2012 IEEE 12th International Conference on Data Mining

GPU-Accelerated Feature Selection for Outlier Detection using the Local Kernel Density Ratio

Fatemeh Azmandian
ECE Department
Northeastern University
Boston, USA
fazmandi@ece.neu.edu

Ayse Yilmazer
ECE Department
Northeastern University
Boston, USA
yilmazer@ece.neu.edu

Jennifer G. Dy
ECE Department
Northeastern University
Boston, USA
jdy@ece.neu.edu

Javed A. Aslam
CCIS Department
Northeastern University
Boston, USA
jaa@ccs.neu.edu

David R. Kaeli
ECE Department
Northeastern University
Boston, USA
kaeli@ece.neu.edu

***Abstract*—Effective outlier detection requires the data to be described by features that capture the behavior of normal data while emphasizing those characteristics of outliers which make them *different* than normal data. In this work, we present a novel non-parametric evaluation criterion for filter-based feature selection which caters to outlier detection problems. The proposed method seeks the subset of features that represent the inherent characteristics of the normal dataset while forcing outliers to stand out, making them more easily distinguished by outlier detection algorithms.**

Experimental results on real datasets show the advantage of our feature selection algorithm compared to popular and state-of-the-art methods. We also show that the proposed algorithm is able to overcome the small sample space problem and perform well on highly imbalanced datasets. Furthermore, due to the highly parallelizable nature of the feature selection, we implement the algorithm on a graphics processing unit (GPU) to gain significant speedup over the serial version. The benefits of the GPU implementation are two-fold, as its performance scales very well in terms of the number of features, as well as the number of data points.

***Keywords*-Feature Selection; Outlier Detection; Imbalanced Data; GPU Acceleration**

I. Introduction

An integral part of any data mining task is having a good set of features that can be used to accurately model the inherent characteristics of the data. In practice, the best set of features is not known in advance. Therefore, a pool of candidate features are collected and processed to removed irrelevant and redundant features. This can improve both the memory and computational cost of the data mining algorithm, as well as the accuracy of the learner. Reducing the space of possible features is done in two ways: feature transformation and feature (subset) selection. In the former, the original space of features is transformed into a new feature space, as in Principal Components Analysis (PCA).

In the latter approach to reducing the size of the feature space, the original set of features remain unchanged and a subset of those features are selected. A simple way to perform feature selection is to use a feature evaluation function, such as relevance [1], to rank the features on an *individual* basis. Then, a subset of the features are selected by taking the top-ranked features (for example, the top m features). Another feature selection methodology is to search the space of feature subsets and select the subset that optimizes a criterion function. For a dataset with d features, there are 2^d possible subsets of features. For even a moderate value of d, an exhaustive search would be too computationally expensive, so it is common to use a greedy search strategy such as sequential forward selection or backward elimination [2].

In addition to the search strategy, the other important component of feature selection is the criterion to be optimized. A brute force way to evaluate features is to utilize the classifier that will ultimately be used. Such a method is called a wrapper approach [3]. Another, less computationally expensive, way is to evaluate features based on some criterion function. This is referred to as a filter method. Existing criteria include measures based on distance, information, dependency, and consistency [4]. A third approach is to perform feature selection as part of the learning task, known as an embedded method, such as a decision tree.

In this work, we present a novel optimization criterion inspired by outlier detection problems where the data is highly imbalanced and outliers comprise a small portion of the dataset. *The criterion tries to find the set of features that maximize the density of normal data points while minimizing the density of outliers.* In other words, it seeks feature subsets wherein normal data points fall in high-density regions and outliers fall in low-density regions of the feature space. The goal is to make outliers stand out more prominently from normal data points, which allows outlier detection algorithms to more easily identify them.

Most of the work on dimensionality reduction for outlier detection have tackled the problem using a feature transformation approach. In these methods, a projection of the feature space is sought which can be used to detect outliers. While this approach subsumes feature selection (i.e., projecting onto the original feature axes is equivalent to selecting a subset of the features), there is a case to be made for the understandability that comes with feature subset selection as opposed to linear or non-linear combinations of

1550-4786/12 $26.00 © 2012 IEEE
DOI 10.1109/ICDM.2012.51

the features. Retaining a subset of the original features goes a long way towards understanding the nature of the underlying data and the features that contribute to an outlier's deviant behavior. This can be advantageous in domains such as fraud detection and intrusion detection, in which anomalous activity can be narrowed down to a subset of the collected features.

In addition to proposing a feature selection algorithm that seeks to intrinsically enhance the quality of outlier detection, an important contribution of this paper is the implementation of the algorithm on a graphics processing unit (GPU) and the substantial speedup that is acquired. GPUs are massively parallel floating point processors attached to dedicated high speed memory, at a fraction of the cost of traditional parallel processing computers. They are designed for compute-intensive, highly data-parallel computation and rely on multithreading to provide throughput-oriented performance. Therefore GPUs are well suited for data-parallel applications, resulting in large improvements in running time.

The remainder of the paper is organized as follows. In section II, we discuss related work and in section III, we describe the details of our proposed local kernel density ratio feature selection algorithm. In section IV, we present the results of our feature selection algorithm on several real-word datasets, including an analysis of the performance gained by a GPU-based implementation. Finally, section V concludes the paper and presents directions for future work.

II. Related Work

Feature selection is a very important and well-studied problem in data mining. Most of the work have focused on the area of feature selection for classification and regression [3]–[7], and as far as we know, there has been no work done to create a feature selection algorithm that caters specifically to outlier detection problems. In our work, we propose the first feature selection algorithm that takes the subsequent task of outlier detection into consideration and chooses features to intrinsically enhance the identification of outliers.

Chen et al. [8] developed a ranking-based feature selection algorithm for classification of high-dimensional datasets that suffer from the "small sample space" problem and whose class labels are highly imbalanced – the latter being a characteristic inherent in outlier detection. Recent work that look for outliers in high-dimensional datasets deal with the issue of high dimensionality in different ways. Aggarwal et al. [9] use an evolutionary search technique to find multiple lower dimensional projections of the data which are locally sparse in order to detect outliers. Other methods perform feature transformation, rather than feature selection, for outlier detection [10].

There has also been some work done that use the idea of *a ratio of densities* to directly perform outlier detection. Hido et al. [11] use the ratio of the density of a data point in the training set to its density in the test set as a measure of the degree to which the point is an *inlier*, as opposed to an outlier. Their training set consists of only normal points and the test set consists of both normal points and outliers. To deal with high-dimensional data, Sugiyama et al. [12] use a projection matrix to find the low-dimensional subspace in which the two densities are significantly different from each other. In [13], the novelty of data points in one distribution are assessed relative to another distribution based on the log-likelihood ratio of the two distributions. In our work, we use a ratio of densities to perform feature selection, with the distinction that our method uses the notion of *local* neighborhoods to measure densities. In the denominator, we utilize the density of only outliers. This ensures that we pick features in which outliers become even more conspicuous as they will be represented in low-density regions of the feature space. For the outlier detection, we take a one-class learning approach and distinguish outliers using well-established methods. In this regard, once features are chosen by our feature selection technique, the outlier detection algorithms proposed in [12] and [13] can be used as alternative methods for identifying outliers. In the next section, we present the details of our feature selection algorithm.

III. Local Kernel Density Ratio Feature Selection

The inspiration for our feature selection algorithm came from an approach taken to solve the outlier (or anomaly) detection problem. Hawkins [14] describes an outlier as "an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism." In outlier detection problems, outliers typically comprise a small portion of the dataset. Examples include intrusion detection data, where malicious attacks are less frequent than normal activity, and certain tumor datasets where patients with a particular type of cancer are few compared to the number of healthy patients. In some cases, obtaining samples from the outlier class may be difficult or expensive, such as a fault detection system where it can be expensive to obtain outlier data representing the faulty situations of a machine. Therefore in the outlier detection domain, learners must deal with highly imbalanced data. Next, we describe our criterion function for feature selection which caters to outlier detection problems and is insensitive to the degree of imbalance in the data as it is based on a ratio of *average* normal to outlier densities.

A. Local Kernel Density Ratio Criterion

In our work, we propose a novel feature selection criterion for outlier detection which tries to find the set of features that best describes the normal class while ensuring that outliers "stand out". While most outlier detection techniques take an unsupervised approach, it is not uncommon to have samples

that belong to the outlier class. For example, in intrusion detection there may be many instances of malware attacks and malicious executions that can provide guidance as to how normal executions differ from malicious ones. In our case, we utilize the supervised information to select features which are intrinsically suitable for outlier detection. Normal data points come from the same distribution while outliers can be any point, from a completely different distribution. Taking advantage of information from both normal and outlier points (if available) is more powerful than normal alone. The aim is to have something to compare the normal distribution against and find out what separates outliers from the normal data. Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n]$ represent a dataset with n data points where each data point $\mathbf{x} \in \mathbb{R}^d$ has d features and is associated with a class label $y \in \{-1, +1\}$, where -1 denotes the normal class and $+1$ denotes an outlier.

The Local Outlier Factor (LOF) algorithm [15] solves the outlier detection problem using a ratio of densities. In the LOF algorithm, the density of a data point is compared to that of its neighbors and based on this, the point is assigned a *degree* of being an outlier, known as its *local outlier factor*. The LOF of a data point is calculated as the average density of data points within its neighborhood divided by its own density. When a data point has a low density compared to points in its neighborhood, it is more likely to be an outlier. Conversely, outliers should have a lower density compared to its neighbors. Therefore, it stands to reason that a feature set which emphasizes this phenomenon would facilitate the detection of outliers.

To test this hypothesis, we developed a criterion that measures the quality of features based on the density induced for normal and outlier data points. More specifically, to maximize the density of normal data points while minimizing the density of outliers, the criterion function takes the ratio of the two, with a focus on the *local* neighborhood density of each data point. To measure density, we make no assumptions about the form of the underlying distribution of the data. Instead, we take a non-parametric approach and calculate the kernel density estimate of the data points with a Gaussian kernel. The objective is to find the optimal set of features $\mathbf{w}^*$ that maximizes the described criterion function $\mathcal{J}(\mathbf{w})$:

$$\mathbf{w}^* = \underset{\mathbf{w} \in \{0,1\}^d}{\arg\max}\, \mathcal{J}(\mathbf{w}) \tag{1}$$

We formally define the criterion function as:

$$\mathcal{J}(\mathbf{w}) = \frac{\frac{1}{|\mathbf{X}_-|} \sum\limits_{\mathbf{x}_- \in \mathbf{X}_-} \sum\limits_{\mathbf{x} \in N_k(\mathbf{x}_-)} K(\mathbf{w} \circ \mathbf{x}_-, \mathbf{w} \circ \mathbf{x})}{\frac{1}{|\mathbf{X}_+|} \sum\limits_{\mathbf{x}_+ \in \mathbf{X}_+} \sum\limits_{\mathbf{x} \in N_k(\mathbf{x}_+)} K(\mathbf{w} \circ \mathbf{x}_+, \mathbf{w} \circ \mathbf{x})} \tag{2}$$

In the above equation, K is a kernel function. In our experiments, we use the Gaussian (or Radial Basis Function) kernel:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\| \mathbf{x} - \mathbf{x}' \|^2}{2\sigma^2}\right) \tag{3}$$

The vector $\mathbf{w} = (w_1, w_2, ..., w_d)$ is a binary vector signifying which features are selected; for $(j = 1, ..., d)$, $w_j = 1$ denotes the presence of feature j and $w_j = 0$ denotes its absence. We use $\mathbf{x}_- \in \mathbf{X}_-$ and $\mathbf{x}_+ \in \mathbf{X}_+$ to represent data points from the normal and outlier class, respectively. The parameter k determines the size of the local neighborhood of a data point, σ is the width of the Gaussian kernel, and the symbol $\circ$ represents the Hadamard product [16]. The Hadamard (or Schur) product of two matrices is their element-wise product.

The size of the local neighborhood of a data point $\mathbf{x}$ is determined by the distance to its k^{th}-nearest neighbor, referred to as its k-distance. All of the data points whose distance to $\mathbf{x}$ is less than this distance comprise its k-distance neighborhood, $N_k(\mathbf{x})$. The k-distance neighborhood of a data point $\mathbf{x}$ is formally defined as follows:

$$N_k(\mathbf{x}) = \{\mathbf{x}' \in X \setminus \mathbf{x} \mid d(\mathbf{x}, \mathbf{x}') \leq k\text{-}distance(\mathbf{x})\} \tag{4}$$

where $d(\mathbf{x}, \mathbf{x}')$ is the distance between the two points. The local neighborhood *density* of a data point $\mathbf{x}$ can be thought of as a measure of the *similarity* of points within that neighborhood to $\mathbf{x}$. Since a kernel function can be used as a measure of the similarity between two data points [17], in Equation 2 other kernel functions can be used in place of the Gaussian kernel. We use the Gaussian kernel and effectively perform kernel density estimation (KDE), also referred to as the Parzen-Rosenblatt Window method [18], [19]. This provides a standard, non-parametric notion of density for the data points.

Our criterion function tries to optimize the ratio of local kernel density estimates for normal and abnormal points (outliers). In the numerator, we sum the local kernel density of all normal data points and in the denominator, we sum the local kernel density of all outliers. By maximizing the ratio of the two, the goal is to find the subset of features that maximizes the density of normal data points and simultaneously minimizes the density of outliers. Intuitively, we would like to find a lower dimensional subspace that corresponds to a subset of the features wherein normal data points are in closely compacted regions of the space while outliers are dispersed, allowing them to be more easily distinguished as anomalous with respect to the normal data. By using a *local* density approach, our criterion can aid outlier detection algorithms in detecting local, as well as global, outliers [15]. In particular, we are already thinking in terms of local neighborhoods, as reflected in the KDE calculations. This notion can be carried over to the outlier detection phase, especially in the case of a local density-based outlier detection algorithm such as LOF which calculates the density of each point within a local neighborhood. This allows the detection

of data points that seem to be *outlying* when considered within the scope of their local neighborhood, not just on a global scale.

To illustrate the advantage of using our proposed criterion function, we show a simple two-dimensional example in Figure 1. Assume we have a dataset with two features and we would like to select the single best feature for outlier detection. In the figure, we use the symbol $\times$ to represent normal points, $+$ for outliers, and we show the projections of the data points onto each of the feature axes. By projecting onto Feature 1, the two classes will have high separability, yet the outliers will have high density which makes it difficult for an outlier detection algorithm to identify them. Projection along Feature 2 gives the normal points high density and the outliers low density, facilitating the detection of outliers. A method that tries to best separate the classes, such as Linear Discriminant Analysis (LDA) which maximizes the trace of the between-class to within-class scatter ratio, would fail to select Feature 2 over Feature 1. Our criterion will correctly select Feature 2 as it maximizes the density ratio of normal points to outliers.

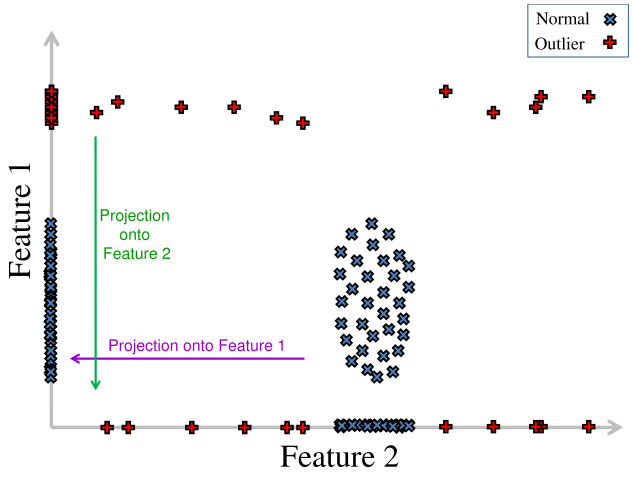


Figure 1. Motivating example of feature selection for outlier detection

B. Forward Search Strategy

Using the described criterion function to evaluate features, the next component of our feature selection algorithm is its search strategy. There are many approaches to searching the space of possible feature subsets, from the naïve exhaustive search to more sophisticated search strategies such as genetic algorithms. In this work, we apply sequential forward selection (SFS), also referred to as sequential forward search [20]. This is a greedy search technique that begins with an initially empty set and adds features one at a time such that the feature added at each round is the one that best improves the criterion function. Note that one can utilize other search strategies, such as backward selection and sequential forward floating search, or applying sparse optimization [6], [21]. For the purposes of this paper, which presents the ability of the proposed feature selection criterion in enhancing outlier detection, we find it sufficient to utilize a simple search strategy that takes feature interaction into account, as in sequential forward search. We name our proposed method Local Kernel Density Ratio (LoKDR) feature selection.

LOF has been shown to perform well in detecting outliers using a ratio of densities. However, it uses a *heuristic* notion of density; the density of a data point is the inverse of the average *reachability* distance between the point and its neighbors, which is the maximum of the actual distance between two points and the k-distance of the latter point [15]. We utilize the success of the main notion in LOF with the goal of producing a cleaner, simpler model that requires no heuristics and is based on KDE, which has a solid statistical foundation. By maximizing the ratio of densities, we emphasize the differences between outliers and normal points, enhancing the ability of outlier detection algorithms to correctly identify outliers.

C. Analysis of Computational Complexity

One of the main components of calculating the criterion function is the k-nearest neighbor (k-NN) search. A simple brute force approach is to calculate the distance between all pairs of points, requiring $\frac{n(n-1)}{2} \times O(d)$ calculations, and then sort the distances to find the k-nearest neighbors of each point using $n \times O(n \log n)$ comparisons, for a total computational complexity of $O(n^2(d+\log n))$. Other k-NN algorithms have been proposed to reduce the computation time, where the main idea is to reduce the number of distances computed [22]. For example, some algorithms partition the data points using an indexing structure, such as a kd-tree, and only compute distances within nearby volumes [23]. This method has been shown to be faster than the brute force approach by up to a factor of 10 [24].

Once the k-NNs of a point is found, the k-nearest neighbor distances are used to calculate its kernel density estimate. These KDE values are then summed up and averaged for the normal points and outliers ($O(n)$), after which the ratio is taken ($O(1)$). This produces the criterion value for one set of features. Assuming feature set F_i has $d_i \leq d$ features, the computational complexity of calculating the criterion function for F_i is: $O(n^2(d_i + \log n)) + O(n) + O(1) = O(n^2(d_i + \log n))$. Therefore, the run-time of the algorithm is bounded by the time it takes to perform the k-nearest neighbor search.

Since there are no dependencies between individual k-NN queries, the k-nearest neighbor search is highly task-parallel, as all queries can be performed in a simultaneous manner, each one independent of any other query. The process is also known to be data-parallel, meaning the data values required to calculate pairwise distances are used for several different calculations. This makes it an excellent candidate

for implementation on a GPU. General purpose computing on a graphics processing unit (GPGPU) has become a popular, cost effective approach for high performance parallel computing. Garcia et al. [24] have shown that implementing the k-nearest neighbor search on a GPU accelerates the search by up to a factor of 400 compared to the brute force CPU-based (serial) implementation.

We build upon this achieved speedup for the k-NN search and implement our entire feature selection algorithm on a GPU. In addition to the parallelism that exists during the k-NN search, we also take advantage of the parallel nature of the forward feature search technique, i.e., each of the candidate feature subsets can be evaluated independently. Thus during each step of the feature search, it possible to calculate the criterion function for all of the possible feature subsets (which is at most d) concurrently on a GPU. This, combined with the parallelism of the k-NN search, enables the algorithm's performance to scale efficiently in terms of both the number of features and the number of data points. It is interesting to note that in an ideal situation where all of the k-NN queries and feature subset evaluations are performed simultaneously, round i of the algorithm would only require $O(d_i + n)$ time, with $O(d_i)$ time for a distance calculation consisting of d_i features and $O(n)$ time for the KDE summation. Since there are at most d rounds of the algorithm (corresponding to the addition of every feature), the best-case computational complexity of a parallel implementation of the LoKDR algorithm is $O(d^2 + nd)$. In practice, the forward search is cut off after a certain constant number of features $c << d$ are selected, yielding a ideal runtime of $O(c(d+n)) = O(d+n)$, making the algorithm linear in the size of the features and sample space.

IV. Experimental Evaluation

In this section, we describe the datasets and outlier detection algorithms used in this study to evaluate the quality of features selected by the LoKDR algorithm. We also briefly describe other feature selection techniques with which we compare our results. We then present the outlier detection results for each of the feature selection methods.

A. Datasets

The datasets used to evaluate our feature selection algorithm are shown in Table I. CNS and LYMPH are microarray gene expression datasets and OVARY and PROST are mass spectrometry datasets provided by Chen et al. [8]. These datasets are examples of real-world problems that consist of a small set of samples and imbalanced class labels. We also evaluate our feature selection algorithm on the ARRHY dataset[1] [25] from the UCI Machine Learning Repository [26], a dataset which has neither highly imbalanced data nor a small sample space. The goal is to distinguish between the presence and absence of cardiac arrhythmia, where class 1 is the normal class and classes 2 to 16 are outliers. Table I provides the number of features, number of samples, and a summary description on each of these datasets.

[1]Preprocessing was done to account for missing values, resulting in the removal of three features and two instances.

Name	Features	Samples	Description
CNS	7129	90	Central Nervous System Embryonal Tumor Data: 60 samples have medulloblastomas and 30 samples have other tumors or no cancer.
LYMPH	7129	77	Lymphoma Data: 58 samples are diffuse large B-cell lymphomas and 19 samples are follicular lymphomas.
OVARY	6000	66	Ovarian Cancer Data: 50 samples are benign tumors and 16 samples are malignant tumors.
PROST	6000	89	Prostate Cancer Data: 63 samples have no evidence of cancer and 26 samples have prostate cancer.
ARRHY	276	450	Cardiac Arrhythmia Data: 244 samples are from class 01 and 206 samples are from classes 02-16.

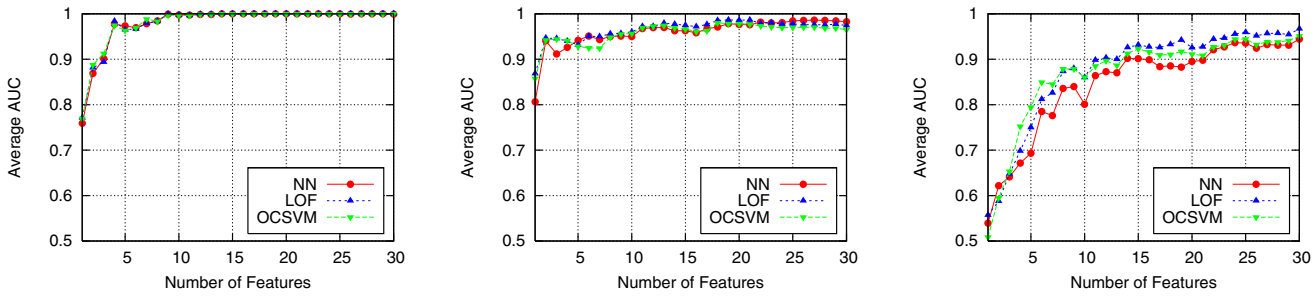
Table I
Overview of datasets

B. Outlier Detection Algorithms

For outlier detection, we use one-class classifiers which are trained on only normal data (inliers). For each data point, the classifier produces a *decision value* that represents its confidence in that point being an outlier. We apply a threshold on the decision value as a cutoff point for decision making. A data point is flagged as an outlier if the decision value exceeds a threshold. Varying the threshold varies the number of correctly classified outliers (true positives) and incorrectly classified normal data (false positives). Using this information, we plot a curve of the true positive rate versus the false positive rate, known as the Receiver Operating Characteristic (ROC) curve [27]. In section IV-D, we perform an evaluation of several feature selection techniques in terms of the area under the ROC curve (AUC) achieved by the outlier detection algorithms on different feature subsets chosen by the feature selection techniques.

The classifiers used to evaluate the feature subsets are Nearest Neighbor (NN), Local Outlier Factor (LOF), and One-Class Support Vector Machines (OCSVM). The (one-class) Nearest Neighbor classifier is a distance-based outlier detection algorithm wherein a data point's decision value is the distance to its nearest neighbor. The greater the distance, the more likely that point is an outlier. The LOF algorithm takes a density-based approach to detect outliers; the greater the density of a point's nearest neighbors compared to its own density, the more outlying the data point. The decision value assigned to a data point is its local outlier factor.

The OCSVM classifier [28] uses a kernel function to map



(a) LoKDR (b) BAHSIC (c) LASSO

Figure 2. Average AUC results for the NN, LOF, and OCSVM classifiers on the CNS dataset

the data into a feature space H with the goal of capturing most of the data vectors within a "small" region. It then tries to separate the mapped vectors from the origin with a hyperplane that has the maximum margin. The origin and data points "close enough" to it are assumed to be outliers. The decision value of a data point is calculated as: $f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i K(\mathbf{x}_i, \mathbf{x}) - \rho$ where α_i are the support vector coefficients, K is a kernel function, and ρ represents the margin. The parameter ρ is effectively the threshold that determines whether points are "close enough" to the origin to be considered outliers. For our OCSVM experiments, we use LIBSVM (version 3.11) [29] with the standard parameters for one-class SVMs.

C. Feature Selection Algorithms

To compare our results with other popular filter-based feature selection algorithms, we evaluate the features selected by RELevance In Estimating Features (RELIEF), Feature Assessment by Sliding Thresholds (FAST), Least Absolute Shrinkage and Selection Operator (LASSO), and BAckward elimination with the Hilbert Schmidt Independence Criterion (BAHSIC). RELIEF [1] is a filter feature selection method that evaluates individual features based on how well they differentiate between neighboring instances from different classes versus from the same class. We use the Weka toolbox [30] to select features with RELIEF. FAST [8] is a feature selection algorithm for small sample and imbalanced data classification problems. The main idea is to rank features based on the area under the ROC curve generated by each feature.

LASSO [6] solves the linear regression problem formulation with an added constraint on the sum of the regression coefficients. This drives the coefficients of less relevant features towards zero. We use the logistic regression variant provided in the GLMNet [31] Matlab toolbox and rank features based on the absolute value of the coefficients. BAHSIC [7] uses the backward elimination search strategy on features evaluated using the Hilbert Schmidt Independence Criterion (HSIC). We use the Python code provided by Song et al. [7] to perform feature selection with BAHSIC.

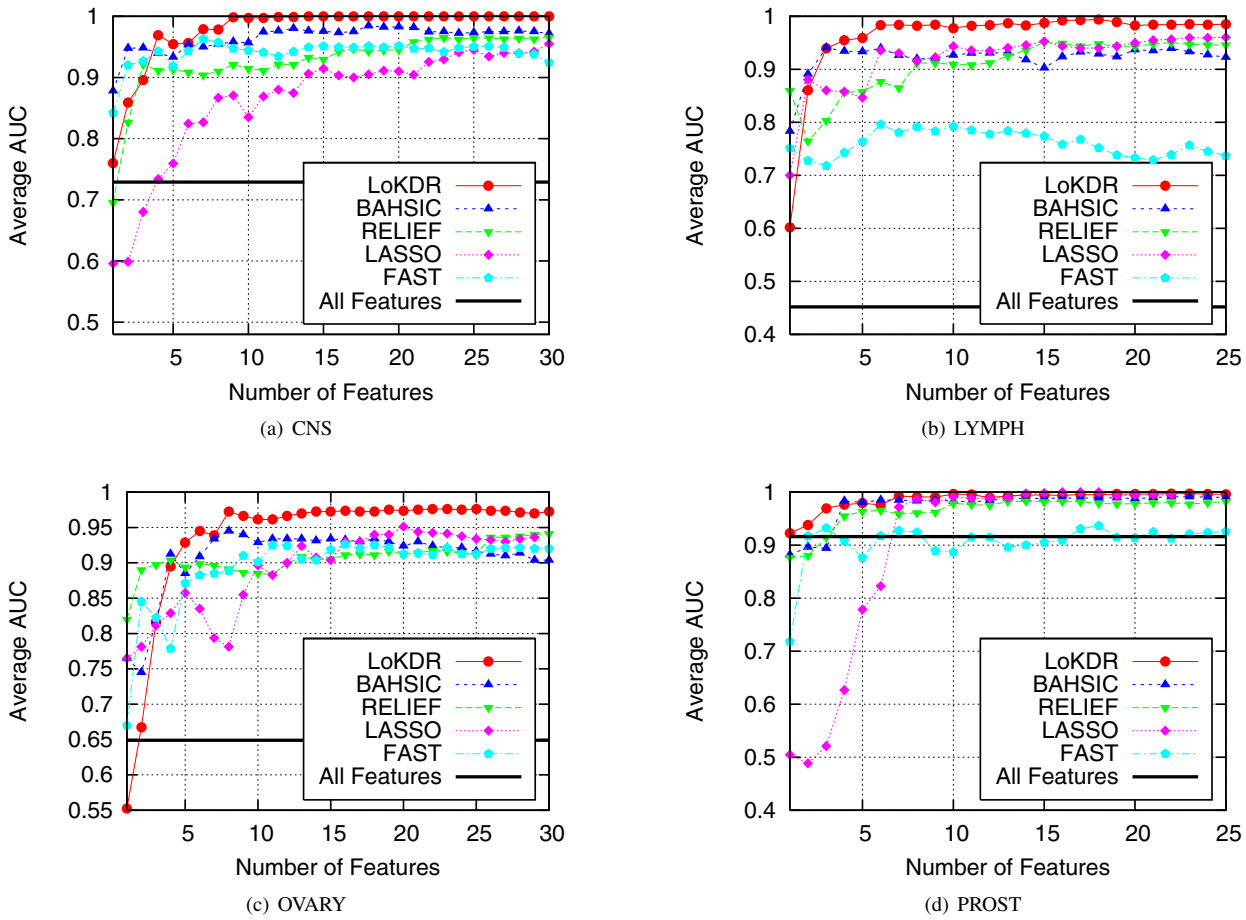
D. Results

For the training phase of the outlier detection algorithms, we take a one-class (or semi-supervised) learning approach and train only on normal data points. During testing, both normal and outlier data points are used to see how well the model is able to detect outliers. We perform 10-fold cross validation by dividing the normal data points into ten folds and training on nine of them while testing on the tenth. Since no outliers are used during the training phase, we use all outliers during the testing phases of the cross validation. In what follows, we present two sets of experiments: in the first, we evaluate the quality of the solution achieved by our novel feature selection algorithm compared to other state-of-the-art methods. In the second set of experiments, we quantify the speedup obtained with a parallel GPU-based implementation of the LoKDR feature selection algorithm, compared to a serial CPU-based implementation.

1) Quality Experiments: We evaluate the quality of the outlier detection results using the area under the ROC curve (AUC). The ROC curve is a plot of the true positive rate (fraction of outliers correctly detected) versus the false positive rate (fraction of normal points misclassified as outliers). It represents the behavior of a classifier across a range of thresholds on the decision values.

For the LoKDR feature selection algorithm, there are two main parameters that can be tuned: k which determines the size of the local neighborhood and σ, the Gaussian kernel width. By varying the value of k in $[1, n-1]$ and σ in [1, 5], we observed that most of the results were not drastically sensitive to the choice of these parameters, though some values produced slightly better results than others. The benefits of this are two-fold; first, it shows the stability of the criterion function, as it is not extremely sensitive to these parameters. Second, with smaller values of k, we can achieve similar (if not better) results than larger values, thereby reducing the computational cost. For our experiments, we set the values of the parameters based on 10-fold cross validation on the training set.

The results of our experiments using NN, LOF, and



(a) CNS (b) LYMPH

(c) OVARY (d) PROST

Figure 3. Average AUC results for the feature selection algorithms on the microarray and mass spectrometry datasets

OCSVM show that in general, the classifiers perform comparably across the various feature selection algorithms and datasets with no clear winner. As an example, in Figure 2 we present the average area under the curve (AUC) results as a function of the number of selected features using 10-fold cross validation for all three outlier detection algorithms on the CNS dataset with features selected by LoKDR, BAHSIC, and LASSO. On the x-axis, we vary the number of selected features and on the y-axis, we plot its corresponding average AUC.

As the goal is not to compare the outlier detection algorithms themselves, but rather to compare the proposed feature selection algorithm (LoKDR) with previous feature selection methods, for the remaining figures of this section we shall present the outlier detection results using the LOF classifier. The other classifiers produce similar results (cf. [32]). In Figure 3, we show the AUC results of the feature selection algorithms on the microarray and mass spectrometry datasets. With a horizontal line, we show the AUC obtained when using all of the features. This displays the importance of performing feature selection, as all of the feature selection algorithms are able to surpass the AUC achieved with the entire feature set. The figure also highlights the strength of the LoKDR algorithm in selecting features for outlier detection. Across the datasets, the features chosen by LoKDR enable the outlier detection algorithm to identify outliers with a high detection rate and few false positives, as reflected in the high average AUC. For the CNS, LYMPH, and OVARY datasets, as the number of selected features increases, the average AUC for LoKDR rapidly exceeds that of the other methods. For the PROST dataset, the performance of LoKDR starts out the strongest and continues to be competitive with the other methods.

To see how well our feature selection algorithm performs on a more general dataset that does not have imbalanced data or a small sample size, we also ran experiments on the ARRHY dataset and present results in Figure 4. The results show that while features selected by BAHSIC produce the highest average AUC for most of the feature subsets, the LoKDR algorithm still performs well and produces AUC

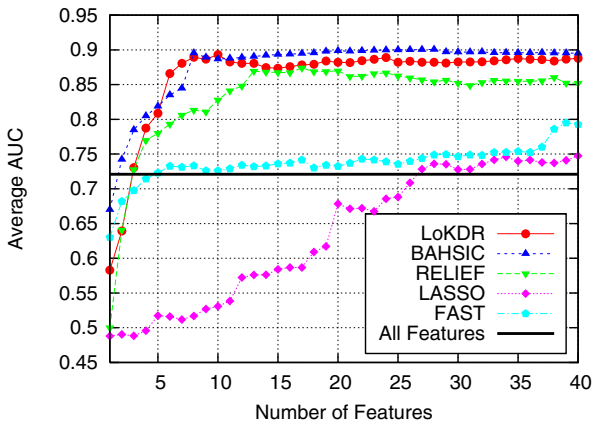


Figure 4. Average AUC results on the ARRHY dataset

values that are comparable with those of BAHSIC.

Using the paired Student's t-test, we confirmed that our experimental results showing the superiority of LoKDR over the other feature selection methods are statistically significant at the 95% confidence level with respect to all methods, except BAHSIC on the PROST and ARRHY datasets, where they perform comparably. From our results, we conclude that the LoKDR feature selection algorithm chooses features that enable outlier detection algorithms to do consistently well across all the datasets, from those which are very high-dimensional with imbalanced class labels and that suffer from the small sample space problem, to a more general dataset without these properties.

2) Performance Experiments: We also evaluate the performance of a GPU-based implementation of our proposed feature selection algorithm as compared to a serial (CPU only) implementation. The GPU version was written using NVIDIA's Compute Unified Device Architecture (CUDA) and the serial version was written in C. CUDA extends the C language with new programming constructs and provides libraries and a platform for the efficient execution of general-purpose applications on GPUs. Our serial experiments were run on an Intel Xeon CPU E5405 running at 2.00 GHz. The GPU used in our experiments is the NVIDIA Tesla M2070 which is based on the NVIDIA Fermi GPU architecture. Tesla M2070 modules are performance-optimized, high-end products which offer 6 GB of GDDR5 ECC-protected memories on board with a 1.566 GHz memory clock and a 384-bit memory interface. The features supported by a CUDA hardware are described by the *Compute Capability*, and Fermi architectures have the support for CUDA Compute Capability 2.0.

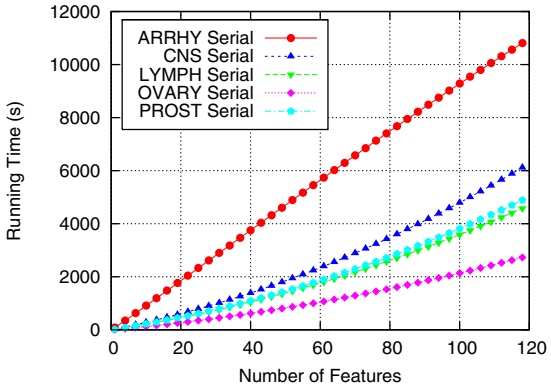
In Figure 5, we present the performance results. Figure 5(a) shows the running time (in seconds) of the serial implementation of LoKDR on all the datasets with increasing number of selected features. From this figure, we see that the ARRHY dataset requires a longer running time than the microarray and mass spectrometry datasets. Although it has far fewer features than the other datasets (276 versus 6000 and 7129), its larger sample size increases its runtime more dramatically than its features. This validates our analysis that the number of data points has a greater impact on the computational complexity of the serial implementation than does the number of features. Another confirmation of this is found in the running time of LYMPH versus PROST. While the number of features in LYMPH is greater than PROST, the number of data points in PROST is greater than LYMPH which leads to its slightly longer running time.

As we add more features to each dataset, the ARRHY dataset shows a linear increase in the running time whereas CNS, LYMPH, OVARY, and PROST show a somewhat superlinear increase. Contrast this to Figure 5(b), which presents the running time of the GPU-based implementation of LoKDR and where ARRHY displays a somewhat sublinear increase and the results of the other datasets are more linear. Also in the GPU implementation, the PROST dataset shows a slightly lower running time compared to the LYMPH dataset. This can be attributed to the improvement gained from performing k-nearest neighbor queries for several different data points concurrently on a GPU, which reduces the running time required for a larger sample size.

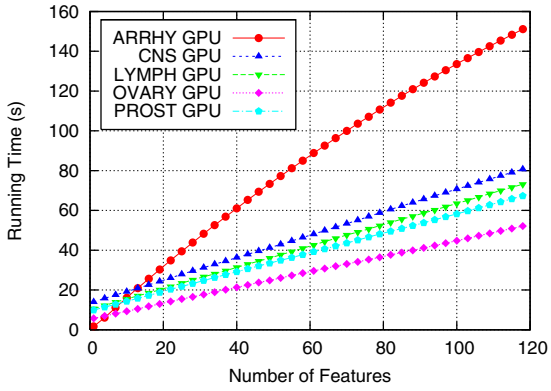
An important characteristic of the GPU performance plot is the drastic improvement in the running time for all of the datasets, compared to the serial version. To see this more clearly, in Figure 5(c), we plot the running time of both the serial and GPU implementations on the ARRHY dataset. The time it takes to select 40 features goes from over an hour in the serial vesion to about a minute on the GPU. This results in a computation time savings of over 98%; in other words, the GPU-based implementation is over 61.5$\times$ faster than its serial counterpart. Figure 5(d) presents this improvement in running time, or *speedup*, for all of the datasets with increasing number of selected features. As can be seen in the figure, the speedup grows progressively as more features are added, exceeding 50$\times$ improvement on all datasets and only going higher from there.

The ARRHY dataset shows the most significant improvement for even a small number of features due to the performance gain of concurrent data point processing on the GPU. As more features are selected, the cost of performing distance calculations (and hence k-NN queries) increases, so there is more performance to be gained when they are done concurrently. This, coupled with the small sample space of the microarray and mass spectrometry datasets, means that more data points can be processed and used to calculate the optimization criterion within a shorter amount of time, which is reflected in the large rise in speedup for the CNS, LYMPH, OVARY, and PROST datasets.

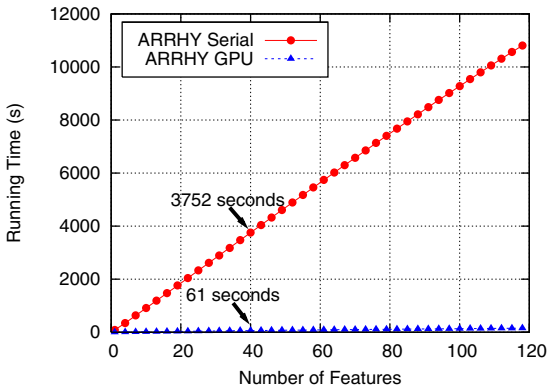
Another observation from the speedup figure is the similarity in speedup of the CNS and PROST datasets. These datasets have comparable sample space sizes and differ in



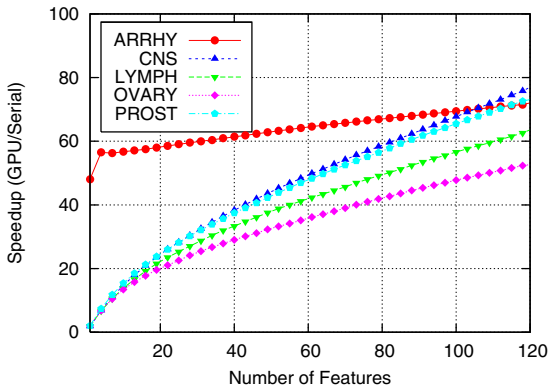
(a) Running time of serial implementation on all datasets



(b) Running time of GPU implementation on all datasets



(c) Comparison of serial and GPU implementations on ARRHY dataset



(d) Speedup of GPU implementation over serial implementation

Figure 5. Evaluation of serial and GPU implementations of LoKDR feature selection algorithm

their feature set sizes, yet the common pattern of their speedup confirms the ability to benefit from parallel processing of data points. As more features are added, their speedups slowly diverge because PROST has fewer features and thus requires evaluating fewer candidate subsets during each subsequent round of the feature selection algorithm.

From these results, it is clearly evident that exploiting the parallelism that exists in the LoKDR algorithm can lead to great rewards in terms of computation time. Data mining problems where adding more features can enhance the quality of the final solution further add to the appeal of a GPU-based implementation, with its ability to achieve vast speedups and performance improvements. The implementation of our proposed feature selection algorithm for outlier detection on a GPU gains from its ability to perform processing on data points and feature subsets concurrently, allowing the performance to scale nicely with respect to both.

V. Conclusions and Future Work

In this work, we presented a novel feature selection criterion catered to outlier detection problems. The proposed method is non-parametric and makes no assumptions about the distribution of the underlying data other than the fact that outliers are *different* than the normal data. It selects features that best capture the behavior of normal data while making outliers more distinct from the normal. We applied a forward search strategy to our feature selection criterion and compared its ability to detect outliers with other popular feature selection methods. Experiments on real datasets showed that our local kernel density ratio (LoKDR) feature selection algorithm does very well to discern features that facilitate the detection of outliers. By taking advantage of its parallel nature in terms of the number of data points and features, we also achieved great speedups with an implementation on a graphics processing unit (GPU).

In future work, we will incorporate other search techniques using our novel feature selection criterion. In particular, the backward search method can reap massive rewards in terms of running time savings since it begins with all the features and removes them one by one. This necessitates a great number of computations for distance calculations and a GPU implementation can perform them concurrently, yielding significant speedups.

REFERENCES

[1] K. Kira and L. A. Rendell, "A practical approach to feature selection," in *International Conference on Machine Learning (ICML)*, 1992, pp. 249–256.

[2] H. Liu and H. Motoda, *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, 1998.

[3] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[4] M. Dash and H. Liu, "Feature selection for classification," *Intelligent Data Analysis*, vol. 1, pp. 131–156, 1997.

[5] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[6] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society, Series B*, vol. 58, pp. 267–288, 1994.

[7] L. Song, J. Bedo, K. M. Borgwardt, A. Gretton, and A. Smola, "Gene selection via the BAHSIC family of algorithms," *Bioinformatics*, vol. 23, pp. 490–498, 2007.

[8] X.-W. Chen and M. Wasikowski, "FAST: A ROC-based feature selection metric for small samples and imbalanced data classification problems," in *KDD*, 2008, pp. 124–132.

[9] C. Aggarwal and S. Yu, "An effective and efficient algorithm for high-dimensional outlier detection," *The VLDB Journal*, vol. 14, pp. 211–221, 2005.

[10] H. V. Nguyen and V. Gopalkrishnan, "Feature extraction for outlier detection in high-dimensional spaces," *Journal of Machine Learning Research*, vol. 10, pp. 66–75, 2010.

[11] S. Hido, Y. Tsuboi, H. Kashima, M. Sugiyama, and T. Kanamori, "Statistical outlier detection using direct density ratio estimation," *Knowledge and Information Systems*, vol. 26, no. 2, pp. 309–336, 2011.

[12] M. Sugiyama, M. Yamada, P. von Bünau, T. Suzuki, T. Kanamori, and M. Kawanabe, "Direct density-ratio estimation with dimensionality reduction via least-squares hetero-distributional subspace search," *Neural Networks*, vol. 24, no. 2, pp. 183–198, 2011.

[13] A. Smola, L. Song, and C. H. Teo, "Relative Novelty Detection," in *Artificial Intelligence and Statistics (AISTATS), JMLR W&CP 5*, 2009.

[14] D. M. Hawkins, *Identification of outliers*. Chapman and Hall, London; New York, 1980.

[15] M. M. Breunig, H. P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," *ACM SIGMOD Record*, vol. 29, no. 2, pp. 93–104, 2000.

[16] R. A. Horn and C. R. Johnson, *Matrix Analysis*. Cambridge University Press, Cambridge; New York, 1985.

[17] M.-F. Balcan and A. Blum, "On a theory of learning with similarity functions," in *International Conference on Machine Learning (ICML)*, 2006, pp. 73–80.

[18] E. Parzen, "On Estimation of a Probability Density Function and Mode," *The Annals of Mathematical Statistics*, vol. 33, no. 3, pp. 1065–1076, 1962.

[19] M. Rosenblatt, "Remarks on Some Nonparametric Estimates of a Density Function," *The Annals of Mathematical Statistics*, vol. 27, no. 3, pp. 832–837, 1956.

[20] P. A. Devijver and J. Kittler, *Pattern recognition: A statistical approach*. Prentice Hall, 1982.

[21] M. Masaeli, G. Fung, and J. G. Dy, "From transformation-based dimensionality reduction to feature selection," in *International Conference on Machine Learning (ICML)*, 2010, pp. 751–758.

[22] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe LSH: efficient indexing for high-dimensional similarity search," in *Proceedings of the 33rd International Conference on Very Large Data Bases*, ser. VLDB '07, 2007, pp. 950–961.

[23] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," *J. ACM*, vol. 45, no. 6, pp. 891–923, Nov. 1998.

[24] V. Garcia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using gpu," in *CVPR Workshop on Computer Vision on GPU*, Anchorage, Alaska, USA, June 2008.

[25] H. A. Güvenir, B. Acar, G. Demiröz, and A. Çekin, "A supervised machine learning algorithm for arrhythmia analysis," in *Computers in Cardiology Conference*, 1998, pp. 433–436.

[26] A. Frank and A. Asuncion, "UCI machine learning repository," 2010. [Online]. Available: http://archive.ics.uci.edu/ml

[27] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006.

[28] L. M. Manevitz, M. Yousef, N. Cristianini, J. Shawe-Taylor, and B. Williamson, "One-class svms for document classification," *Journal of Machine Learning Research*, vol. 2, pp. 139–154, 2001.

[29] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 1–27, 2011.

[30] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *SIGKDD Explorations Newsletter*, vol. 11, pp. 10–18, 2009.

[31] J. H. Friedman, T. Hastie, and R. Tibshirani, "Regularization paths for generalized linear models via coordinate descent," *Journal of Statistical Software*, vol. 33, no. 1, pp. 1–22, 2010.

[32] F. Azmandian, "Learning at the Virtualization Layer: Intrusion Detection and Workload Characterization from within the Virtual Machine Monitor," Ph.D. dissertation, Northeastern University, August 2012.