Large Scale Discriminative Metric Learning

Peter D. Kirchner*, Matthias Boehm**, Berthold Reinwald**, Daby Sow*, Michael Schmidt***,
Deepak Turaga* and Alain Biem*
***Columbia University Medical Center, New York, New York
**IBM Almaden Research Center, San Jose, California
*IBM T.J. Watson Research Center, Yorktown Heights, New York
{pdk,mboehm,reinwald,sowdaby,turaga,biem}@us.ibm.com
mjs2134@cumc.columbia.edu

Abstract

We consider the learning of a distance metric, using the Localized Supervised Metric Learning (LSML) scheme, that discriminates entities characterized by high dimensional feature attributes, with respect to labels assigned to each entity. LSML is a supervised learning scheme that learns a Mahalanobis distance grouping together features with the same label and repulsing features with different labels. In this paper, we propose an efficient and scalable implementation of LSML allowing us to scale significantly and process large data sets, both in terms of dimensions and instances. This implementation of LSML is programmed in SystemML with an R-like syntax, and compiled, optimized, and executed on Hadoop. We also propose experimental approaches for the tuning of LSML parameters yielding significant analytical and empirical improvements in terms of discriminative measures such as label prediction accuracy. We present experimental results on both synthetic and real-world data (feature vectors representing patients in an Intensive Care Unit with labels corresponding to different conditions) assessing respectively how well the algorithm scales and how well it works on real world prediction problems.

I. Introduction

Distance metric learning corresponds to a class of problems that has received a lot of recent interest in data mining literature [9]. There are different types of distance metrics including well known *fixed metrics* – such as Euclidean distance for numeric data – and *learning based metrics* that require learning appropriate (linear, non-linear, global or local) transformations of the data before fixed metric computation, such that the resulting metric supports the needs of a data mining task. Different types of metric learning schemes have been designed for various data mining tasks, that include supervised, unsupervised and semi-supervised problems. Metric learning schemes are also closely related to Kernel learning and dimensionality reduction problems [5] that have been studied extensively in data mining.

Supervised metric learning approaches require the presence of an annotated/labeled set of data items that define requirements on the metric to be learned. The resulting metrics allow for better separation of the data into different classes – as is the case for classification tasks. What this means is that the metric (corresponding transformation) must be chosen such that it takes small values for data items that have the same class label, and large values for data items with different class labels, allowing for a cleaner separation of these data items. Such classification tasks are very common in several applications ranging from object recognition in images, image classification and retrieval, and text document analysis. Recently, discriminative metric learning has also been applied to medical informatics [8] for the computation of patient similarity in support of diagnostic and predictive analysis tasks.

In this paper, we consider one such discriminative metric learning scheme, Localized Supervised Metric Learning [7], designed for high-dimensional data classification tasks. Based upon labeled data items, LSML calculates a projection of the data items onto a chosen reduced dimensionality d in order to optimize a "local" scatteredness objective function. This objective function captures the ratio between the average distance (after transformation) to k_{inner} nearest neighbors with the same class label, and the average distance to the k_{outer} nearest neighbors with different class label – for each data item in the training set. Minimizing this ratio then leads to naturally bringing closer neighbors with the same class label, while moving farther apart neighbors with different class labels, thereby supporting the task of discriminating among the two classes. Learning this projection requires an iterative gradient descent based optimization, that is discussed in more detail in Section II.

LSML has been shown to be very effective in solving various problems in different domains, however prior implementations with tools such as Matlab and Python rely upon the programmer using a lower level of abstraction to express the parallelization strategy (if any), which tends to constrain the scalability of the algorithm to specific sizes and dimensionality of the training dataset. With several problems now requiring the analysis of Big Data, these implementations are inadequate. In this paper, we first realize an implementation of the LSML algorithm using SystemML [4] that is designed specifically for Big Data problems, and runs on Hadoop to scale computation. SystemML provides a declarative language with an R-like syntax for algorithm developers to write programs without hardcoding problem-size or platform-specific details. SystemML compiles the programs, costs alternative execution plans, and automatically picks the most efficient execution plan given the constraints of JVM memory, data characteristics, and other infrastructure and configuration details.

Additionally, prior work on LSML has ignored the selection of the algorithm parameters d, k_{inner} and k_{outer}, requiring instead the user to choose reduced dimension and neighborhood size arbitrarily, or painstakingly determine the optimal parameterizations for their problem. In this paper, we investigate the impact of these parameters on the resulting discrimination ability, specifically in terms of classification performance. By exploiting some of the inherent parallelism in determining projections for many values of reduced dimension and neighborhood size, and by amortizing the cost of the ordered-neighbor calculations across those values, we can efficiently determine the optimum LSML parameterization for a specific training dataset.

Finally, we use the resulting implementation to determine patient similarity on a real-world dataset from a neurological Intensive Care Unit (ICU), such that patients with different complications are better discriminated by the resulting metric, allowing for improved diagnoses and predictions of patient complications.

This paper is organized as follows: In Section II we describe the LSML formulation in more detail, highlighting the essential computations, as well as algorithmic complexity. In Section III we discuss SystemML, and present some details of our implementation of the LSML algorithm. We then present experimental evaluations of our work in Section IV where we discuss the impact of the different LSML parameters and classification performance on real-world patient datasets. We conclude in Section V.

II. Localized Supervised Metric Learning

The notion of "similarity" is intrinsically subjective. In a healthcare setting, while looking for similarities across patients, physicians often have an objective in mind. They do not base their comparisons exclusively on quantitative measurements such as lab results and physiological sensor measurements. They naturally take into account many additional clinical factors characterizing their cohorts of patients, such as demographics and disease history. To capture this subjective clinical notion of similarity, we learn a similarity distance metric using a *Localized Supervised Metric Learning* (LSML) scheme as described in [8]. This learning process is done in a supervised way, based on labels provided by physicians. LSML automatically adjusts the importance of each numeric feature to produce a similarity metric attempting to cluster patients with similar labels together and discriminate patients with different labels.

Once the metric is learned, it can be used in several ways. All the features can be clustered, using any well known clustering techniques such as the k-means clustering algorithm with the learned metric in attempt to discover structural properties in the data set. Classifiers can also be built either by a direct application of instance-based classification schemes like the K-Nearest Neighbor algorithm or using standard statistical classification schemes after transformation of the feature set into a new space according to the projection matrix derived from the metric learned.

Formally, let feature vectors be represented by a N-dimensional feature vector x. Such vectors may represent patients in a healthcare setting. Let x_i represent a feature vector for an entity i. Our goal is to learn a generalized Mahalanobis distance between feature vectors x_i and x_j defined as:

$$d_m(x_i,x_j) = \sqrt{(x_i - x_j)^\top \mathbf{P}(x_i - x_j)} \qquad (1)$$

where $P \in R^{N\times N}$ is called the *precision matrix*. Matrix P is positive semi-definite and is used to incorporate the correlations between different feature dimensions. The key is to learn the optimal P such that the resulting distance metric has the following properties:

- *Within-class compactness:* patients of the same label are close together;
- *Between-class scatteredness:* patients of different labels are far away from each other.

To formally measure these properties, we use two kinds of neighborhoods as defined in [11]: The homogeneous neighborhood of x_i, denoted as $\mathscr{N}_i^o$, is the k_{inner}-nearest patients of x_i with the same label. The heterogeneous neighborhood of x_i, denoted as $\mathscr{N}_i^e$, is the k_{outer}-nearest patients of x_i with different labels. In the rest of this paper, we assume that $k_{inner} = k_{outer} = k$, unless specified otherwise.

We define the local compactness $\mathscr{C}$ of point x_i from these two neighborhoods as:

$$\mathscr{C}_i = \sum_{x_j \in \mathscr{N}_i^o} d_m^2(x_i,x_j) \qquad (2)$$

Similarly, let $\mathscr{S}_i$ denote the local scatteredness of point x_i. $\mathscr{S}_i$ is defined as:

$$\mathscr{S}_i = \sum_{x_l \in \mathscr{N}_i^e} d_m^2(x_i,x_l) \qquad (3)$$

The discriminability of the distance metric d_m is defined as

$$\mathscr{J} = \frac{\sum_i \mathscr{C}_i}{\sum_i \mathscr{S}_i} = \frac{\sum_i \sum_{x_j \in \mathscr{N}_i^o} (x_i - x_j)^\top \mathbf{P}(x_i - x_j)}{\sum_i \sum_{x_l \in \mathscr{N}_i^e} (x_i - x_l)^\top \mathbf{P}(x_i - x_l)} \qquad (4)$$

We aim at finding a P that minimizes $\mathscr{J}$. This is equivalent to minimizing the local compactness and maximizing the local scatteredness simultaneously. In contrast with linear discriminant analysis [3], which seeks for a discriminant subspace in a global sense, the localized supervised metric aims to learn a distance metric with enhanced local discriminability.

Since P is a low-rank positive semi-definite matrix, we can decompose the precision matrix as $P = W \cdot W^T$, where $W \in R^{N\times d}$ and $d \leq N$. The distance metric can be rewritten as $d_m(x_i,x_j) = \|W^T x_i - W^T x_j\|$. Therefore, the distance metric is equivalent to the Euclidean distance over the low-dimensional projection $W^T k$. Furthermore, as explained by Wang, *et al.* [10] and Jai, *et al.* [6], the minimization of $\mathscr{J}$

can be rewritten as a maximization problem where we are seeking the optimal transformation W^* as follows:

$$W^* = \arg \max_{WW^T=I} \frac{Tr[W^T S_p W]}{Tr[W^T S_l W]} \qquad (5)$$

Here S_p and S_l are respectively called the *penalty matrix* and *similarity matrix* and can be derived respectively from local scatteredness $\mathscr{S}$ and the local compactness $\mathscr{C}$. This reformulation of the problem allows us to map it into a well studied trace ratio minimization problem where finding W^* can be done numerically using the Decomposed Newton's Method as shown in [6].

III. System Description

Figure 1 shows a block diagram of our implementation of the LSML algorithm in scripts written for SystemML. The SystemML compiler parses our scripts and represents the computation as Directed Acyclic Graphs (DAGs) of operators. The compiler applies optimizations such as common subexpression elimination, operator reordering, operator selection, and piggybacking to group multiple instructions (runtime operators) into a small number of Map-Reduce jobs. Instructions in SystemML range from sequential, in-memory implementations running on a single machine to distributed implementations running in Map-Reduce on large clusters. The MR instructions operate on blocks of matrices, where individual matrix blocks can be in dense or sparse representations. At runtime, the SystemML control program (CP) executes the single node instructions and also drives the execution of the Map-Reduce jobs. The SystemML control program maintains a multi-level buffer pool to reuse/evict in-memory datasets and exchange datasets with operators running in MapReduce.

The compiler selects operators based on a cost model which uses data characteristics and worst-case memory estimates that guarantee hard memory constraints to avoid out-of-memory situations. For missing data characteristics, the compiler sets re-optimization hooks that use current data characteristics at runtime to regenerate execution plans. As data sizes grow to the extent that they don't fit in a single node computation, the SystemML compiler selects operators to run on the MapReduce side.

The LSML scripts exploit the SystemML ParFOR language construct for task parallelism [1]. Assuming no loop-carried dependencies, each iteration of a ParFOR may be executed independently and in parallel. Conceptually, SystemML groups iterations into tasks, the tasks are executed by workers, and the worker results are merged into the final results. SystemML supports various runtime strategies for task partitioning methods (e.g., fixed-size, factoring), parallel workers (e.g., local multi-core, remote MapReduce tasks), result merge strategies, as well as optimizations for data access-aware data partitioning and locality. Based on data and system characteristics, the SystemML optimizer then estimates costs of alternative execution strategies to automatically create efficient parallel execution plans. Example ParFOR rewrites are operator selection such as parallel worker execution types and configuration changes such as the degree of parallelism.

Our implementation of LSML in SystemML consists of two parts that communicate through the Hadoop Distributed File System (HDFS): the neighborhood computation and the LSML projection optimization as shown in Figure 1. This partitioning is not required, but it does facilitate studying scaling of the different parts of the calculation, independent development and debugging of those parts, and reuse of intermediate results when desired. The neighborhood computation consists of two scripts sharing results on HDFS. The first script implements a Euclidean distance computation between each feature vector in the data set. This *orderedDistances* script produces two matrices of size $n_t \times n_t$ where n_t is the total number of training cases processed. The matrix elements record for within- and between-classes, respectively, the ordering of nearest-neighbors. An element of the distance matrix at position (i, j) records the rank of feature vector j in the ordered list of nearest neighbors for feature vector i, i.e. 1 if the (i, j) are nearest-neighbors. This encoding allows the second script, *sMatrixCalculation*, to calculate and store the LSML algorithm's penalty and similarity matrices, S_p and S_l in parallel over a specified range of k.

The second part of the implementation of LSML consists of a script computing optimum transformations W over a range of reduced dimensionality values d, using the S_p and S_l matrices from the neighborhood computation. This script performs matrix eigenvalue decompositions and the Decomposed Newton's Method to converge to optimum transformations.

The eigendecomposition is currently implemented as an external user-defined function which wraps calls through java to the java translation of LAPACK, jlapack, which is not parallelized. DSYEV is currently what we use for eigenvalue decomposition. On the final Decomposed Newton's Method iteration, assuming convergence, the d eigenvectors corresponding to the algebraically largest eigenvalues form the optimum projection for the given d and k. The trace ratio calculated for a given d and k could be used to speed convergence at the next lower d, but this would limit the available parallelism by introducing a dependency.

Figures 2 and 3 show snippets of LSML scripts demonstrating the ease of programming Big Data analytics in SystemML. Figure 2 shows how distances are currently evaluated as part of the neighborhood computation while Figure 3 shows the heart of the Decomposed Newtons Method iteration including the invocation of an external function for eigendecomposition. Figure 3 shows a fully vectorized expression for gradient calculation (cf grad =) that epitomizes the what not how declarative programming style that does not constrain the SystemML optimizer in rewriting and planning. Figure 2 includes a nested parfor construct where we have reverted to a more imperative mode of expression in explicitly specifying the triangular, row-column computation of distances. However, the SystemML optimizer

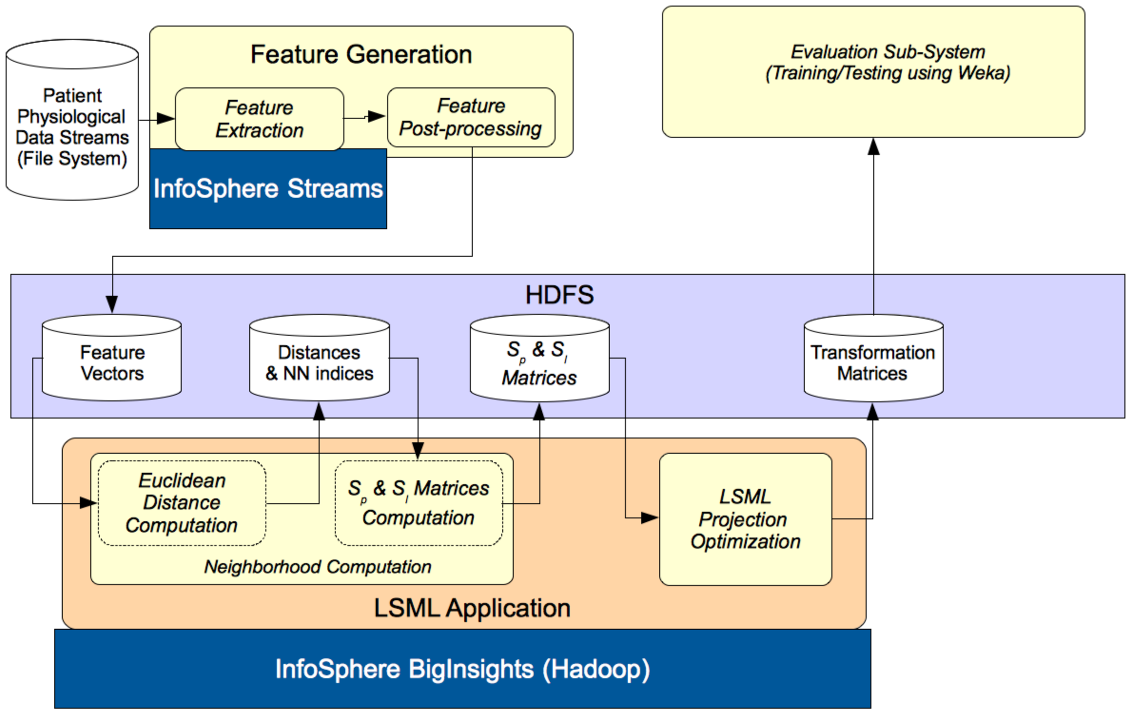
Feature Generation
Patient Physiological Data Streams (File System)
Feature Extraction
Feature Post-processing
InfoSphere Streams
Evaluation Sub-System (Training/Testing using Weka)
HDFS
Feature Vectors
Distances & NN indices
S_p & S_l Matrices
Transformation Matrices
Euclidean Distance Computation
S_p & S_l Matrices Computation
Neighborhood Computation
LSML Projection Optimization
LSML Application
InfoSphere BigInsights (Hadoop)

Fig. 1. Overall system architecture.

```
...
# calculate half of the symmetric
# square-distance matrix
# for all training cases
alldist = matrix(0, rows=n_t, cols=n_t);
parfor(r in 1:(n_t-1)){
  Xr = X_train[r,];
  myd = matrix (0, rows = 1, cols = n_t)
  parfor(c in (r+1):n_t) {
    myd[1,c] =
        sum((Xr - X_train[c,])^2);
  }
  alldist[r,] = myd;
}
alldist = alldist + t(alldist);
...
```

Fig. 2. Snippet of LSML script for the neighborhood computation.

can compile this task parallel formulation into distributed in-memory operations exploiting the full degree of cluster parallelism. Furthermore, even if X_train and alldist are too large to fit in memory of remote tasks, the optimizer will automatically apply data and result partitioning according to the row-wise access pattern. Thus, it still ensures good scalability for large numbers of instances.

IV. Experimental Evaluations

We have performed several experiments to measure the compute scaling and analytical performance of the imple-

```
...
# Decomposed Newton's Method iteration.
S = Sp - (St * lambda);
S = 0.5 * (S + t(S));
[D,V] = eigen(S);
grad = t(diag(t(V)%*%St%*%V));
next_lambda=
      getexpectation(lambda,D,grad,d);
...
```

Fig. 3. Snippet of LSML script for the projection optimization.

mentation presented in Section III. We focused exclusively on the performance of the LSML implementation on Hadoop. For these tests, we used a 5 node cluster running IBM Hadoop Cluster 1.3. Each node is an IBM model HS22 7870-AC1 with 64GB memory, dual quad-core 3.6 GHz Intel P4 x5687 Xeon processors with hyperthreading enabled. This appears as a total of 16 cores to the node's operating system, and Hadoop is configured accordingly at 16 cores/node. Each node has two 500GB SAS hard drives in a raid 0 configuration for both local and HDFS storage. Inter-node communication is provided by both gigabit ethernet and 10Gb/sec Infiniband interconnects. Java and map-reduce were configured for 2 GB JVM memory. SystemML was configured with JVM reuse enabled, a default number of reducers of 10, and a memory budget ratio of 0.7.

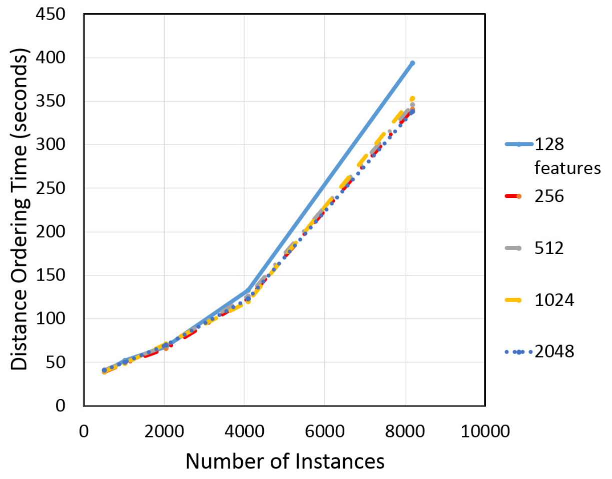

Fig. 4. LSML Euclidean distance neighbor-ordering time versus the number of instances for selected numbers of features.

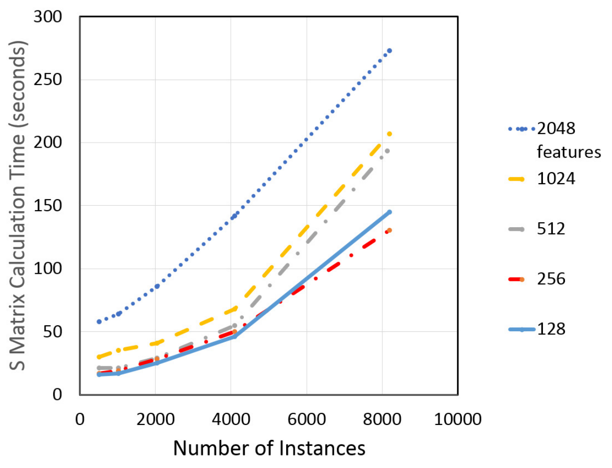

Fig. 5. Time to calculate the penalty and similarity matrices S_p and S_l as a function of the number of instances in the training set, for several different feature dimensions and all neighborhood sizes from 1 to 64.

A. Scaling Experimental Results

As presented in Section III, our implementation of LSML has two main components: the neighborhood computation and the projection optimization. To assess the scaling performance of these components, we synthetically generated feature sets by augmenting an existing real patient data set provided by the Columbia Medical Center with randomly generated features.

Figure 4 shows the time, including file i/o, to order inner and outer neighbors by Euclidean distance as a function of the number of training instances n_t. This figure shows five series corresponding to different numbers of features per instance (n_f). In all five series, we observed that the calculation time increases superlinearly with the number of instances, reflecting the ($O(n_t^2)$) cost of computing pairwise distances (linearly dependent upon n_f), and the cost of sorting which is independent of n_f. Therefore there is only weak dependence upon n_f observed for this portion of the neighborhood calculation.

Figure 5 shows the time required, including file i/o, to calculate the S matrices as a function of the total number of training instances n_t for neighborhood sizes k from 1 to 64. This figure depicts five series corresponding to different feature dimensions. In all five series, computing the S matrices increases superlinearly with the number of training instances, and also increases with the number of features (the dimensionality of the S matrices is n_f x n_f). There is also some apparent variability in the time taken for the calculation to complete, leading to the intersection of two of the series.

Figure 6 shows the projection optimization time as a function of n_f, the dimension of our feature vectors. As expected, this relationship converges to a cubic dependency due to the gradient calculation in the Newton's method iteration. The projection computation has no dependency on the number of training instances (cases), but because this additional timing data was collected versus n_t, it is shown here to illustrate the extent to which the runtimes are reproducible for a given n_f.

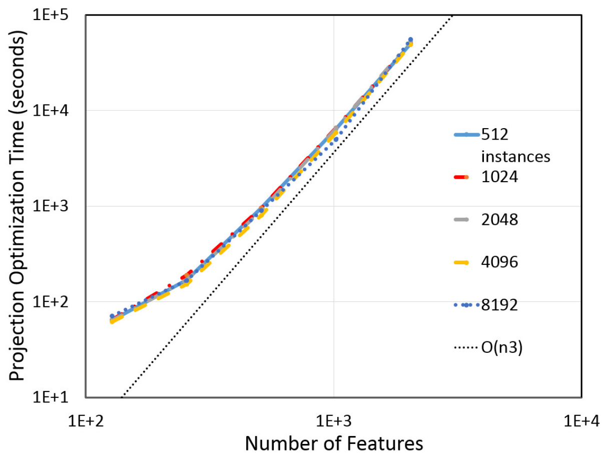

Fig. 6. Projection optimization time versus feature dimension over neighborhood sizes from 1 to 64 and reduced simension from 2 to 32. Execution time converges to a n_f^3 dependency reflecting the $O(n^3)$ dependence of the gradient calculation in Newton's Method. No dependence on number of training cases n_t is expected for this portion of the LSML computation.

B. Analytical Experimental Results

A series of experiments has been performed to assess the analytical performance of the algorithm in its ability to discriminate feature vectors. The methodology used for these experiments is shown on Figure 7. For these experiments, we used a real-world data set consisting of physiological time-series data from 295 patients from the Columbia Medical Center Neuro ICU. Cases had a binary label whose value indicated whether or not the patient had developed secondary complications. Patients were temporally aligned by time of injury (or admission). We produced a feature set for LSML training from the features available 24 hours prior to development of the complication. Based upon the label, LSML will compute projections that should help analytics predict which future patients will develop complications in the next 24 hours.

We selected 72 features which were available for all

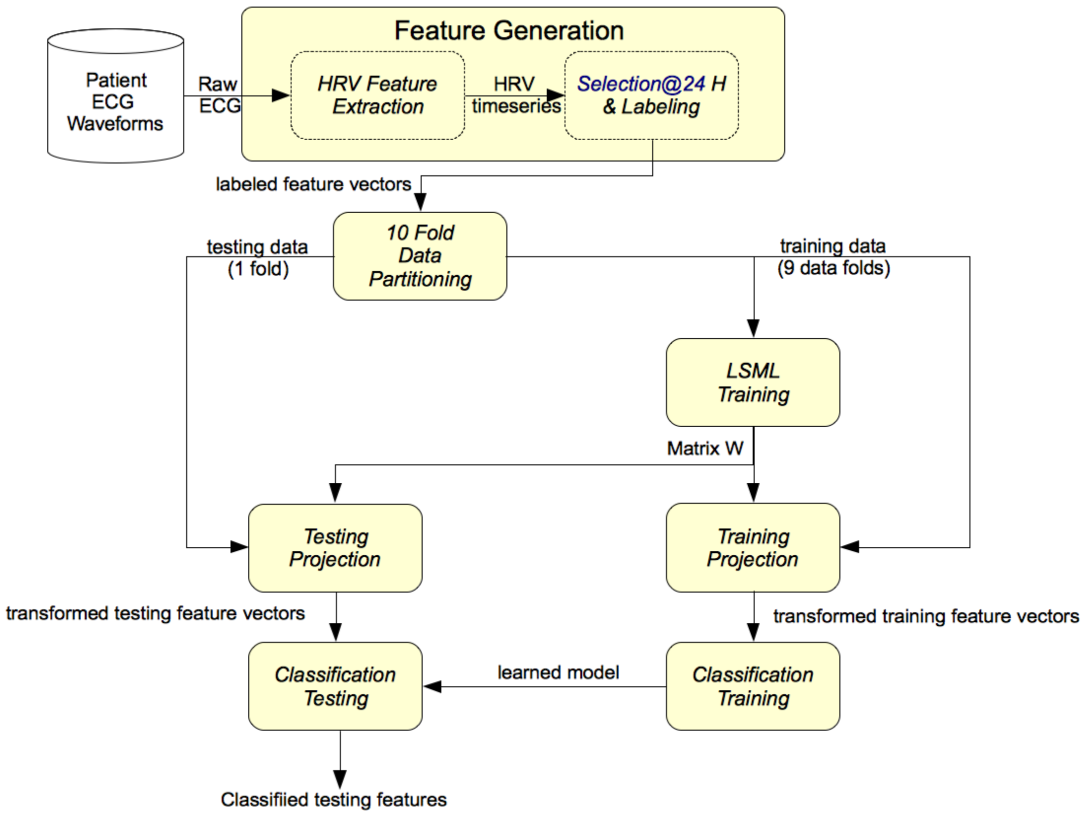

Fig. 7. Approach used to classify for secondary complications

patients. Most of these features were Heart Rate Variability (HRV) [2] statistics extracted from continuous electrocardiogram waveforms sampled at 240 Hz. These waveforms were condensed to HRV statistics via analytics deployed on our stream computing platform (IBM InfoSphere Streams version 3.0). For LSML, all features were further post-processed and Z-normalized to produce feature vectors on a per patient basis. Only physiological data recorded 24 hours before the complication diagnosis were used in the feature generation. The output of these steps is a labeled feature set that we partition into 10 stratified folds for cross validation. In cross-validation, we hold out one fold for testing and use the remaining 9 folds for training. We then repeat for each testing fold, so that training and testing occur with all the data, but no training case is used for testing in any given run. The folding is maintained throughout the subsequent analytic evaluation.

We evaluated projection matrices W by applying the projection to the corresponding folds and then applying several standard machine learning algorithms to the transformed training and testing datasets. For this study we used a selection of Weka version 3.6.10 classifiers (some invoked with a selection of parameterizations), but could have additionally or alternatively used a somewhat different set of classifiers in SystemML. Following standard cross validation techniques we repeated these experiments for all 10 folds and measured the performance of the classification.

The computational efficiency of our implementation of the LSML algorithm on Hadoop allowed us to run this experiment for many combinations of d and k, where d is the reduced dimension after LSML projection and k is the neighborhood size. For these tests, d took all values between 2 and 45. k_{inner} and k_{outer} were set to be equal at k which took all values from 1 to 104. For the 10 folds, the Decomposed Newton's Method converged within 20 iterations 38,088 times (out of 41,184), producing dimension-reducing transformations. The total computation (Ordered Distances, S Matrix Computation, and Projection Optimization, plus writing the transformed training and testing datasets to disk for subsequent classification experiments) took just over 61 minutes (3673 seconds) on the 5 node, 80 core cluster. The amortization of computation (e.g. neighborhood calculation) and available parallelism with multiple values of d and k is clearly beneficial, as the $per-transform$ cost is 96 $milliseconds$ in average.

We subsequently systematically evaluated the performance of several different classifiers for these tests. To each of the 38,088 transformed and folded training sets produced, 9 different classifier/parameter combinations were applied producing 342,792 models. Each model was applied to the single corresponding transformed testing fold. If results were available for all folds, the test results were then combined across folds, producing classification prediction results covering all instances. Each set was unique in its combination

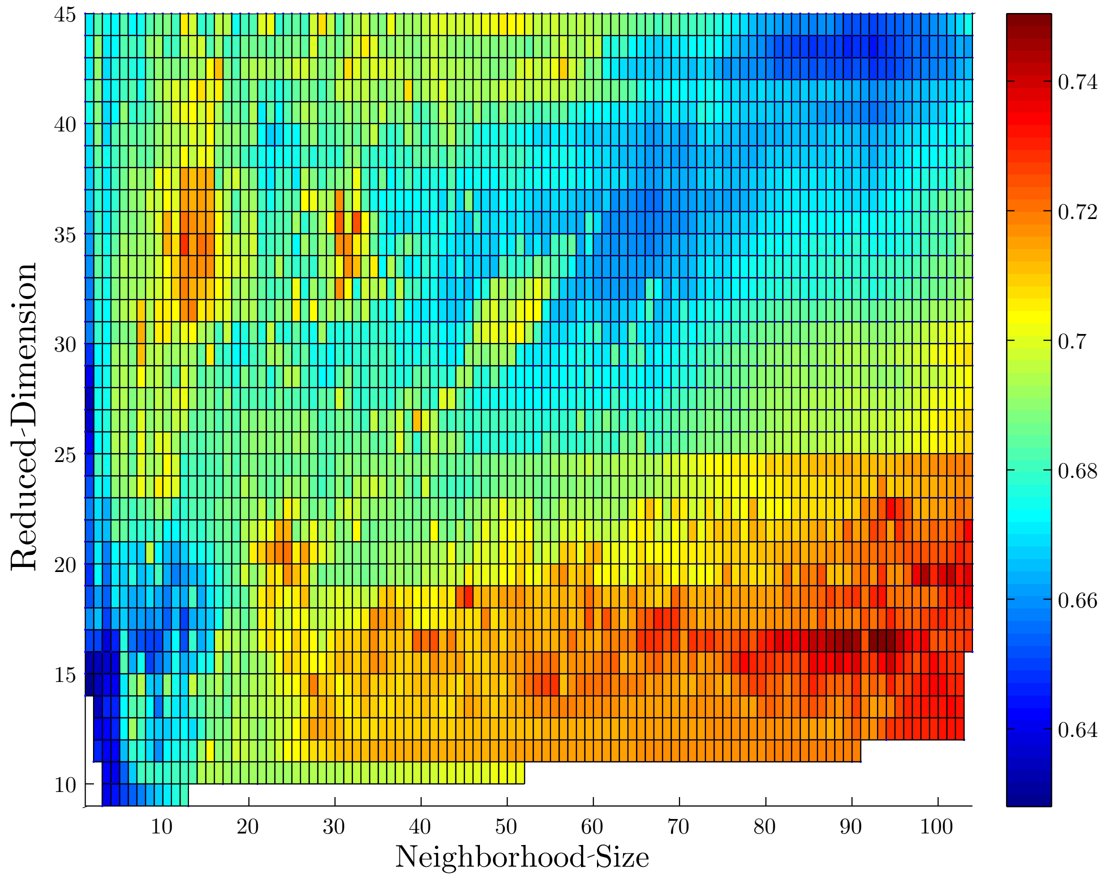

Fig. 8. Complication Classification Area Under the Curve for different values for d (reduced dimension) and k (neighborhood size).

of d, k, classifier and classifier parameter set; 29,412 full sets of prediction results were produced in this experiment. The results were processed to calculate the receiver operator curve and the corresponding Area Under the Receiver Operating Characteristic Curve (AUC). Figure 8 shows the best AUC obtained for a given pair (d,k) (this figure does not reveal the classifier and parameterization that produced this performance). This surface allowed us to exhaustively identify the best values for d and k for this data set. Table I lists each classifier evaluated in column 1, and gives its performance on the untransformed dataset in column 2. The third column selects the best classification results for this specific algorithm, and the fourth and fifth columns present the (d,k) value at which this result was achieved. We note significant improvement in AUC with application of LSML to the feature set before classification. For some classifiers, the improvement is profound.

Algorithm	AUC_{noLSML}	AUC_{LSML}	$d_{optimal}$	$k_{optimal}$
Naive Bayes -D	0.542	0.750	16	92
Logistic Regression	0.696	0.729	16	104
Naive Bayes Kernel	0.691	0.721	20	24
Random Forest	0.625	0.711	20	96
Decision Tree	0.606	0.705	16	104

TABLE I
CLASSIFICATION PERFORMANCE ON NEURO-ICU DATA SET BEFORE AND AFTER LSML TRANSFORMATION.

V. CONCLUDING REMARKS

In conclusion, we have developed and evaluated a scalable, distributed system for Localized Supervised Metric Learning that includes automatic optimization for problem size and available computational resources. The system is capable of efficient, exhaustive parallel exploration of the range of input parameterizations of the LSML algorithm for which the problem- and data-dependent optimum values are not known *a priori*. We show scalability of the system across a range of problem sizes, varying the number of cases and number of features. We apply LSML to learn patient similarity, and improve classifier performance, in order to predict medical diagnosis of complications in Neurological Intensive Care Unit patients from data available 24 hours prior to diagnosis, a window of clinical significance where prediction has the potential to improve treatment outcomes. Full exploration of neighborhood size and reduced dimension permits optimum selection of these LSML input parameters. Currently this selection is based upon impact of these parameters on classifier performance. SystemML, with an R-like syntax, makes the system more accessible to, and adaptable by, data analysts, while the declarative programming style allows the underlying SystemML to optimize the generated code for scalability, given the variability in the explored datasets.

There are several directions for future work. These include evaluating the LSML implementation in SystemML on large-scale problems in other domains to evaluate the scalability, and accuracy-performance tradeoffs more comprehensively. Given the non-trivial dependence of the algorithm on parameters d and k, there is an opportunity to develop efficient search strategies to identify appropriate settings of these parameters for new problems. While the current work uses the same value k for k_{inner} and k_{outer}, separate parameterization could be explored. We also anticipate extensions of the algorithm for cases with multiple classes in the data (as opposed to binary problems), as well as scenarios with multiple types of labels associated with each data item. A patient may have co-morbidities, where such multi-attribute labels need to be considered appropriately to determine similarity. Extensions to fully integrate LSML into streaming scenarios, where data items, features describing data items, or labels about these items arrive incrementally over time, are important in real-world settings.

References

[1] M. Boehm, S. Tatikonda, B. Reinwald, P. Sen, Y. Tian, D. R. Burdick, and S. Vaithyanathan. Hybrid parallelization strategies for large-scale machine learning in SystemML. In *Proceedings of the VLDB Endowment (PVLDB)*, 7(7), 2014.

[2] G. Clifford. *Signal Processing Methods For Heart Rate Variability,DPhil. Thesis*. Oxford University, 2002.

[3] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, San Diego, California, 1990.

[4] A. Ghoting, R. Krishnamurthy, E. Pednault, B. Reinwald, V. Sindhwani, S. Tatikonda, Y. Tian, and S. Vaithyanathan. Systemml: Declarative machine learning on mapreduce. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ICDE '11, pages 231–242, Washington, DC, USA, 2011. IEEE Computer Society.

[5] J. Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[6] Y. Jia, F. Nie, and C. Zhang. Trace ratio problem revisited. *IEEE Transactions on Neural Networks*, 2009.

[7] J. Sun, D. Sow, J. Hu, and S. Ebadollahi. Localized supervised metric learning on temporal physiological data. In *Proceedings of the 2010 20th International Conference on Pattern Recognition*, ICPR '10, pages 4149–4152, Washington, DC, USA, 2010. IEEE Computer Society.

[8] J. Sun, D. Sow, J. Hu, and S. Ebadollahi. A system for mining temporal physiological data streams for advanced prognostic decision support. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 1061–1066, Washington, DC, USA, 2010. IEEE Computer Society.

[9] F. Wang and J. Sun. Distance metric learning in data mining part 1 and 2. In *SIAM SDM Tutorial*, 2012.

[10] F. Wang, J. Sun, T. Li, and N. Anerousis. Two heads better than one: Metric+active learning and its applications for it service classification. In *ICDM*, 2009.

[11] F. Wang and C. Zhang. Feature extraction by maximizing the neighborhood margin. In *CVPR*, 2007.