL-AK to select only the auxiliary kernels that are relevant to the similarity-learning task, and omit those that are not. Experimentally, we show that RCKL-AK can increase the human efficiency of RCKL methods by learning more complete models of similarity from fewer triplet responses. Also, we compare RCKL-AK to similar methods in metric learning and show that because RCKL-AK is not limited to a linear transformation of features as metric learning methods are, it can learn more general models.

To further increase the human efficiency of learning perceptual similarity from relative triplet feedback, we also introduced a novel method to actively select triplet queries in order to get the most informative responses. For this, we developed a method that learns an object embedding from triplet feedback that is based on Crowd Kernel Learning (CKL). Our embedding-learning method produces object embeddings that consists of two components: a parametric, weighted concatenation of auxiliary features, and a non-parametric set of features whose values are

learned directly. We call this method TACKL. TACKL is formulated probabilistically. As a result, we can define uncertainty of the positions of objects in the learned embedding using information-theoretic concepts, such as Shannon entropy. With TACKL as a basis, we develop an adaptive query selection scheme that selects triplet queries whose responses in expectation will reduce the uncertainty in the object embeddings the most. Our scheme benefits both the non-parametric component, quickly finding the appropriate feature weighing, as well as the non-parametric component which models object relationships not covered in the parametric component. We evaluate TACKL with our adaptive triplet selection scheme (A-TACKL) experimentally on both synthetic and real-world triplet feedback. We show that A-TACKL requires less feedback to learn more general models than methods that do not use auxiliary information and/or select queries at random.

In Chap. 5 we addressed the computational inefficiency of RCKL methods in the online setting. In practice, relative triplet feedback is often passively obtained in an online matter. If traditional batch RCKL methods are used to update similarity models as feedback is obtained, they face serious issues in terms of computational efficiency. To ensure the models they learn are valid kernel matrices, batch RCKL methods must maintain that their solutions are positive semidefinite (PSD). For this, RCKL methods typically employ projected gradient descent procedures, which project intermediate solutions onto the cone of PSD matrices after each gradient step. With no knowledge of the structure of the gradient, this is an $O\left(n^3\right)$ procedure that must be performed multiple times, every time feedback is obtained. This necessitates the development of efficient, online RCKL methods.

To this end, we develop a framework to learn a non-parametric kernel matrix from triplet feedback that is given in an online manner. We call our framework **E**fficient online **R**elative comparison **K**ernel **LE**arning (ERKLE). ERKLE updates a kernel using a stochastic gradient step with respect to a single triplet response. We show that due to the structure of this gradient, we can project onto the PSD cone in $O\left(n^2\right)$ time. Furthermore, if we maintain an estimate of the smallest eigenvalue of the learned kernel, then we can often skip the projection step entirely. To avoid as many projections as possible and to eliminate the need to manually set a step size, we formulate a passive-aggressive version of ERKLE that steps the minimal amount needed to

satisfy a given response, but does not step at all if the response is already satisfied by the current solution. In our experimental evaluation, we show that ERKLE is able to produce kernels that are as accurate as those learned by batch solutions while taking only a fraction of the time to learn.

In Chap. 6 we addressed the human efficiency and model effectiveness of learning a metric for clinical decision support. Here, we focused on a specific application of learning a model of similarity such that inference can be performed as part of a Clinical Decision Support System (CDSS). More specifically, we consider the application of learning an inference model to alert a clinician when a patient is at risk for a disease from patient Electronic Health Record (EHR) data and expert provided class labels. In order for a clinician to trust its inferences, a CDSS must be able to provide sufficient information to allow the clinician to understand the model. As a result, in this problem setting, model effectiveness not only entails accuracy of inferences, but also interpretability in the learned model. Most common supervised learning techniques that can be applied to this setting, such as linear classifiers, are not sufficiently interpretable for this case. In addition, learning an inference model from clinical feedback introduces an additional challenge not common to other classification tasks. Because expert supervision is especially costly, methods applied to this task must focus on being human efficient in order to reduce the cost of obtaining feedback.

With these challenges in mind, we developed a method called **C**onfidence-b**A**sed **ME**tric **L**earning (CAMEL), which learns a Mahalanobis distance metric from which class predictions can be made. CAMEL was developed with a specific focus on interpretability. CAMEL learns a metric using well-defined confidence scores, which can be shown to clinicians to indicate how sure CAMEL is in its predictions. This additional information can be used by clinicians to motivate treatment decisions. Also, CAMEL is formulated to induce sparsity on the learned metric. As a result, only a few small combinations of features are used in the model. Each combination represents a different "factor" in how the metric models patient relationships with respect to disease. These factors can be shown to clinicians to provide insight into the model. To increase human efficiency, we prompt clinicians for auxiliary confidence labels that indicate how sure they are in the class labels they provide. Because most of the time spent

labeling patients is in considering the EHRs themselves, giving confidence information requires only a small amount of effort in addition to providing a class label. CAMEL is able to use this inexpensive, extra information to learn accurate models from fewer labeled patient instances.

We performed an experimental evaluation of CAMEL on a real-world clinical data set where experts indicated whether patient instances, represented by features drawn from EHRs, were at risk for Heparin Induced Thrombocytopenia (HIT). Using this data, we show that CAMEL is able to learn models as accurate as competing methods that use just the class labels, just the confidence labels, or both, while using much fewer labeled instances. We also provided a qualitative assessment of the interpretability of the models CAMEL produces. We show that the features used most often in CAMEL models are ones commonly used to diagnosis HIT. Furthermore, we showed the "factors" learned by CAMEL. CAMEL learned only a few key combinations of small sets of features whose interaction may be interpreted by a clinician.

Through these contributions, we are able to support the thesis of this dissertation. Chapters 4, 5, and 6 introduce novel similarity-learning methods that improve on the state-of-the-art in one or more challenge categories. These methods learn models of similarity commonly utilized by modern machine learning methods. For these reasons, the methods therein enable similarity-methods to be used in practical learning settings where human feedback is obtained.

7.2 FUTURE WORK

To conclude this dissertation, we discuss directions of future work, both in regards to the specific methods introduced here, but also to the general problem of learning similarity from human feedback.

7.2.1 Directions of Future Work Specific to Methods Introduced in this Dissertation

1. One shortcoming of RCKL methods is that they do not learn models that can generalize to out-of-sample objects. While there are many practical applications that require a similarity model

defined only over a fixed set of object, the kernels learned by RCKL methods are not applicable to inductive learning settings. RCKL is not unique in this regard. Other similarity-learning methods, such as IsoMap [133] and Local Linear Embedding [111] learn models that cannot be applied to out-of-sample objects. To correct this deficiency, out-of-sample extensions have been developed for these methods [11]. Similar extensions may be possible for RCKL methods. Another potential direction is mapping the object relationships learned by RCKL methods into relative attribute spaces, which can be applied to out-of-sample objects. Doing so would benefit both models of similarity: RCKL models would have a way to include include out-of-sample objects and have nameable attributes associated with how humans perceive objects, while relative attribute spaces can be refined to more specifically model the perceptual similarity that certain tasks require.

2. With exception of Chap. 6, the work presented in this dissertation focused on the process of learning models of similarity, and not on how they are used *after* they are learned. Recently, there has been works in utilizing models that RCKL methods produce for fine-grained image classification [148, 147]. However, many other transductive learning tasks could benefit from a perceptual model of similarity such as those learned by RCKL methods. Furthermore, if out-of-sample extensions for RCKL methods are developed, then the resulting models can be used for a larger number of applications.

3. While adaptively selecting mini-batches of triplet queries is of practical use in crowdsourcing for human perception, other applications may benefit from selecting single queries so that a model may be updated in a true, online fashion. The adaptive selection schemes of A-CKL or A-TACKL can be directly applied to online selection by iteratively updating a model as each query in a round is answered. Intuitively, however, gains in human efficiency may be obtained by adaptively choosing the best triplet query regardless of head object, instead of having each object appear as a head once per round. We performed preliminary experiments where the head of a triplet query could be any object when A-CKL chose a query in an online fashion. Unfortunately, this lead to only a few objects being chosen as the head objects and the rest being starved by the selection algorithm. As a result, the certainty in the positions of the starved objects was never improved. More sophisticated adaptive selection schemes must be used if truly online adaptive triplet query selections methods are to be developed.

4. A practical issue with both A-CKL and A-TACKL is the high run time of their adaptive selection schemes. Both can be used in practice by sub-sampling at various steps in their respective algorithms. Nevertheless, the inefficiency in these, and other potential query selection schemes is largely due to the sizable pool of candidate queries from which they select. If this pool can be reduced, then both adaptive selection methods can improve greatly in terms of computational efficiency. We have performed preliminary experiments in reducing the pool of candidate queries by assuming structure on the embedding of objects, but further experiments are necessary.

5. As mentioned in Chap. 5, ERKLE does not perform any form of regularization. Because of this, users of PA-ERKLE need not manually set hyperparameters. The drawback is that the kernels produced by ERKLE tend to be high rank, and could potentially benefit from some form of rank-reducing regularization when the true perceptual similarity space being learned is low-dimensional. We have experimented with incorporating a form of incremental singular value decomposition [17] into ERKLE to add regularization. Unfortunately, doing so requires finding at least one eigenvalue of the learned learned at each update, eliminating the possibility to perform constant-time updates. In addition, we found this method somewhat unstable, occasionally leading to inaccurate models. Further investigation is required to efficiently and effectively include regularization into the ERKLE framework.

6. The metric learned by CAMEL is defined over the entire domain of patient instances. Often, a single metric cannot accurately model the relationships over an entire domain. Indeed, this seemed to be the case in the HIT data for which we evaluated CAMEL. Certain EHR features were binary (e.g. "Patient was given Heparin"), which indicates clear sub-populations within the data (e.g. patients who were given Heparin, and patients who were not). It is possible that patients within the same sub-population cannot be modeled the same way as patients in different sub-populations. This is the motivation behind local metric learning, where different metrics are defined over separate subspaces in the domain of objects. Application of local metrics to patient sub-populations could further improve the accuracy of CAMEL.

7.2.2 Directions of Future Work in Similarity Learning from Human Feedback

1. We discussed in Sec. 2.3 that different forms of feedback have different strengths, weaknesses, roles, and applications. Relative triplet feedback is beneficial when trying to elicit multidimensional relationships from humans, but is human inefficient. Pair-wise similarity/dissimilarity assessment are more human efficient, but convey more simple, yet more strict object relationships. Consider, again, the task of learning multi-dimensional perceptual similarity. Prompting humans for a combination of the two forms could mitigate the weaknesses of both forms for this task. For instance, one could pose pair-wise queries to humans to get gain cluster structure in the set of object if it exists. Then, a few, carefully chosen triplet queries can be asked to refine that structure. Other forms of feedback could even be used, if appropriate.

   To learn perceptual similarity from multiple different forms of feedback, not only would novel similarity-learning methods need to be developed, but certain new questions would need to be answered. Active RCKL methods attempt to choose the best triplet query to learn accurate similarity models with less feedback. If queries of different forms could be asked, what form the query should take would also need to be considered. What defines a good query in this context has yet to be studied. Intuitively, however, a good query should be easy for humans to provide, but also result in responses that are accurate and informative. Furthermore, it remains to be studied how different forms should interact if they contain conflicting information. Noise within single forms of feedback has been previously considered. For example, in [91] the authors develop a method to handle conflicting triplet responses. However, if a pair-wise assessment conflicts with a triplet response, it is unclear how this, or other inter-form conflicts should be resolved.

2. In Chap 4 we show that learned Mahalanobis distance metrics are limited in their ability to express perceptual similarity spaces due to restricting solutions to a class of linear transformations. Recently, *deep learning* methods have gained immense popularity for their success in numerous applications. While, the term "deep learning" is used to describe a wide variety of techniques, what they have in common is that they learn complex functions to model high-level abstractions of data. As such, they are able to express object relationships that a Mahalanobis distance metric cannot. In fact, there has been recent work in developing deep metrics [64]

using convolutional neural networks for the application of face recognition. If these concepts can be used in learning perceptual similarity, then they could produce more expressive metrics that can be applied to inductive learning tasks.

3. Finally, throughout this work we have considered models of similarity that have the necessary properties to define them as metrics. Recent work [114] has suggested that visual recognition, a task for which perceptual similarity models are often applied, is non-metric. The author remarks that work in cognitive psychology [138, 139] suggests that human's perception of objects is not symmetric and does not satisfy the triangle inequality. In essence, the context in which objects are given to a human determines how she perceives them. In terms of triplet feedback, this could mean what we consider a "conflict" between triplet responses is actually not noise, but a result of perception being non-metric. From a modeling standpoint, this is inconvenient. Many machine learning methods, including most similarity-learning methods, assume object relationships are metric. If perception is to be modeled as cognitive psychology understands it, then similarity-learning methods must be developed to model non-metric object relationships, and machine learning methods must be adapted to use this kind of similarity model.

APPENDIX A

PROOFS OF CONVEXITY FOR RCKL-AK FORMULATIONS

The strategy employed throughout this section to prove the stated functions are convex is to build each using convex combinations of convex functions. In order to use this strategy, we need to establish the following Lems. (Above each Lemma is a reference to a source for each lemma, respectively).

Section 3.2.1 of [16]:

Lemma 1 *If f and g are both convex functions, then so is their sum f + g.*

Section 2.3.2 of [16]:

Lemma 2 *Affine functions of the form $f(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$, where $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{x} \in \mathbb{R}^n$, and $\mathbf{b} \in \mathbb{R}^m$ are convex in $\mathbf{x}$*

Section 3.2.3 of [16]:

Lemma 3 *If f and g are convex functions, then their point-wise maximum, $\max(f(x), g(x))$, is also convex.*

In addition, we will use the concept of *logarithmic convexity*:

Definition 1 A function f is *logarithmically convex (log-convex)* if $f(x) > 0$ for all $x \in \mathbf{dom} f$ and $\log f$ is convex.

Which we then use to state the following Lemma.

Section 3.5.2 of [16]:

Lemma 4 *If f and g are both log-convex functions, then so is their sum $f + g$.*

Finally, for the sake of notational brevity, let us define the following:

$$d_{\mathbf{K}}^{ab} = d_{\mathbf{K}}(x_a, x_b)$$
$$D_{\mathbf{K}}^{abc} = d_{\mathbf{K}}(x_a, x_b) - d_{\mathbf{K}}(x_a, x_c)$$

These short-hand versions of our established notation will be used throughout this section.

A.1 PROOF OF PROPOSITION 1

PROOF

1. In order for (4.6) to be a convex optimization problem, it's objective and constraints must be convex in the optimization variables.
2. By Lem. 1, if $E(\mathbf{K}'', \mathcal{T})$, $\lambda_1 \text{trace}(\mathbf{K}_0)$, and $\lambda_2 \|\boldsymbol{\mu}\|_1$ are convex, then the objective in (4.6) is convex.
3. It is an assumption of the proposition that $E(\mathbf{K}'', \mathcal{T})$ is convex.
4. $\lambda_1 \text{trace}(\mathbf{K}_0)$ is defined as a constant times the sum of the diagonal elements of the matrix $\mathbf{K}_0$, which is a sum of convex functions (Lem. 1).
5. $\lambda_2 \|\boldsymbol{\mu}\|_1$ is defined as a constant times the the sum of the absolute values of the elements of $\boldsymbol{\mu}$, which is a sum of convex functions (Lem. 1)
6. By lines 2, 3, 4, and 5, the objective in (4.6) is convex.
7. The positivity constraint on $\boldsymbol{\mu}$ is trivially convex.
8. The positive semidefinite constraint is known to be convex [**?**].
9. By lines 7 and 8, both constraints of (4.6) are convex.
10. By lines 1, 6, and 9, (4.6) is a convex optimization problem.

A.2 PROOF OF PROPOSITION 2

PROOF

1. Moving the negation into the sum, (4.7) becomes the sum of terms of the following form:

$$
\begin{aligned}
& -\log\left(p_{abc}^{\mathbf{K}''}\right) && \text{(A.1)}\\
= \; & -\log\left(\frac{\exp\left(-d_{\mathbf{K}''}^{ab}\right)}{\exp\left(-d_{\mathbf{K}''}^{ac}\right)+\exp\left(-d_{\mathbf{K}''}^{ab}\right)}\right)\\
= \; & -\log\left(\frac{1}{1+\exp\left(D_{\mathbf{K}''}^{abc}\right)}\right)\\
= \; & -\log(1) && \text{(A.2)}\\
+ \; & \log\left(\exp\left(D_{\mathbf{K}''}^{abc}\right)+1\right) && \text{(A.3)}
\end{aligned}
$$

2. By Lem. 1, if (A.1) is convex for all triplets (a, b, c), then (4.7) is convex.
3. By Lem. 1, (A.1) is convex if both (A.2) and (A.3) are convex.
4. (A.2) is a constant and trivially convex.
5. By Definition 1, if $\exp\left(D_{\mathbf{K}''}^{abc}\right)+1$ is log-convex, then (A.3) is convex.
6. By Lem. 4 if $\exp\left(D_{\mathbf{K}''}^{abc}\right)$ and 1 are both log-convex, then $\exp\left(D_{\mathbf{K}''}^{abc}\right)+1$ is log-convex.
7. 1 is a constant and trivially log-convex.
8. The codomain of the exponential function is $\mathbb{R}^+$, so $\exp\left(D_{\mathbf{K}''}^{abc}\right) > 0$ for all $\mathbf{K}''$, which satisfies the first condition for log-convexity.
9. To show $\log\left(\exp\left(D_{\mathbf{K}''}^{abc}\right)\right)$ is convex, thus satisfying the second condition of log-convexity, we start by stating the following equivalence by using the definition of $\mathbf{K}''$ (4.4):

$$
\begin{aligned}
\log\left(\exp\left(D_{\mathbf{K}''}^{abc}\right)\right) &= D_{\mathbf{K}''}^{abc}\\
&= D_{\mathbf{K}_0}^{abc}+\sum_{i=1}^{A}\mu_i D_{\mathbf{K}_i}^{abc} && \text{(A.4)}
\end{aligned}
$$

10. By Lem. 1 if $D_{\mathbf{K}_0}^{abc}$ and $\sum_{i=1}^{A}\mu_i D_{\mathbf{K}_i}^{abc}$ are convex, then (A.4) is convex.
11. Let $\mathbf{k}_{abc} = \left(D_{\mathbf{K}_1}^{abc}, ..., D_{\mathbf{K}_A}^{abc}\right)$. Then, $\sum_{i=1}^{A}\mu_i D_{\mathbf{K}_i}^{abc} = \mathbf{k}_{abc}^T\boldsymbol{\mu}$.
12. $\mathbf{k}_{abc}^T\boldsymbol{\mu}$ is an affine function of $\boldsymbol{\mu}$ that has the form $f(\boldsymbol{\mu}) = \mathbf{A}\boldsymbol{\mu}+\mathbf{b}$ where $\mathbf{A} = \mathbf{k}_{abc}^T$ and $\mathbf{b}$ is 0.

13. By Lem. 2 and the previous step, $\mathbf{k}_{abc}^T \boldsymbol{\mu}$ is convex in $\boldsymbol{\mu}$.
14. Using the definition of kernel distance from (3.8) (Note the slight change in notation: $\mathbf{K}_0^{ab}$ refers to the ath column and bth row of $\mathbf{K}_0$):

$$D_{\mathbf{K}_0}^{abc} = \mathbf{K}_0^{bb} + 2\mathbf{K}_0^{ac} - \mathbf{K}_0^{cc} - 2\mathbf{K}_0^{ab} \tag{A.5}$$

15. By Lem. 1, if the individual terms of (A.5) are convex then (A.5) is convex.
16. The individual terms of (A.5) are simply elements of $\mathbf{K}_0$ multiplied by scalers, which are convex in $\mathbf{K}_0$.
17. By lines 3-16, (A.1) is convex.
18. By lines 1, 2, and 17, (4.7) is convex.

A.3 PROOF OF PROPOSITION 3

PROOF

1. By Lem. 1 if $\max(0, D_{\mathbf{K}''}^{abc} + 1)$ is convex for any triplet (a, b, c), then (4.8) is convex.
2. By Lem. 3, if 0 and $D_{\mathbf{K}''}^{abc} + 1$ are convex, then $\max(0, D_{\mathbf{K}''}^{abc} + 1)$ is convex.
3. 0 is trivially convex.
4. By Lem. 1, if 1 and $D_{\mathbf{K}''}^{abc}$ are convex, then $D_{\mathbf{K}''}^{abc} + 1$ is convex.
5. 1 is trivially convex.
6. Steps 9-16 of Section A.2 showed $D_{\mathbf{K}''}^{abc}$ is convex in the optimization variables.
7. By lines 2-6, $\max(0, D_{\mathbf{K}''}^{abc} + 1)$ is convex for any triplet (a, b, c).
8. By line 1 and 6, (4.8) is convex

APPENDIX B

DERIVATION OF STE PASSIVE-AGGRESSIVE STEP SIZE

To derive the STE version of the passive-aggressive step size we wish to solve the following optimization (5.11):

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

As with the GNMDS derivation with the assumption that the triplet is not satisfied by a probability greater than or equal to P, only a positive value of δ_j can satisfy the first constraint, making the positive constraint on δ_j redundant. In addition, the smallest δ_j that satisfies the remaining constraint is the one that makes the left hand side exactly zero. As a result, the inequality constraint can be handled as equality. Next, we take the Lagrangian:

$$\begin{aligned} \delta_j^2 \quad & +\alpha \left(\log \left(P \right) - \log \left(1 - P \right) \right) \\ & +\alpha \left(d_{\mathbf{K}_{j-1}}(a, b) - d_{\mathbf{K}_{j-1}}(a, c) \right) - 10\delta_j \alpha \end{aligned}$$

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

$$\begin{aligned} -25\alpha^2 \quad & +\alpha \left(\log \left(P \right) - \log \left(1 - P \right) \right) \\ & +\alpha \left(d_{\mathbf{K}_{j-1}}(a, b) - d_{\mathbf{K}_{j-1}}(a, c) \right) \end{aligned}$$

Taking the partial derivative of the Lagrangian with respect to α, setting it to 0, and solving for α results in:

$$\alpha = \frac{\log \left(P \right) - \log \left(1 - P \right) + d_{\mathbf{K}_{j-1}}(a, b) - d_{\mathbf{K}_{j-1}}(a, c)}{50}$$

Substituting this into the solution for δ_j gives us:

$$\delta_j = \frac{\log(P) - \log(1-P) + d_{\mathbf{K}_{j-1}}(a,b) - d_{\mathbf{K}_{j-1}}(a,c)}{10}$$

This is what is given in (5.12).

BIBLIOGRAPHY

[1] M. E. Abbasnejad, D. Ramachandram, and R. Mandava. A survey of the state of the art in learning the kernels. *Knowledge and information systems*, 31(2):193–221, 2012.

[2] M. A. Acevedo, C. J. Corrada-Bravo, H. Corrada-Bravo, L. J. Villanueva-Rivera, and T. M. Aide. Automated classification of bird and amphibian calls using machine learning: A comparison of methods. *Ecological Informatics*, 4(4):206–214, 2009.

[3] A. Agarwal, M. J. Wainwright, P. L. Bartlett, and P. K. Ravikumar. Information-theoretic lower bounds on the oracle complexity of convex optimization. In *Advances in Neural Information Processing Systems*, pages 1–9, 2009.

[4] S. Agarwal, J. Wills, L. Cayton, G. Lanckriet, D. J. Kriegman, and S. Belongie. Generalized non-metric multidimensional scaling. In *International Conference on Artificial Intelligence and Statistics*, pages 11–18, 2007.

[5] E. Amid, A. Gionis, and A. Ukkonen. A kernel-learning approach to semi-supervised clustering with relative distance comparisons. In *Machine Learning and Knowledge Discovery in Databases*, pages 219–234. Springer, 2015.

[6] E. Amid and A. Ukkonen. Multiview triplet embedding: Learning attributes in multiple maps. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1472–1480, 2015.

[7] F. Bach. Adaptivity of averaged stochastic gradient descent to local strong convexity for logistic regression. *The Journal of Machine Learning Research*, 15(1):595–627, 2014.

[8] F. R. Bach, G. R. Lanckriet, and M. I. Jordan. Multiple kernel learning, conic duality, and the smo algorithm. In *Proceedings of the twenty-first international conference on Machine learning*, page 6. ACM, 2004.

[9] S. Basu, A. Banjeree, E. Mooney, A. Banerjee, and R. J. Mooney. Active semi-supervision for pairwise constrained clustering. In *In Proceedings of the 2004 SIAM International Conference on Data Mining (SDM-04*, 2004.

[10] A. Bellet, A. Habrard, and M. Sebban. A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv:1306.6709*, 2013.

[11] Y. Bengio, J.-f. Paiement, P. Vincent, O. Delalleau, N. L. Roux, and M. Ouimet. Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering. In *Advances in Neural Information Processing Systems*, pages 177–184, 2004.

[12] E. S. Berner. *Clinical decision support systems: theory and practice*. Springer Science & Business Media, 2007.

[13] A. Biswas and D. Parikh. Simultaneous active learning of classifiers & attributes via relative feedback. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 644–651. IEEE, 2013.

[14] 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.

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

[16] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[17] M. Brand. Incremental singular value decomposition of uncertain data with missing values. In *Computer VisionECCV 2002*, pages 707–720. Springer, 2002.

[18] S. Branson, C. Wah, F. Schroff, B. Babenko, P. Welinder, P. Perona, and S. Belongie. Visual recognition with humans in the loop. In *Computer Vision–ECCV 2010*, pages 438–451. Springer, 2010.

[19] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96. ACM, 2005.

[20] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon. Adapting ranking svm to document retrieval. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 186–193. ACM, 2006.

[21] N. Cesa-Bianchi and G. Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006.

[22] O. Chapelle and V. Vapnik. Model selection for support vector machines. In *Advances in Neural Information Processing Systems*, 2000.

[23] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine learning*, 46(1-3):131–159, 2002.

[24] D. Charles. *Adoption of electronic health record systems among US non-federal acute care hospitals: 2008-2012*. Office of the National Coordinator for Health Information Technology, 2013.

[25] J. Chen, T. Yang, and S. Zhu. Efficient low-rank stochastic gradient descent methods for solving semidefinite programs. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, pages 122–130, 2014.

[26] V. Cherkassky and Y. Ma. Practical selection of svm parameters and noise estimation for svm regression. *Neural networks*, 17(1):113–126, 2004.

[27] W. Chu and Z. Ghahramani. Preference learning with gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 137–144. ACM, 2005.

[28] H. Cohen and C. Lefebvre. *Handbook of categorization in cognitive science*. Elsevier, 2005.

[29] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[30] A. Cotter, O. Shamir, N. Srebro, and K. Sridharan. Better mini-batch algorithms via accelerated gradient methods. In *Advances in neural information processing systems*, pages 1647–1655, 2011.

[31] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 7:551–585, 2006.

[32] K. Crammer and Y. Singer. Pranking with ranking. In *Advances in Neural Information Processing Systems 14*. Citeseer, 2001.

[33] I. Dagan and S. P. Engelson. Committee-based sampling for training probabilistic classifiers. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 150–157, 1995.

[34] J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon. Information-theoretic metric learning. In *Proceedings of the 24th international conference on Machine learning*, pages 209–216. ACM, 2007.

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

[36] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[37] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International Conference on Machine Learning*, pages 647–655, 2014.

[38] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification*. John Wiley & Sons, 2012.

[39] I. El-Naqa, Y. Yang, M. N. Wernick, N. P. Galatsanos, and R. M. Nishikawa. A support vector machine approach for detection of microcalcifications. *Medical Imaging, IEEE Transactions on*, 21(12):1552–1563, 2002.

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

[41] J.-H. Eom, S.-C. Kim, and B.-T. Zhang. Aptacdss-e: A classifier ensemble-based clinical decision support system for cardiovascular disease level prediction. *Expert Systems with Applications*, 34(4):2465–2479, 2008.

[42] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[43] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

[44] X. Fan, Y. Liu, N. Cao, J. Hong, and J. Wang. Mindminer: A mixed-initiative interface for interactive distance metric learning. In *Human-Computer Interaction–INTERACT 2015*, pages 611–628. Springer, 2015.

[45] A. Farhadi, I. Endres, D. Hoiem, and D. Forsyth. Describing objects by their attributes. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1778–1785. IEEE, 2009.

[46] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1627–1645, 2010.

[47] V. Ferrari and A. Zisserman. Learning visual attributes. In *Advances in Neural Information Processing Systems*, pages 433–440, 2007.

[48] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[49] E. Frank and M. Hall. A simple approach to ordinal classification. In *Proceedings of the 12th European Conference on Machine Learning*, pages 145–156. Springer-Verlag, 2001.

[50] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *The Journal of machine learning research*, 4:933–969, 2003.

[51] C. F. Gauss. *Theoria motus corporum coelestium in sectionibus conicis solem ambientium auctore Carolo Friderico Gauss*. sumtibus Frid. Perthes et IH Besser, 1809.

[52] A. Globerson and S. T. Roweis. Metric learning by collapsing classes. In *Advances in neural information processing systems*, pages 451–458, 2005.

[53] M. Gönen and E. Alpaydın. Multiple kernel learning algorithms. *The Journal of Machine Learning Research*, 12:2211–2268, 2011.

[54] Y. Guo and R. Greiner. Optimistic active-learning using mutual information. In *IJCAI*, volume 7, pages 823–829, 2007.

[55] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422, 2002.

[56] M. Hauskrecht, I. Batal, M. Valko, S. Visweswaran, G. F. Cooper, and G. Clermont. Outlier detection for patient monitoring and alerting. *Journal of biomedical informatics*, 46(1):47–55, 2013.

[57] M. Hauskrecht, M. Valko, I. Batal, G. Clermont, S. Visweswaran, and G. F. Cooper. Conditional outlier detection for clinical alerting. In *AMIA annual symposium proceedings*, volume 2010, page 286. American Medical Informatics Association, 2010.

[58] E. Hazan and S. Kale. Projection-free online learning. In *Proceedings of the 30th International Conference on Machine Learning*, 2012.

[59] H. Heikinheimo and A. Ukkonen. The crowd-median algorithm. In *First AAAI Conference on Human Computation and Crowdsourcing*, 2013.

[60] E. Heim, M. Berger, L. Seversky, and M. Hauskrecht. Efficient online relative comparison kernel learning. *Proceedings of the 2015 SIAM International Conference on Data Mining (SDM15)*, 2015.

[61] E. Heim and M. Hauskrecht. Sparse multidimensional patient modeling using auxiliary confidence labels. *Proceedings of the 2015 IEEE International Conference on Bioinformatics and Biomedicine*, 2015.

[62] E. Heim, H. Valizadegan, and M. Hauskrecht. Relative comparison kernel learning with auxiliary kernels. In T. Calders, F. Esposito, E. Hllermeier, and R. Meo, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 8724 of *Lecture Notes in Artificial Intelligence*, pages 563–578. Springer Berlin Heidelberg, 2014.

[63] S. C. Hoi, R. Jin, and M. R. Lyu. Learning nonparametric kernel matrices from pairwise constraints. In *Proceedings of the 24th international conference on Machine learning*, pages 361–368. ACM, 2007.

[64] J. Hu, J. Lu, and Y.-P. Tan. Discriminative deep metric learning for face verification in the wild. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1875–1882. IEEE, 2014.

[65] K. Huang, Y. Ying, and C. Campbell. Generalized sparse metric learning with relative comparisons. *KAIS*, 28(1):25–45, 2011.

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

[67] A. K. Jha, M. F. Burke, C. DesRoches, M. S. Joshi, P. D. Kralovec, E. G. Campbell, and M. B. Buntin. Progress toward meaningful use: hospitals' adoption of electronic health records. *The American journal of managed care*, 17, 2011.

[68] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2002.

[69] D. Kedem, S. Tyree, F. Sha, G. R. Lanckriet, and K. Q. Weinberger. Non-linear metric learning. In *Advances in Neural Information Processing Systems*, pages 2573–2581, 2012.

[70] D. E. Knuth. *The art of computer programming: sorting and searching*, volume 3. Pearson Education, 1998.

[71] I. Kononenko. Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in medicine*, 23(1):89–109, 2001.

[72] A. Kovashka, D. Parikh, and K. Grauman. Whittlesearch: Image search with relative attribute feedback. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2973–2980. IEEE, 2012.

[73] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

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

[75] B. Kulis. Metric learning: A survey. *Foundations & Trends in Machine Learning*, 5.4:287–364, 2012.

[76] B. Kulis, M. Sustik, and I. Dhillon. Learning low-rank kernel matrices. In *Proceedings of the 23rd international conference on Machine learning*, pages 505–512. ACM, 2006.

[77] N. Kumar, P. N. Belhumeur, A. Biswas, D. W. Jacobs, W. J. Kress, I. C. Lopez, and J. V. Soares. Leafsnap: A computer vision system for automatic plant species identification. In *Computer Vision–ECCV 2012*, pages 502–516. Springer, 2012.

[78] J. T. Kwok and I. W. Tsang. Learning with idealized kernels. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 400–407, 2003.

[79] C. H. Lampert, H. Nickisch, and S. Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 951–958. IEEE, 2009.

[80] G. R. Lanckriet, N. Cristianini, P. Bartlett, L. E. Ghaoui, and M. I. Jordan. Learning the kernel matrix with semidefinite programming. *The Journal of Machine Learning Research*, 5:27–72, 2004.

[81] J. D. Lee, B. Recht, N. Srebro, J. Tropp, and R. R. Salakhutdinov. Practical large-scale optimization for max-norm regularization. In *Advances in Neural Information Processing Systems*, pages 1297–1305, 2010.

[82] A. M. Legendre. *Nouvelles méthodes pour la détermination des orbites des comètes*. Number 1. F. Didot, 1805.

[83] D. D. Lewis and J. Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the eleventh international conference on machine learning*, pages 148–156, 1994.

[84] D. D. Lewis and W. A. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–12. Springer-Verlag New York, Inc., 1994.

[85] M. Li, T. Zhang, Y. Chen, and A. J. Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 661–670. ACM, 2014.

[86] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *The Journal of Machine Learning Research*, 2:419–444, 2002.

[87] M. Mahdavi, T. Yang, R. Jin, S. Zhu, and J. Yi. Stochastic gradient descent with only one projection. In *Advances in Neural Information Processing Systems*, pages 494–502, 2012.

[88] A. K. McCallum and K. Nigamy. Employing em and pool-based active learning for text classification. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML)*. Citeseer, 1998.

[89] P. McCullagh. Regression models for ordinal data. *Journal of the royal statistical society. Series B (Methodological)*, pages 109–142, 1980.

[90] B. McFee and G. Lanckriet. Heterogeneous embedding for subjective artist similarity. In *ISMIR*, 2009.

[91] B. McFee and G. Lanckriet. Learning multi-modal similarity. *The Journal of Machine Learning Research*, 12:491–523, 2011.

[92] C. A. Micchelli, Y. Xu, and H. Zhang. Universal kernels. *The Journal of Machine Learning Research*, 7:2651–2667, 2006.

[93] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. Machine learning, neural and statistical classification. 1994.

[94] R. Mises and H. Pollaczek-Geiringer. Praktische verfahren der gleichungsauflösung. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, 9(1):58–77, 1929.

[95] E. Moulines and F. R. Bach. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *Advances in Neural Information Processing Systems*, pages 451–459, 2011.

[96] Y. Nesterov and J.-P. Vial. Confidence level solutions for stochastic programming. *Automatica*, 44(6):1559–1568, 2008.

[97] Q. Nguyen, H. Valizadegan, and M. Hauskrecht. Learning classification with auxiliary probabilistic information. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 477–486. IEEE, 2011.

[98] Q. Nguyen, H. Valizadegan, and M. Hauskrecht. Learning classification models with soft-label information. *Journal of the American Medical Informatics Association*, 21(3):501–508, 2014.

[99] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International journal of computer vision*, 42(3):145–175, 2001.

[100] V. Y. Pan and Z. Q. Chen. The complexity of the matrix eigenproblem. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 507–516. ACM, 1999.

[101] S. Parameswaran and K. Q. Weinberger. Large margin multi-task metric learning. In *Advances in neural information processing systems*, pages 1867–1875, 2010.

[102] D. Parikh and K. Grauman. Relative attributes. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 503–510. IEEE, 2011.

[103] G. Patterson and J. Hays. Sun attribute database: Discovering, annotating, and recognizing scene attributes. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2751–2758. IEEE, 2012.

[104] P. Peng, R. C.-W. Wong, and S. Y. Phillp. Learning on probabilistic labels. In *Proceedings of the 2014 SIAM International Conference on Data Mining (SDM14)*. SIAM, 2014.

[105] A. M. Qamar, E. Gaussier, J.-P. Chevallet, and J. H. Lim. Similarity learning for nearest neighbor classification. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 983–988. IEEE, 2008.

[106] A. Rakotomamonjy, F. Bach, S. Canu, Y. Grandvalet, et al. Simplemkl. *Journal of Machine Learning Research*, 9:2491–2521, 2008.

[107] S. Ramaswamy, P. Tamayo, R. Rifkin, S. Mukherjee, C.-H. Yeang, M. Angelo, C. Ladd, M. Reich, E. Latulippe, J. P. Mesirov, et al. Multiclass cancer diagnosis using tumor gene

expression signatures. *Proceedings of the National Academy of Sciences*, 98(26):15149–15154, 2001.

[108] C. E. Rasmussen. Gaussian processes for machine learning. 2006.

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

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

[111] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[112] N. Roy and A. McCallum. Toward optimal active learning through monte carlo estimation of error reduction. In *Proceedings of the Eighteenth International Conference on Machine Learning*, 2001.

[113] 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.

[114] W. J. Scheirer, M. J. Wilber, M. Eckmann, and T. E. Boult. Good recognition is non-metric. *Pattern Recognition*, 47(8):2721–2731, 2014.

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

[116] M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. *Advances in neural information processing systems (NIPS)*, 41, 2004.

[117] B. Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52:55–66, 2012.

[118] B. Settles, M. Craven, and S. Ray. Multiple-instance active learning. In *Advances in neural information processing systems*, pages 1289–1296, 2008.

[119] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294. ACM, 1992.

[120] S. Shalev-Shwartz. Online learning: Theory, algorithms, and applications. *PhD. Thesis*, 2007.

[121] S. Shalev-Shwartz, O. Shamir, N. Srebro, and K. Sridharan. Stochastic convex optimization. In *COLT*, 2009.

[122] S. Shalev-Shwartz, Y. Singer, and A. Y. Ng. Online and batch learning of pseudo-metrics. In *Proceedings of the twenty-first international conference on Machine learning*, page 94. ACM, 2004.

[123] U. Shalit, D. Weinshall, and G. Chechik. Online learning in the embedded manifold of low-rank matrices. *The Journal of Machine Learning Research*, 13(1):429–458, 2012.

[124] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948.

[125] A. Shashua and A. Levin. Ranking with large margin principle: Two approaches. In *Advances in neural information processing systems*, pages 937–944, 2002.

[126] R. N. Shepard. Metric structures in ordinal data. *Journal of Mathematical Psychology*, 3(2):287–315, 1966.

[127] M. A. Shipp, K. N. Ross, P. Tamayo, A. P. Weng, J. L. Kutok, R. C. Aguiar, M. Gaasenbeek, M. Angelo, M. Reich, G. S. Pinkus, et al. Diffuse large b-cell lymphoma outcome prediction by gene-expression profiling and supervised machine learning. *Nature medicine*, 8(1):68–74, 2002.

[128] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large scale multiple kernel learning. *The Journal of Machine Learning Research*, 7:1531–1565, 2006.

[129] A. Statnikov, L. Wang, and C. F. Aliferis. A comprehensive comparison of random forests and support vector machines for microarray-based cancer classification. *BMC bioinformatics*, 9(1):319, 2008.

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

[131] M. Takac, A. Bijral, P. Richtarik, and N. Srebro. Mini-batch primal and dual methods for svms. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1022–1030, 2013.

[132] O. Tamuz, C. Liu, S. Belongie, O. Shamir, and A. Kalai. Adaptively learning the crowd kernel. *Proceedings of the 30th International Conference on Machine Learning*, 2011.

[133] J. B. Tenenbaum, V. De Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[134] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[135] A. C. G. Title. *Rank Correlation Methods*. Kendall, Maurice and Gibbons, Jean D, 5th edition, 1990.

[136] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *The Journal of Machine Learning Research*, 2:45–66, 2002.

[137] G. Tsoumakas and I. Katakis. Multi-label classification: An overview. *Dept. of Informatics, Aristotle University of Thessaloniki, Greece*, 2006.

[138] A. Tversky. Features of similarity. *Psychological review*, 84(4):327–352, 1977.

[139] A. Tversky and I. Gati. Similarity, separability, and the triangle inequality. *Psychological review*, 89(2):123–154, 1982.

[140] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. *Speech and Audio Processing, IEEE transactions on*, 10(5):293–302, 2002.

[141] A. Ukkonen, B. Derakhshan, and H. Heikinheimo. Crowdsourced nonparametric density estimation using relative distances. 2015.

[142] M. Valko and M. Hauskrecht. Feature importance analysis for patient management decisions. *Studies in health technology and informatics*, 160(Pt 2):861, 2010.

[143] L. Van Der Maaten and K. Weinberger. Stochastic triplet embedding. In *Machine Learning for Signal Processing (MLSP), 2012 IEEE International Workshop on*, pages 1–6. IEEE, 2012.

[144] V. Vapnik. *The nature of statistical learning theory*. Springer Science & Business Media, 2013.

[145] M. Varma and B. R. Babu. More generality in efficient multiple kernel learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1065–1072. ACM, 2009.

[146] K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl. Constrained k-means clustering with background knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 577–584, 2001.

[147] C. Wah, S. Maji, and S. Belongie. Learning localized perceptual similarity metrics for interactive categorization. In *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on*, pages 502–509. IEEE, 2015.

[148] C. Wah, G. Van Horn, S. Branson, S. Maji, P. Perona, and S. Belongie. Similarity comparisons for interactive fine-grained categorization. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 859–866. IEEE, 2014.

[149] J. Wang, H. T. Do, A. Woznica, and A. Kalousis. Metric learning with multiple kernels. In *Advances in Neural Information Processing Systems*, pages 1170–1178, 2011.

[150] 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.

[151] K. Q. Weinberger, J. Blitzer, and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. In *Advances in neural information processing systems*, pages 1473–1480, 2005.

[152] K. Q. Weinberger and L. K. Saul. Fast solvers and efficient implementations for distance metric learning. In *Proceedings of the 25th international conference on Machine learning*, pages 1160–1167. ACM, 2008.

[153] K. Q. Weinberger and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. *The Journal of Machine Learning Research*, 10:207–244, 2009.

[154] K. Q. Weinberger and G. Tesauro. Metric learning for kernel regression. In *International Conference on Artificial Intelligence and Statistics*, pages 612–619, 2007.

[155] M. J. Wilber, I. S. Kwak, and S. J. Belongie. Cost-effective hits for relative similarity comparisons. In *Second AAAI Conference on Human Computation and Crowdsourcing*, 2014.

[156] E. P. Xing, M. I. Jordan, S. Russell, and A. Y. Ng. Distance metric learning with application to clustering with side-information. In *Advances in neural information processing systems*, pages 505–512, 2002.

[157] L. Yang and R. Jin. Distance metric learning: A comprehensive survey. *Michigan State Universiy*, 2, 2006.

[158] A. Yu and K. Grauman. Fine-Grained Visual Comparisons with Local Learning. In *Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[159] X. Zhu, J. Lafferty, and Z. Ghahramani. Combining active learning and semi-supervised learning using gaussian fields and harmonic functions. In *ICML 2003 workshop on the continuum from labeled to unlabeled data in machine learning and data mining*, pages 58–65, 2003.

[160] J. Zhuang, I. W. Tsang, and S. C. Hoi. Simplenpkl: simple non-parametric kernel learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1273–1280. ACM, 2009.