Fast Factorization-Free Kernel Learning for Unlabeled Chunk Data Streams

Yi Wang\textsuperscript{1,2}, Nan Xue\textsuperscript{1,2}, Xin Fan\textsuperscript{1,2*}, Jiebo Luo\textsuperscript{3}, Risheng Liu\textsuperscript{1,2}, Bin Chen\textsuperscript{1,2}, Haojie Li\textsuperscript{1,2} and Zhongxuan Luo\textsuperscript{1,2}

\textsuperscript{1} DUT-RU International School of Information Science and Engineering, Dalian University of Technology, P. R. China
\textsuperscript{2} Key Laboratory for Ubiquitous Network and Service Software of Liaoning Province, P. R. China
\textsuperscript{3} Department of Computer Science, University of Rochester, USA
{dlutwangyi, xin.fan, rsliu, hjli, zxluo}@dlut.edu.cn, xnfiona@gmail.com, jluo@cs.rochester.edu

Abstract

Data stream analysis aims at extracting discriminative information for classification from continuously incoming samples. It is extremely challenging to detect novel data while updating the model in an efficient and stable fashion, especially for the chunk data. This paper proposes a fast factorization-free kernel learning method to unify novelty detection and incremental learning for unlabeled chunk data streams in one framework. The proposed method constructs a joint reproducing kernel Hilbert space from known class centers by solving a linear system in kernel space. Naturally, unlabeled data can be detected and classified among multi-classes by a single decision model. And projecting samples into the discriminative feature space turns out to be the product of two small-sized kernel matrices without needing such time-consuming factorization like QR-decomposition or singular value decomposition. Moreover, the insertion of a novel class can be treated as the addition of a new orthogonal basis to the existing feature space, resulting in fast and stable updating schemes. Both theoretical analysis and experimental validation on real-world datasets demonstrate that the proposed methods learn chunk data streams with significantly lower computational costs and comparable or superior accuracy than the state of the art.

1 Introduction

One of the challenging aspects of data analytics nowadays is dealing with streaming and fast-moving input data, e.g., digital monitor images, social media feedbacks, marketing and financial data. In real applications, these data might come from several terminals without labels, simultaneously, forming chunk data streams. Particularly, a chunk of data may contain some new examples belong to the seen/known classes and/or unseen/novel classes. Therefore, we have to detect novelties first, and then incorporate all new information into the existing model. Although, novelty detection and incremental learning for data streams have received increasing attention in the past few years [Wang et al., 2016; Faria et al., 2016; Lu et al., 2017], efficient and stable solutions to simultaneously detect novelties and learn chunk data streams are still rare.

Recently, kernel null-space based discriminant analysis (KNDA) [Bodesheim et al., 2013] and incremental KNDA (IKNDA) [Liu et al., 2017] have been reported that their performances outperform classifiers SVDD [Tax and Duin, 2004] and GP-Mean/GP-Var [Kemmler et al., 2010], and deep approaches (e.g., Alexnet) [Krizhevsky et al., 2012] in multi-classification and novelty detection. KNDA learns a decision model by removing the intra-class variances in a reproducing kernel Hilbert null (RKHN) space, which provides elegant discrimination for highly complex and non-linear distribution data. Unfortunately, the eigen-decomposition of a large kernel matrix makes KNDA hard to scale up. IKNDA provides an updating scheme to renew the null space based on singular value decomposition (SVD). However, its incremental updating mechanism is not efficient for on-line learning tasks on large datasets. Besides, kernel methods also suffer from high memory cost of kernel matrix on large and/or high-dimensional datasets.

In this work, we put forward an efficient kernel learning method to unify the novelty detection and incremental learning for unlabeled chunk data streams in one framework. Specifically, our contributions are twofold:

- We propose an efficient factorization-free batch kernel discriminant analysis (FKDA). FKDA constructs a \( k \)-dimensional RKH space from \( k \) known class centers by solving a linear system [Chu et al., 2015] in kernel space. The mapping (projection) to the feature space turns out to be the production of two small kernel matrices, thus the computational complexity of FKDA is significantly lower than other kernel DA methods, especially for large-scale scenarios. Moreover, each class is represented by a single point with a deterministic novelty threshold in RKH space, that also caters for multi-class novelty detection.
We develop a fast and stable incremental scheme for FKDA, called IFKDA. In IFKDA, inserting a novel class to the learned model acts by appending a new orthogonal basis to the original feature space without disturbing the distribution of known classes, which makes the evolution more robust and stable. Moreover, IFKDA is very efficient in on-line learning scenarios, requiring only a small portion of the key kernel matrices to be recomputed.

2 Overview of Discriminative Analysis (DA)

In this section, we give a short overview of linear discriminative analysis and kernel discriminative analysis.

Given a data matrix with \( k \) clusters, \( X = \{X_1, X_2, \ldots, X_k\} \in \mathbb{R}^{d \times n} \), where \( X_i \in \mathbb{R}^{d \times n_i} \) and \( n = \sum_{i=1}^{k} n_i \). The widely known Foley-Sammon transform [Chen et al., 2000] estimates a low-dimensional subspace to discriminative features by maximizing the following criterion:

\[
G^* := \arg \max_G \text{trace}((G^T S_G)^{-1}(G^T S_i G)), \quad (1)
\]

where \( G \in \mathbb{R}^{d \times k} \) is a transformation matrix which projects data in \( d \)-dimensional space to \( k \)-dimensional subspace, \( S_G \) is between-class scatter matrix, and \( S_i \) is total scatter matrix (\( S_i = S_t + S_w \), where \( S_w \) is within-class scatter matrix).

A common issue of discriminative learning is the small-size-sample (SSS) problem that \( n \) is far less than \( d \), such that \( S_w \) is singular in general. One popular extension replaces the inverse in the classical model by pseudo-inverse, resulting in the following optimization model [Huang et al., 2002],

\[
G^* := \arg \max_G \text{trace}((G^T S_G)^{-1}(G^T S_i G)), \quad (2)
\]

where \( (\cdot)^{\dagger} \) denotes the pseudo-inverse of \( (\cdot) \). And any transfer matrix \( G \) with the form

\[
\text{trace}((G^T S_G)^{-1}(G^T S_i G)) = k - 1
\]

should be an optimal solution to (2) [Chu et al., 2015].

In [Chu et al., 2015], Chu et al. proposed a more efficient way for SSS problem by solving a linear system in (4) via the economic QR-factorization of \( X \).

\[
X^T G = E, \quad (4)
\]

where \( G \in \mathbb{R}^{d \times k} \) and \( E \) is a \( n \)-by-\( k \) matrix and the \( i \)-th column is \((0, \ldots, e_i, \ldots, 0)^T\) where \( e_i = [1 \ldots 1]^T \in \mathbb{R}^{n_i \times 1} \).

Assuming that training samples are linearly independent, the solution of (4) with the minimal 2-norm is

\[
G = QR^{-T} E. \quad (5)
\]

If considering kernel tricks for DA, we should further map \( X \) into a feature space \( H \) with an implicit mapping function \( \Phi : x_j \rightarrow \phi(x_j) \), where \( \phi(x_j) \in H \) and \( x_j \) denote the \( i \)-th item of the \( i \)-th class. And we have

\[
X^\Phi = \Phi(X) = [\phi(x_1), \ldots, \phi(x_n)]. \quad (6)
\]

Accordingly, the criterion in (2) extends to

\[
G^* := \arg \max_{G^\Phi} \text{trace}(((G^\Phi)^T S_t^\Phi G^\Phi)^{-1}(G^\Phi)^T S_i^\Phi G^\Phi). \quad (7)
\]

And any \( G^\Phi \) with the form

\[
\text{trace}(((G^\Phi)^T S_t^\Phi G^\Phi)^{-1}(G^\Phi)^T S_i^\Phi G^\Phi) = k - 1
\]

is an optimal solution of the optimization problem (7).

3 Factorization-free KDA (FKDA)

In this section, a fast factorization-free batch kernel DA method (FKDA) is developed. We first induce the linear system in (4) into the kernel space.

**Theorem 1:** Assuming \( X^\Phi \) are linearly independent, and the linear system in (9) is solvable, then the solution to the optimization problem (7) is given by \( G^\Phi \).

\[
(X^\Phi)^T G^\Phi = E. \quad (9)
\]

Proof: See the Appendix.

Considering the advantages of using class centers as input in storage and computation as in AKDA/QR [Xiong et al., 2005], we introduce them into (9) and have

\[
(C^\Phi)^T G^\Phi = M^T (X^\Phi)^T G^\Phi = M^T E, \quad (10)
\]

where \( C^\Phi = X^\Phi M \) is a global centroid matrix in the kernel space, and \( M \) is a \( n \)-by-\( k \) matrix with the \( i \)-th column is \((0, \ldots, e_i/n_i, \ldots, 0)^T \). Since

\[
M^T E = \begin{bmatrix}
\frac{1}{n_1} e_1^T \\
\vdots \\
\frac{1}{n_k} e_k^T \\
\end{bmatrix} \begin{bmatrix}
e_1 \\
\vdots \\
e_k \\
\end{bmatrix} = I,
\]

where \( I \in \mathbb{R}^{k \times k} \) is an identity matrix, then

\[
(C^\Phi)^T G^\Phi = I. \quad (12)
\]

A key observation for most widely used Gaussian kernel function: \( \exp(-\|x-y\|^2/\sigma) \), is that for relatively large \( \sigma \), the center of each class in the original space will be projected very close to the center of each class in the kernel space [Xiong et al., 2005]. So we adopt this idea to compute the centroid matrix \( C_o \) in the original space first, then we approximate the kernel centroid matrix \( C^\Phi \) by

\[
C^\Phi \approx [\phi(c_1), \ldots, \phi(c_n)](\equiv C_o^\Phi), \quad (13)
\]

where \( c_i \) denotes the center vector of class \( X_i \) in the original space. Therefore, we have

\[
(C_o^\Phi)^T G^\Phi = I. \quad (14)
\]

Assuming that class centers are linearly independent, we can resort to QR-decomposition of \( C_o^\Phi = Q_o^\Phi R_o^\Phi \) to solve \( G^\Phi \), where \( Q_o^\Phi \) is the column orthogonal and \( R_o^\Phi \) is nonsingular. Then we have

\[
G^\Phi = Q_o^\Phi (R_o^\Phi)^{-T}. \quad (15)
\]

Denote \( K_o = (C_o^\Phi)^T C_o^\Phi \in \mathbb{R}^{n \times n} \). Since \( (R_o^\Phi)^T R_o^\Phi = (Q_o^\Phi)^T Q_o^\Phi R_o^\Phi = (C_o^\Phi)^T C_o^\Phi \), thus

\[
G^\Phi = Q_o^\Phi R_o^\Phi (R_o^\Phi)^{-1}(R_o^\Phi)^{-T} = C_o^\Phi K_o^{-1}. \quad (16)
\]

**Result:** Given a sample \( z \), we can get its projection in the \( k \)-dimensional RKH space by \( (G^\Phi)^T \phi(z) \) in (17).

\[
(G^\Phi)^T \phi(z) = (C_o^\Phi K_o^{-1})^T \phi(z) = K_o^{-1} K_o^{-1} \phi(z), \quad (17)
\]
Algorithm 1 FKDA-batch method

Require: Data matrix \(X\), nonlinear mapping \(\Phi\) and a testing sample \(z\).
Ensure: The projected feature of \(z\).

Stage1:
1: Compute the centroid matrix: \(C_o\).
2: Construct the kernel matrix: \(K_o = (C_o^\Phi)^T C_o^\Phi\).
3: Construct the kernel vector: \(K_{cz} = <\phi(C_o^\Phi), \phi(z)>\).
4: Compute \((G^\Phi)^T z^\Phi = K_o^{-1} K_{cz}\).

Stage2:
5: Compute the centroid matrix: \(C_{cz}\).
6: Construct the kernel matrix: \(K_{cz} = (C_{cz}^\Phi)^T C_{cz}^\Phi\).
7: Construct the kernel vector: \(K_{cz} = <\phi(C_{cz}^\Phi), \phi(z)>\).
8: Compute \((G^\Phi)^T z^\Phi = K_{cz}^{-1} K_{cz}\).

Table 1: Space and time complexities for batch DAs (\(n\) is the training sample number, \(d\) is the feature dimension, and \(k\) is the class number).

<table>
<thead>
<tr>
<th>Method</th>
<th>Space</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>AKDA/QR</td>
<td>(O(dk + nk + k^2))</td>
<td>(O(dnk + dn + dk^2))</td>
</tr>
<tr>
<td>SRKDA</td>
<td>(O(n^2 + nk + k^2))</td>
<td>(O(dn^2 + n^3 + nk^2))</td>
</tr>
<tr>
<td>KNDA</td>
<td>(O(n^2 + nk))</td>
<td>(O(dn^2 + n^3 + nk^2))</td>
</tr>
<tr>
<td>LDA/QR</td>
<td>(O(n^2 + nk + n))</td>
<td>(O(dn^2 + dk))</td>
</tr>
<tr>
<td>KPE</td>
<td>(O(n^2))</td>
<td>(O(dn^2 + d))</td>
</tr>
<tr>
<td>FKDA</td>
<td>(O(dk + k^2))</td>
<td>(O(dn + dk^2))</td>
</tr>
</tbody>
</table>

where \(K_{cz} \in \mathbb{R}^{k \times 1}\) and \(K_{cz}(i) = <\phi(c_i), \phi(z)>\).

Obviously, there is no need to really compute QR-decomposition of \(C_o^\Phi\).

FKDA is summarized as Algorithm 1.

Next, we compare the space and time complexities of FKDA with AKDA/QR [Xiong et al., 2005], SRKDA [Cai et al., 2007], KNDA [Bodesheim et al., 2013], KPE [Min et al., 2016], and LDA/QR [Chu et al., 2015] in Table 1. Since FKDA gives an implicit transform matrix \(G^\Phi\), we add the projecting step to other methods for fairness. We also compare with a linear DA method (i.e., LDA/QR), so the space and time complexities of constructing the kernel matrix are added for all kernel methods in comparisons.

AKDA/QR and FKDA both use centroid matrix as input, so they need less space and time cost in constructing kernel matrix than other kernel DAs. The cost of constructing centroid matrix is as low as \(O(dn)\) in our implementation. Although, FKDA has to compute the inverse of \(K_o \in \mathbb{R}^{k \times k}\) with the complexity of \(O(k^3)\), other kernel DAs also have such computation, e.g., AKDA/QR needs four times matrix inversion with the same scale, though it employs QR-decomposition, and KNDA has to compute the inversion of a matrix with the complexity of \(O(n^3)\). In addition, the total computation cost of FKDA, including constructing kernel matrix, is no more than LDA/QR, especially when \(k\) is far less than \(n\). Besides, LDA/QR has a demand of \((d >> n)\) of \(Q\) matrix of \(X\). In contrast, FKDA needs no factorization and can perform well on both \((d >> n \gg k)\) and \((n >> d > k)\) datasets, that will be demonstrated in Section 5.

4 On-line Learning for Chunk Data Streams

The on-line phase, shown in Figure 1, is composed of three operations: detect novelties, classify new samples and evolute the decision model. We will detail them in this section.

Figure 1: The framework of on-line learning for unlabeled chunk data streams. FKDA learns a \(k\)-RKH space from \(k\) known classes centers, and the projection of each known class center is fixed in the unit position on each axis, e.g., \(C_1\) of \(X_1\) (Dog) and \(C_2\) of \(X_2\) (Rabbit). New or/and novel examples can be detected and discriminated in one decision model with a deterministic boundary threshold. The model evolves with the centralized new data. Particularly, inserting novel classes, e.g., \(X_3\) (Eagle) and \(X_4\) (Polar Bear), equals to append new basics, e.g., \(D_3\) and \(D_4\), perpendicular to the original space.

4.1 Novelty Detection and Classification

Given a chunk of unlabeled data, we assume they are clustered without noises or outliers by clustering algorithms, e.g., CluStream [Aggarwal et al., 2003], for clustering is not the focus of this work. Then we have a set of unlabeled cluster centers as input. And novelty detection and classification are performed in the learned \(k\)-RKH space.

To be noted, from the derivation of (14), we have

\[
((C_o^\Phi)^T G^\Phi)^T = (G^\Phi)^T (C_o^\Phi) = I. \tag{18}
\]

This means \(k\) class centers are projected to the unit positions of the \(k\) orthogonal axes, respectively, for the value of each volume of \(I\) represents the coordinate of the mapping points of known class centers in \(k\)-RKH space. For instance, given two known classes as shown in Figure 1, we have a 2-RKH space, where \(c_1\) of \(X_1\) (Dog) is projected to \(C_1(1, 0)\) and \(c_2\) of \(X_2\) (Rabbit) is projected to \(C_2(0, 1)\).

Therefore, we can build a hypersphere surrounding each class center as classification boundary. New instances will be classified with respect to their distances to the previously known classes centers. The instances that being projected out of boundaries of all the known classes are classified as novel classes. Furthermore, we set the same value for the thresholds of boundaries of known classes, i.e., the half value of the Euclidean distance between two arbitrary classes. Since the positions of class centers are fixed on the axes, this value is also valid for the new inserted classes, that will be explained in the next subsection.

4.2 Updating Schemes (IFKDA)

The updating task can be easily accomplished by recomputing or appending some parts of the kernel matrices \(K_o\) and \(K_{cz}\).
We suppose that a chunk of data is composed of some new samples belong to \( k_a \) existing classes and some samples belong to \( k_b \) new classes. Then the new kernel matrix \( \tilde{K}_o \in \mathbb{R}^{(k+k_o) \times (k+k_o)} \) can be updated by (19)

\[
\tilde{K}_o = \begin{bmatrix}
K'_o & K^b_o \\
K^b_oT & K^c_o
\end{bmatrix},
\]

where \( K'_o \in \mathbb{R}^{k \times k} \) is the updated original kernel matrix \( K_o \), by recomputing the items relating to those known classes which have new samples, and \( K^b_o \in \mathbb{R}^{k_b \times k} \) and \( K^c_o \in \mathbb{R}^{k_b \times k} \) are the new items introduced by novel classes in (20) and (21), respectively.

\[
K^b_o = \begin{bmatrix}
< \alpha_1, \alpha_{k+1} >, \ldots, < \alpha_1, \alpha_{k+k_b} > \\
< \alpha_{k+1}, \alpha_{k+1} >, \ldots, < \alpha_{k}, \alpha_{k+k_b} > \\
\vdots \\
< \alpha_{k+k_b}, \alpha_{k+1} >, \ldots, < \alpha_{k+b}, \alpha_{k+k_b} >
\end{bmatrix}
\]

\[
K^c_o = \begin{bmatrix}
< \alpha_{k+1}, \alpha_{k+1} >, \ldots, < \alpha_{k+1}, \alpha_{k+k_b} > \\
< \alpha_{k+k_b}, \alpha_{k+1} >, \ldots, < \alpha_{k+k_b}, \alpha_{k+k_b} >
\end{bmatrix}
\]

where \( \alpha_t \) represents \( \phi(c_i) \).

The time complexity for computing \( \tilde{K}_o \). \( K^b_o \) and \( K^c_o \) are \( O(dk_n) \), \( O(dk_b) \), and \( O(dk^2) \), respectively.

Similarly, we update \( K_{c \times 2} \) in two parts by

\[
\tilde{K}_{c \times 2} = [K'_{c \times 2}, K^b_{c \times 2}]^T,
\]

where \( K'_{c \times 2} \) is the updated kernel vector after inserting labeled new samples by (23), and \( K^b_{c \times 2} \) is the kernel vector after appending new classes by (24), where \( \beta \) denotes \( \phi(z) \).

\[
K'_{c \times 2} = \begin{bmatrix}
< \alpha_1, \beta >, \ldots, < \alpha_t, \beta >, \ldots, < \alpha_k, \beta >
\end{bmatrix}
\]

\[
K^b_{c \times 2} = \begin{bmatrix}
< \alpha_{k+1}, \beta >, \ldots, < \alpha_{k+b}, \beta >
\end{bmatrix}
\]

As we can see, the updating of known-class centers by new arriving samples does not change the rank of the kernel matrix, so as to the dimension of feature space. And the positions of known-class centers do not change under the constraint of (18). While inserting a novel class is equal to increasing one rank of the kernel matrix, resulting in appending a new basis perpendicular to the original space, see the demonstration in Figure 1. And the center of a new class is also projected into the unit position of the new axis, therefore its boundary threshold can be set to the same value as the known classes.

Both of these features make IFKDA more stable and robust for classification, that will be verified by experiments on real datasets in Section 5.

We then compare space and time complexities for updating chunk data for IFKDA, ILDA/SSS [Kim et al., 2011], and ILDA/QR [Chu et al., 2015] in Table 2. In both respects, the updating costs of IFKDA are minimal, especially when \( k \) is far less than \( n (k << n) \) the advantages are more obvious. Besides, IKNDA only has the algorithm for learning novel classes, with \( O(ns + s^2) \) and \( O(d(n+s)+ns + n^2s + ns^2) \) in space and time complexities (where \( s \) is the sample number in a new class), respectively, for the insertion of one novel class.

In contrast, IFKDA only needs extra space of \( (d+k+1) \), and has \( O(dk + d) \) in time complexity in this case.

<table>
<thead>
<tr>
<th>Method</th>
<th>Space</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILDA/SSS</td>
<td>( O(dh + dk_b) )</td>
<td>( O(dk^2 + n^2) )</td>
</tr>
<tr>
<td>ILDA/QR</td>
<td>( O(dh + k^2 + h^2) )</td>
<td>( O(dkh + dk^2 + dh^2) )</td>
</tr>
<tr>
<td>IFKDA</td>
<td>( O(dk_b + kk_b) )</td>
<td>( O(dh + df + dk^2) )</td>
</tr>
</tbody>
</table>

Table 2: The space and time complexities of incremental DAs for learning chunk data. (\( h \) is the sample number in a chunk, \( k_b \) is the new class number in a chunk, \( n \) = \( n + h \), \( k = k + k_b \), and \( f \) is the total class number in a chunk.)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>( d )</th>
<th>( k_i )</th>
<th>( n_i )</th>
<th>( k_{z/b} )</th>
<th>( n_{c_i} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR</td>
<td>2000</td>
<td>100</td>
<td>6</td>
<td>5/4</td>
<td>4</td>
</tr>
<tr>
<td>MNIST-F</td>
<td>784</td>
<td>5</td>
<td>1000</td>
<td>1/1</td>
<td>50</td>
</tr>
<tr>
<td>AWA</td>
<td>4096</td>
<td>40</td>
<td>56</td>
<td>5/2</td>
<td>18</td>
</tr>
<tr>
<td>Caltech256</td>
<td>4096</td>
<td>200</td>
<td>37</td>
<td>5/3</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 3: The settings of datasets. (\( d \): the dimension of data, \( k_i \): the training class number, \( n_i \): the sample number in each training class, \( k_{c_i} \): the known class number in a chunk, \( k_b \): the novel class number in a chunk, and \( n_{c_i} \): the sample number of each class in a chunk)

5 Experimental Results and Analysis

In this section, we evaluate the performances of the proposed methods on four publicly-available datasets: AR [Kim et al., 2011], AWA [Lampert et al., 2014], Caltech256 [Griffin et al., 2007] and MNIST-Fashion (MNIST-F for short) [Xiao et al., 2017]. For AWA and Caltech256, we take the outputs from the 7-th full connected layer of very deep 19-layer CNN as features (4096 dims). The kernel function: \( \text{exp}(-\|x-y\|^2/\sigma) \) is used for all kernel DAs. Since FKDA uses approximate centers to construct kernel matrix as AKDA/QR, \( \sigma \) should be assigned a large value. Experiments show that choosing (\( \sigma = d \)) for FKDA(IFKDA) and AKDA/QR produce good overall results, we thus use this value for them in all the experiments. All methods are implemented in MATLAB and ran on an Intel (R) Core (TM) i7 PC with 3.40 GHz CPU and 8 GB RAM.

5.1 Multi-class Novelty Detection for Chunks

The settings of initial batch training stage and on-line incoming chunk data are in Table 3. Table 4 tabulates the results for FKDA and KNDa for detecting five unlabeled chunks with 20-fold cross-validation.

As we can see, although KNDa (\( \sigma = 1 \)) produces good overall results on AWA and Caltech256, and KNDa (\( \sigma = d \)) performs better than KNDa (\( \sigma = 1 \)) on AR and MNIST-F, the overall misclassification rates (denoted by Err) of FKDA (\( \sigma = d \)) are far less than those of KNDa on all datasets.

5.2 Incremental Learning

We first compare the performances of IFKDA and ILDA/QR with batch methods AKDA/QR, KNDa (\( \sigma = 1 \)), and FKDA in learning chunk data streams on AWA and Caltech256.

The settings of experimental data are the same as those in Table 3. The testing data are randomly selected from the rest samples of training classes and the novel classes with
Table 4: The performance of novelty detection for chunk data streams. 

<table>
<thead>
<tr>
<th>Method</th>
<th>Dataset</th>
<th>( \text{Err} )</th>
<th>( F_a )</th>
<th>( F_b )</th>
<th>( F_c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNDA (( \sigma = 1 ))</td>
<td>AR</td>
<td>77.8</td>
<td>50.0</td>
<td>50.0</td>
<td>50.0</td>
</tr>
<tr>
<td></td>
<td>MNIST-F</td>
<td>82.0</td>
<td>80.0</td>
<td>20.0</td>
<td>64.0</td>
</tr>
<tr>
<td></td>
<td>AWA</td>
<td>12.9</td>
<td>40.0</td>
<td>1.2</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>Caltech256</td>
<td>43.0</td>
<td>81.3</td>
<td>20.0</td>
<td>0.0</td>
</tr>
<tr>
<td>KNDA (( \sigma = d ))</td>
<td>MNIST-F</td>
<td>48.4</td>
<td>75.5</td>
<td>12.0</td>
<td>14.8</td>
</tr>
<tr>
<td></td>
<td>AWA</td>
<td>28.0</td>
<td>56.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>Caltech256</td>
<td>58.8</td>
<td>45.3</td>
<td>66.8</td>
<td>0.0</td>
</tr>
<tr>
<td>FKDA (( \sigma = d ))</td>
<td>AR</td>
<td>20.7</td>
<td>0.5</td>
<td>36.8</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>MNIST-F</td>
<td>22.0</td>
<td>44.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>AWA</td>
<td>5.7</td>
<td>17.0</td>
<td>1.2</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>Caltech256</td>
<td>31.0</td>
<td>68.0</td>
<td>8.8</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Err = \((F_a + F_b + F_c)/N_c\) is the overall percentage for misclassification, where \( F_a \) is the number of instances belonging to novel classes but misclassified into known classes, \( F_b \) is the number of instance belonging to known classes but misclassified into novel classes, and \( F_c \) is the number instance belonging to some known classes but misclassified into other known classes in a chunk. \( N_c \) is the total instance number in a chunk. \( N_o \) is the total instance number of novel classes in a chunk. \( F_a = F_{n_o}/N_c \); \% of novel-class instances falsely identified to known classes. \( F_b = F_{p_o}/(N_c - N_o) \); \% of known-class instances falsely identified to novel classes; \( F_c = F_{e_o}/(N_c - N_o) \); \% of known-class instances falsely identified to other known classes.

the same scale of chunk data. The results of classification accuracy and CPU time on two datasets are shown in Figure 2 and Figure 3 with 20-fold-cross validation, respectively. We have the following observations:

1) IFKDA is the exact incremental version of FKDA, giving the same accuracies to FKDA in all tests. The accuracies of IFKDA and FKDA are comparable to the best algorithm. All kernel DAs are more accurate than ILDA/QR.

2) Both IFKDA and ILDA/QR are far faster than batch methods, which verifies the efficiency of incremental methods over the batch ones. IFKDA performs about 5-10 times faster than FKDA, hundreds of times faster than AKDA, and thousands of times faster than KNDA on both datasets. Although ILDA/QR runs similar as IFKDA on AWA (\( d = 4096 \) and \( n = 2240 \)), but it can not work on Caltech256 (with \( d = 4096 \) and \( n = 7400 \)), due to the limitation of \( (d >> n) \) for \( Q \) matrix of \( X \) in its algorithm.

3) FKDA spends less time than other batch methods in the training stage on both datasets. FKDA even has five times speed up over LDA/QR (the batch version of ILDA/QR, used in the initial training stage) on AWA. FKDA is dozens of times faster than AKDA/QR on AWA with comparable accuracy, and hundreds of times faster than AKDA/QR on Caltech256 with higher accuracy, even though they share the common on using centers as the input.

We then compare the classification accuracy and CPU time of IFKDA with IKNDA (\( \sigma = 1 \)) in learning novel classes on Caltech256. AKDA/QR and FKDA are also taken as the baseline. The settings of batch training stage are the same as Table 3. And in on-line stage, fifteen novel classes (each has \( n_c \) samples) are inserted one by one. The testing samples are
We propose a fast factorization-free kernel method (FKDA) by constructing an approximate kernel centroid matrix to solve a linear system in kernel space. Then the incremental FKDA (IFKDA) runs extremely fast by updating part of matrices. Both theoretical analysis and empirical experiments on several benchmarks validate the superiority of FKDA and IFKDA against the state-of-the-art methods. By taking advantage of our proposed algorithms, the kernel methods may be more widely applied especially when handling incremental learning problems for chunk data streams.

Acknowledgments
This research has been supported by the National Key Research and Development Program of China (Grant No. 2017YFB1103704), and the National Natural Science Foundation of China (Grant No. 61733002 and No.61572096).

A Proof of Theorem 1
The within class scatter in kernel space is

\[ S_w^\Phi = X^\Phi \left( I - \begin{bmatrix} \varepsilon_1 & \cdots & \varepsilon_k \end{bmatrix} \right) (X^\Phi)^T, \]  

where \( \varepsilon = \frac{1}{n} ee^T, \varepsilon_i = \frac{1}{n_i} e_i e_i^T, e = [1..1]^T \in \mathbb{R}^{n \times 1} \) and \( e_i = [1..1]^T \in \mathbb{R}^{n_i \times 1} \). Thus

\[ (G^\Phi)^T S_w^\Phi G^\Phi = (G^\Phi)^T X^\Phi \left( I - \begin{bmatrix} \varepsilon_1 & \cdots & \varepsilon_k \end{bmatrix} \right) (X^\Phi)^T G^\Phi \]

\[ = E^T \left( I - \begin{bmatrix} \varepsilon_1 & \cdots & \varepsilon_k \end{bmatrix} \right) E \]

\[ = E^T E - E^T \begin{bmatrix} \frac{e_1}{\sqrt{n_1}} & \cdots & \frac{e_k}{\sqrt{n_k}} \\ \cdot & \cdot & \cdot \\ \frac{e_1}{\sqrt{n_1}} & \cdots & \frac{e_k}{\sqrt{n_k}} \end{bmatrix} E \]

\[ = \begin{bmatrix} n_1 & \cdots & 0 \\ \cdots & \cdots & \cdots \\ 0 & \cdots & n_k \end{bmatrix} - \begin{bmatrix} \sqrt{n_1} & \cdots & 0 \\ \cdots & \cdots & \cdots \\ 0 & \cdots & \sqrt{n_k} \end{bmatrix} \]

\[ \times \begin{bmatrix} \sqrt{n_1} & \cdots & 0 \\ \cdots & \cdots & \cdots \\ 0 & \cdots & \sqrt{n_k} \end{bmatrix} = 0. \]

Since \( S_b^\Phi = S_b^\phi + S_w^\phi \), therefore,

\[ (G^\phi)^T S_b^\phi G^\phi = (G^\phi)^T S_b^\phi G^\phi. \]  

And

\[ S_b^\phi = X^\phi \left( \begin{bmatrix} \varepsilon_1 & \cdots & \varepsilon_k \end{bmatrix} - \varepsilon \right) (X^\phi)^T \]
We have the following result:

\[(G^\phi)^T S_b^\phi G^\phi = (G^\phi)^T X^\phi \left( \begin{array}{c}
\varepsilon_1 \\
\vdots \\
\varepsilon_k
\end{array} \right) - \varepsilon \right) (X^\phi)^T G^\phi
\]

\[= E^T \left( \begin{array}{c}
\varepsilon_1 \\
\vdots \\
\varepsilon_k
\end{array} \right) - \varepsilon \right) E \]

\[= \left[ \begin{array}{cccc}
n_1 & \cdots & n_k \\
\vdots & \ddots & \vdots \\
n_1 & \cdots & n_k
\end{array} \right] - \frac{1}{n} \left[ \begin{array}{cccc}
n_1 & \cdots & n_k \\
\vdots & \ddots & \vdots \\
n_1 & \cdots & n_k
\end{array} \right] = \left[ \begin{array}{c}
1 \\
\vdots \\
1
\end{array} \right] \cdot \left[ \begin{array}{c}
n_1 \\
\vdots \\
n_k
\end{array} \right].
\]

Thus, \( \text{rank}((G^\phi)^T S_b^\phi G^\phi) = k - 1 \), which together with (27) yields

\[\text{trace}((G^\phi)^T S_t^\phi G^\phi((G^\phi)^T S_b^\phi G^\phi) = k - 1, \]

(30)

so \( G^\phi \) is a solution of (7).

References


[Cai et al., 2007] Deng Cai, Xiaofei He, and Jiawei Han. Efficient kernel discriminant analysis via spectral regression. In Proceedings of Seventh IEEE International Conference on Data Mining (ICDM ‑ 07), pages 427–432. IEEE, 2007.


