> Technical Paper

Kernel Principal Component Analysis Using SAS

Kai Shen and Zohreh Asgharzadeh
Research and Development Department, Internet of Things

Contents

Kernel Principal Component Analysis Using SAS

Abstract

In the realm of multivariate statistics, kernel principal component analysis (KPCA) is an extension of principal component analysis (PCA) that uses kernel methods. Compared with PCA, KPCA better exploits the complicated spatial structure of high-dimensional features. KPCA provides eigenvectors of a kernel matrix that capture nonlinear structures within data sets. To explore nonlinear relationships, in general a full $n \times n$ kernel matrix needs to be constructed over all n data points, and thus poses a burden on space and time resources for large values of n. Techniques such as the Nyström method can be applied to alleviate this problem. In this paper, we introduce KPCA and a low-rank approximation method that uses *k*-means clustering and greatly reduces the space and time complexity of a KPCA implementation.

Introduction

Kernel principal component analysis (kernel PCA) is a nonlinear form of PCA [2]. It uses the same basic idea as PCA uses; that is, it seeks to project the set of data onto a low-dimensional subspace that captures the highest possible amount of variance in the data. Whereas PCA performs a linear projection of the data onto a subset of the original space, kernel PCA uses a mapping function to embed the data in a high-dimensional RKHS (reproducing kernel Hilbert space) called $\mathcal{F}$ by a linear dimensionality reduction through the “kernel trick” in that space. Different kernels correspond to different mapping functions. This way, we can find a nonlinear subspace (with respect to the original input space) that contains the data. The applications of kernel PCA include nonlinear dimensionality reduction, nonlinear data classification, kernel principal component regression, image denoising, and novelty detection.

To perform an exact KPCA when the input matrix M is of size $n \times m$, the full kernel matrix $K \in \mathbb{R}^{n\times n}$ needs to be constructed and the expensive eigendecomposition operation, with computational complexity of $\mathcal{O}(n^3)$, must be applied on K. So for large values of n, the exact KPCA method can be very slow. To address this issue, an approximation method called the Nyström method was proposed in [1]. This method is based on sampling from the rows of the input matrix. Instead of the eigendecomposition method being applied on the full kernel matrix K, it is applied on a smaller matrix of size $c \times c$, where c is the sample size. In this technical paper, we further explain the details of this approximation method. The sampling scheme is important in the performance of Nyström method. In [3], an error analysis is conducted and the resulting error bound suggests using *k*-means clustering as the sampling scheme. The greater the number of clusters chosen in *k*-means clustering, the more accurate the reconstruction of K would be. If the number of clusters equals the number of observations, the reconstruction will be exact. Low-Rank approximation reduces the memory complexity of kernel PCA from $\mathcal{O}(n^2)$ to $\mathcal{O}(nc)$ and reduces the computational complexity of kernel PCA from $\mathcal{O}(n^3)$ to $\mathcal{O}(nc^2)$.

Methodology

Suppose the mean of the data in an RKHS is

$$\mu = \frac{1}{n}\sum_{i=1}^{n}\phi(x_i) = 0 \tag{1}$$

where x_i is a $q \times 1$ data point in the original space and $\phi(\cdot)$ is the mapping function that maps the original data to some high-dimensional RKHS. Then, assuming that the data vector x_i is already centered, the covariance matrix is:

$$\mathrm{C} = \frac{1}{n}\sum_{i=1}^{n} \phi(x_i)\phi(x_i)^T \tag{2}$$

The eigen-system of C is

$$\mathrm{C}v = \lambda v \tag{3}$$

Schölkopf, Smola, and Müller in [2] show that the eigenvectors can be expressed as a linear combination of features in RKHS, namely

$$v = \sum_{i=1}^{n} \alpha_i \phi(x_i) \tag{4}$$

Since

$$\mathrm{C}v = \frac{1}{n}\sum_{i=1}^{n} \phi(x_i)\phi(x_i)^T v = \lambda v \tag{5}$$

thus

$$v = \frac{1}{\lambda n}\sum_{i=1}^{n} \phi(x_i)\phi(x_i)^T v = \frac{1}{\lambda n}\sum_{i=1}^{n} \underbrace{\left(\phi(x_i)\cdot v\right)}_{\text{scalar}} \phi(x_i) \tag{6}$$

This means that all solutions v with $\lambda \neq 0$ lie in the span of $\phi(x_1), \ldots, \phi(x_n)$; that is,

$$v = \sum_{i=1}^{n} \alpha_i \phi(x_i) \tag{7}$$

Finding the eigenvectors is equivalent to finding the coefficients α_i. In what follows, we demonstrate how Schölkopf, Smola, and Müller in [2] drove the values of α_i.

By substituting (7) back into (5), for jth eigenvector, we get

$$\frac{1}{n}\sum_{i=1}^{n} \phi(x_i)\phi(x_i)^T \Big(\sum_{l=1}^{n} \alpha_{jl}\phi(x_l)\Big) = \lambda_j \sum_{l=1}^{n} \alpha_{jl}\phi(x_l) \tag{8}$$

Rewrite it as

$$\frac{1}{n}\sum_{i=1}^{n} \phi(x_i)\Big(\sum_{l=1}^{n} \alpha_{jl}k(x_i, x_l)\Big) = \lambda_j \sum_{l=1}^{n} \alpha_{jl}\phi(x_l) \tag{9}$$

Multiply this by $\phi(x_k)^T$ from the left:

$$\frac{1}{n}\sum_{i=1}^{n} \phi(x_k)^T\phi(x_i)\Big(\sum_{l=1}^{n} \alpha_{jl}k(x_i, x_l)\Big) = \lambda_j \sum_{l=1}^{n} \alpha_{jl}\phi(x_k)^T\phi(x_l). \tag{10}$$

By plugging in the kernel and rearranging the equation, we get

$$K^2\boldsymbol{\alpha}_j = n\lambda_j K\boldsymbol{\alpha}_j \tag{11}$$

where the $n \times n$ matrix K is called the kernel matrix, whose klth element is the value of kernel function $k(x_k, x_l) = \phi(x_k)^T \phi(x_l)$; and $\boldsymbol{\alpha}_j$ is a $n \times 1$ vector, whose lth element is the coefficient α_{jl}.

Now we remove factor K from both sides and finally rewrite the eigensystem in terms of kernel matrix K as

$$K\boldsymbol{\alpha}_j = n\lambda_j \boldsymbol{\alpha}_j \tag{12}$$

We apply a normalization condition such that the eigenvector v_j in the RKHS satisfies

$$v_j^T v_j = 1 \Longrightarrow \sum_{k=1}^{n}\sum_{l=1}^{n} \alpha_{jl}\alpha_{jk}\phi(x_l)^T\phi(x_k) = 1 \Longrightarrow \boldsymbol{\alpha}_j^T K \boldsymbol{\alpha}_j = 1 \tag{13}$$

By left-multiplying $K\boldsymbol{\alpha}_j = n\lambda_j\boldsymbol{\alpha}_j$ by $\boldsymbol{\alpha}_j^T$ and using the normalization condition, we get:

$$n\lambda_j \boldsymbol{\alpha}_j^T \boldsymbol{\alpha}_j = 1 \qquad \forall j \tag{14}$$

The projection of a new or old data point x onto the jth principal component is:

$$\phi(x)^T v_j = \sum_{i=1}^{n} \alpha_{ji}\phi(x)^T\phi(x_i) = \sum_{i=1}^{n} \alpha_{ji} K(x, x_i) \tag{15}$$

Low-Rank Approximation Using the Nyström Method

This section describes the Nyström approach that is proposed in [3] and uses a sampling method to approximate the kernel matrix.

Consider the integral equation

$$\int p(y)k(x,y)\phi_i(y)\mathrm{d}y = \lambda_i\phi_i(x) \tag{16}$$

where $p(y)$ is a probability distribution, k is a kernel function, and λ_i's and ϕ_i's are eigenvalues and eigenfunctions, respectively. Given i.i.d. samples $(x_1, x_2, \ldots, x_q)$, we can approximate the preceding integral by the empirical average,

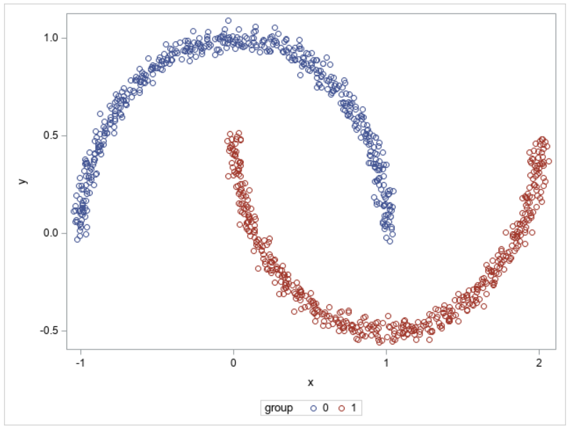
$$\frac{1}{q}\sum_{j=1}^{q} k(x, x_j)\phi_i(x_j) \approx \lambda_i\phi_i(x) \tag{17}$$

which leads to a standard eigenvalue problem $K^{(q)}U^{(q)} = U^{(q)}\Lambda^{(q)}$, where $K_{ij}^{(q)} = k(x_i, x_j)$ for $i, j = 1, 2, \ldots, q$, and $U^{(q)} and \Lambda^{(q)} \in \mathbb{R}^{q\times q}$. Thus $\phi_i(x_j) \approx \sqrt{q}U_{ij}^{(q)}$ and $\lambda_i \approx \lambda_i^{(q)}/q$. In [8] an approximation for the ith eigenfunction is proposed as

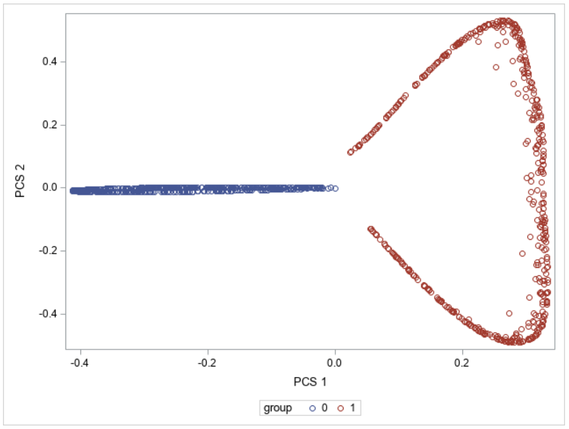
$$\phi_i(y) \approx \frac{\sqrt{q}}{\lambda_i^{(q)}} \sum_{k=1}^{q} k(y, x_k) U_{k,i}^{(q)} \tag{18}$$

Suppose the data set is $\mathcal{X} = \{x_i\}_{i=1}^{n}$ and K is an $n \times n$ full kernel matrix. Thus the eigensystem of K is

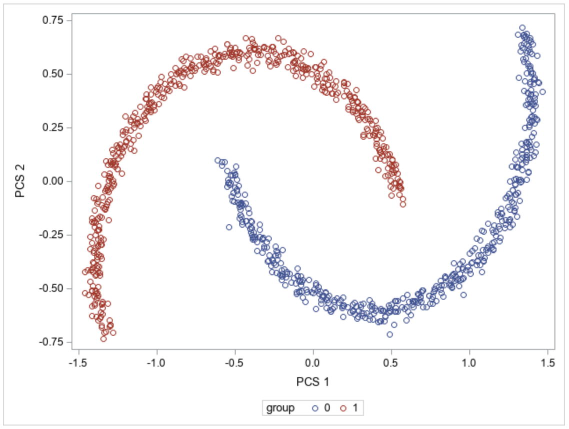
$$K\Phi_K = \Phi_K\Lambda_K \tag{19}$$



(a) Original “half-moon” pattern



(b) Projection onto the first two kernel PCs



(c) Projection onto the first two PCs

Figure 1: Comparison of Kernel PCA and PCA

In our implementation of KPCA in SAS/IML, we choose q landmark points, $\mathcal{Z} = \{z_j\}_{j=1}^{q}$, by using *k*-means clustering instead of sampling. The first q eigenvectors and eigenvalues of the full eigensystem can be approximated by

$$\Phi_K \approx \sqrt{\frac{q}{n}} E \Phi_z \Lambda_z^{-1}, \quad \Lambda_K \approx \frac{n}{q} \Lambda_z \tag{20}$$

where $E \in \mathbb{R}^{n \times q}$ with $E_{ij} = k(x_i, z_j)$, Φ_z and $\Lambda_z \in \mathbb{R}^{q \times q}$ are eigenvectors and eigenvalues of $W \in \mathbb{R}^{q \times q}$, and $W_{ij} = k(z_i, z_j)$. Thus the full kernel matrix can be approximated as

$$K \approx \left(\sqrt{\frac{q}{n}} E \Phi_z \Lambda_z^{-1}\right) \left(\frac{n}{q} \Lambda_z\right) \left(\sqrt{\frac{q}{n}} E \Phi_z \Lambda_z^{-1}\right)^T = E W^{-1} E^T \tag{21}$$

Figure 1 demonstrates a simple comparison between kernel PCA and PCA. We can see that the two nonlinear “half-moon” patterns become linearly separable after the data points are projected on the first two kernel principal components. However, projecting these data points on the first two principal components does not make the data points between the two half moon patterns linearly separable.

KPCA in SAS/IML

The KPCA function is implemented in SAS/IML 9.4 and does an eigendecomposition on the centered kernel matrix. Currently, three types of kernels are supported:

- linear kernel $k(x_i, x_j) = (x_i \cdot x_j)$, where $i, j \in (1, \dots, n)$, which is equivalent to PCA.
- polynomial kernel $k(x_i, x_j) = (x_i \cdot x_j + c)^d$, where $d \geq 1$ is the polynomial order and $c \geq 0$ is a free parameter that trades off the influence of higher-order versus lower-order terms.
- Gaussian kernel $k(x_i, x_j) = e^{(-\frac{\|x_i - x_j\|^2}{2s^2})}$, where $s > 0$ is the Gaussian kernel parameter (bandwidth).

In the exact KPCA method, all elements of the kernel matrix $K_{n\times n} = \left(k(x_i, x_j)\right)_{ij}$ are constructed. Next, the centered kernel matrix is calculated as $\tilde{K}_{n\times n} = K_{n\times n} - \mathbf{1}_{1/n}K_{n\times n} - K_{n\times n}\mathbf{1}_{1/n} + \mathbf{1}_{1/n}K_{n\times n}\mathbf{1}_{1/n}$, where $\mathbf{1}_{1/n}$ is an $n \times n$ matrix with all elements equal to $1/n$. Then eigendecomposition is performed on matrix $\tilde{K} = U\Lambda U^T$. The normalized version of U with respect to space $\mathcal{F}$ is $U\Lambda^{-1/2}$.

The computation steps of the low-rank approximation method for KPCA are as follows:

1. The centroids are obtained from *k*-means clustering; they are denoted as $z_i \ (i = 1, \dots, c)$.
2. The kernel matrix between the training data and the centroids is constructed as $E_{n\times c} = \left(k(x_i, z_j)\right)_{ij}$.
3. The kernel matrix between the centroids themselves is calculated as $\mathbf{W}_{c\times c} = \left(k(z_i, z_j)\right)_{ij}$.
4. $\mathbf{P}^{-1} = U\Lambda^{-1/2}U^T$ is formed, where $U\Lambda U^T$ is the eigendecomposition of W.
5. $G = \mathbf{E}\mathbf{P}^{-1}$ is formed, and the centered version $\tilde{G} = (G - \bar{G})$ is calculated, where $\bar{G}$ is column means of G.
6. Eigenvector decomposition on $\tilde{G}^T\tilde{G} = V\Lambda V^T$ is performed.
7. The centered kernel matrix is approximated as $\tilde{K} = \tilde{G}\tilde{G}^T$, and the first c eigenvectors of $\tilde{K}_{n\times n}$ can be approximated by $U \approx \tilde{G}V\Lambda^{-1}$.

In order to score p new data points after training, we first calculate the kernel matrix between the new data points and the training data: $K_{p\times n} = k(y_j, x_i)$ for $j = 1, \dots, p$ and $i = 1, \dots, n$. Next we calculate the centered version of this kernel matrix, $\tilde{K}_{p\times n} = K_{p\times n} - \mathbf{1}_{1/n}K_{n\times n} - K_{p\times n}\mathbf{1'}_{1/n} + \mathbf{1}_{1/n}K_{n\times n}\mathbf{1'}_{1/n}$, where $K_{n\times n}$ is the kernel matrix of the training data, $\mathbf{1}_{1/n}$ is a $p \times n$ matrix with all elements equal to $1/n$, and $\mathbf{1'}_{1/n}$ is an $n \times n$ matrix with all elements equal to $1/n$. The projection of the new data onto the first q principal components produces the first q columns of $S = \tilde{K}_{p\times n}U\Lambda^{-1/2}$.

Examples

In this section, we provide examples to demonstrate some applications of kernel PCA along with implementation in SAS/IML. For more information about KPCA implementation in SAS/IML, see the sections “KPCATRAIN Call” and “KPCASCORE Function” in [9].

Visualization

In this example, we illustrate how to use KPCA for visualization and we compare the visualization results with those of PCA. The data set that is used for this example is the Statistical Control Chart Time Series data available in the UCI Machine Learning Repository [6]. This data set includes 600 examples of control charts that are generated synthetically by the process explained in [4, 5] and are used to monitor the behavior of the system. There are six different classes in the data set, and each class has 100 instances. The classes of the data set include: normal, cyclic, increasing trend, decreasing trend, upward shift, and downward shift. All classes except the normal class illustrate that the process being monitored is not functioning correctly and requires adjustment. The goal here is to use KPCA (with Gaussian kernel) and PCA to reduce the data dimensions to two so that we can see whether the classes are well separated. In this example, we choose 50% of the samples in each class to train the KPCA model, but we project all 600 instances onto the principal components for visualization. The following statements run the KPCATRAIN function in SAS/IML to train a KPCA model, and then run the KPCASCORE function to project the data onto the first two kernel principal components. Here scct_train and scct_test are the names of the training data set and the test data set, respectively.

```
/********Visualization example **********/
%let url=https://archive.ics.uci.edu/ml/machine-learning-databases/
synthetic_control-mld/synthetic_control.data;

filename cc url "&url";

data scct_test;
infile cc;
input var1-var60;
length type $20;
obsid = _n_;
select;
when( 1 <= _n_ <= 100 )    type = "Normal";
when( 101 <= _n_ <= 201 )  type = "Cyclic";
when( 201 <= _n_ <= 301 )  type = "Increasing Trend";
when( 301 <= _n_ <= 401 )  type = "Decreasing Trend";
when( 401 <= _n_ <= 501 )  type = "Upward Shift";
when( 501 <= _n_ <= 601 )  type = "Downward Shift";
end;
run;

filename cc clear;

proc sort data=scct_test;
by type obsid;
run;

proc surveyselect data=scct_test
      method=srs n=50
      seed=100 out=scct_train;
```

```
   strata type;
run;


proc iml;
varName = "var1":"var60";

use scct_train;
   read all var (varName) into X_train[c=varName];
close;

use scct_test;
   read all var (varName) into X_test[c=varName];
close;

 /*standardization*/
std=std(X_test);
mean=mean(X_test);

X_train=(X_train-mean)/std;
X_test=(X_test-mean)/std;


/* Kernel PCA */
opt = {.,         /* eigenvalue cutoff (default) */
       2,         /* Gaussian kernel */
       4,         /* kernel parameter=4 */
       1,         /* exact method for KPCA */
       2};        /* project on 2 PCs */
call KPCATrain(eigenVal, eigenVec, centroids, RowMeans, score,
               X_train, opt);
opt= {2, 2, 4};  /* score options, project on 2 PC's, Gaussian kernel, kernel parameter=4 */
score=KPCAScore(eigenVec, X_train, RowMeans, X_test, opt);

varNames = {"score_1" "score_2"};
create score from score [colname=varNames];
append from score;
close score;

submit;
data scoreByID;
   merge score scct_test(keep=type);
run;

ods graphics on / attrpriority=none;
proc sgplot data=scoreByID;
  title "Score Plot: Kernel PCA";
  styleattrs datasymbols=(circlefilled squarefilled starfilled X Triangle Asterisk);
  scatter x=score_1 y=score_2 /group=type markerattrs=(size=6px);
run;
ods graphics / reset;
endsubmit;
quit;
```

In Figure 2, we plot the projections of the data in 2D by using the first two principal components of KPCA and the first two principal components of PCA. We can clearly observe that the projections that use KPCA result in a better separation of the classes, whereas the projections that use the PCA are heavily overlapped.

Dimension Reduction

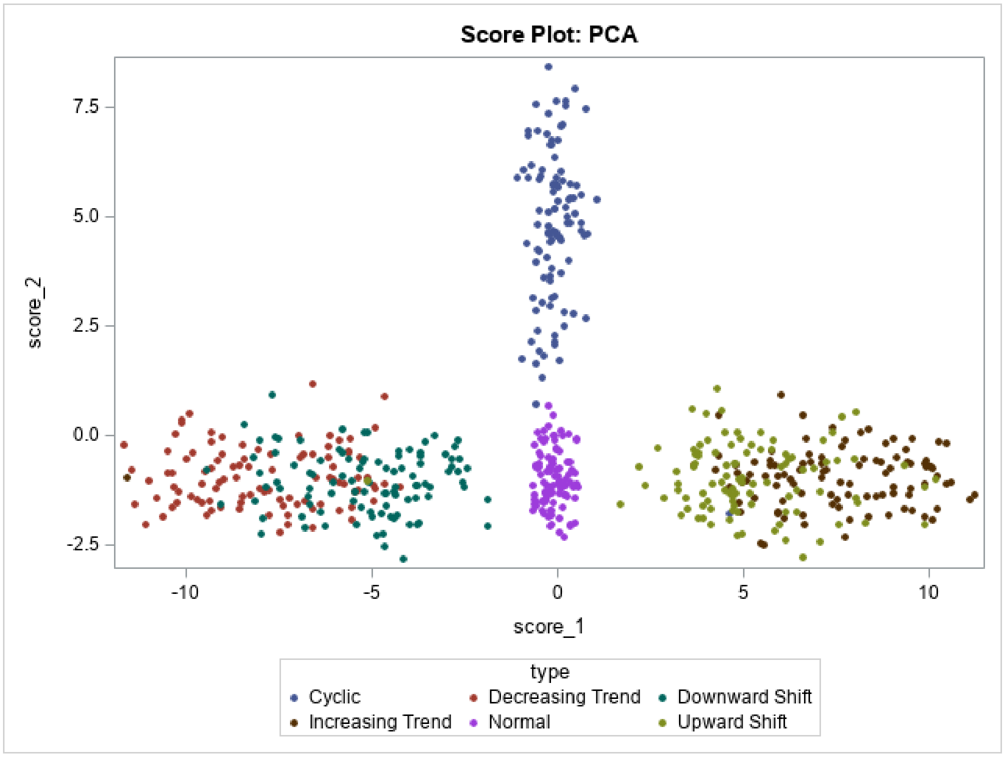
In this example, we demonstrate the effectiveness of KPCA in dimension reduction and compare it with that of PCA. We use the Ionosphere data set, which is also available in the UCI Machine Learning Repository [6]. This data set contains radar data that were collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power of about 6.4 kilowatts. The targets are free electrons in the ionosphere. “Goo” radar returns are those that show evidence of a certain type of structure in the ionosphere.“Bad” returns are those that do not show this evidence and pass through the ionosphere. Received signals are processed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There are 17 pulse numbers for the Goose Bay system. Instances in this database are described by two attributes per pulse number, So the total number of attributes is 34. The goal is to apply KPCA and PCA to reduce these 34 dimensions, and then use linear discriminant analysis (LDA) on the reduced dimensions to classify the radar returns as “Good” or “Bad”. Here we randomly choose a training set of size 280 and a test set of size 71. In both KPCA and PCA, a total of 18 principal components are used.

The following statements run the KPCATrain function in SAS/IML to train a KPCA model, and run the KPCASCORE function to project the data onto the first 18 kernel principal components. Next the DISCRIM procedure, available in SAS/STAT software, is applied on the reduced dimensions. Similar steps were taken for the PCA. Here ionosphere_train and ionosphere_test are the names of the training data set and the test data set, respectively.

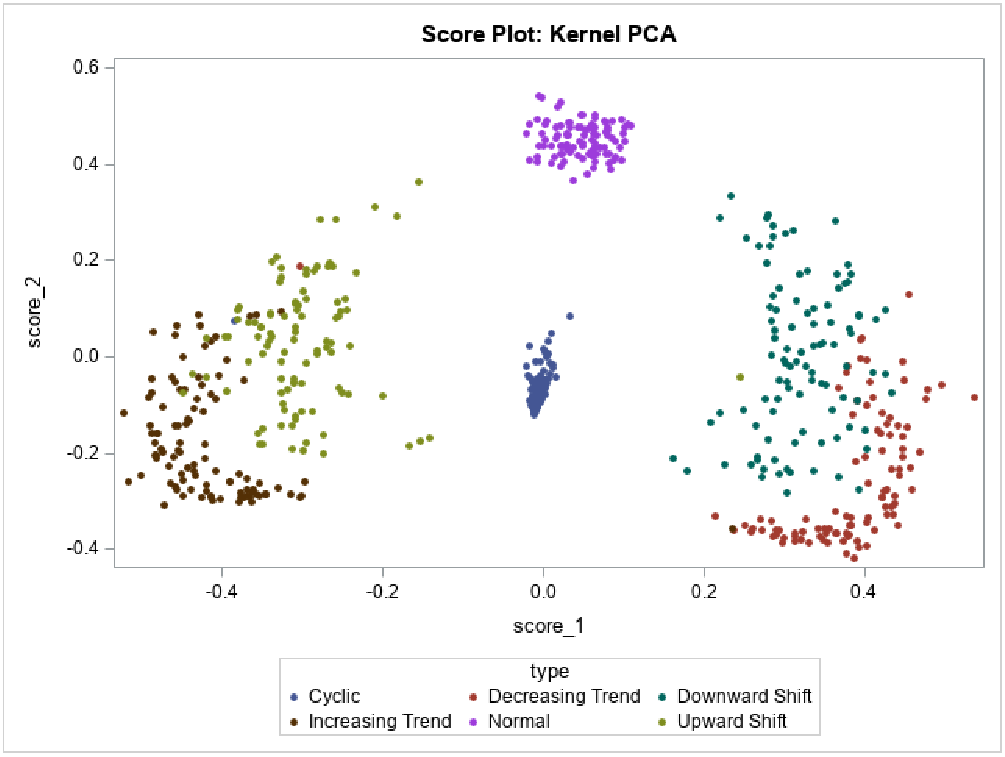
```
/*******Radar example *******/
%let url=https://archive.ics.uci.edu/ml/machine-learning-databases/ionosphere/ionosphere.data;
filename cc url "&url";

data ionosphere;
infile cc delimiter=',';
input var1 +1 var2-var33 group $;
obsid = _n_;
run;

data ionosphere;
set ionosphere;
if group="g" then group_n=1;
else if group="b" then group_n=0;
```



(a) Projection onto the first two PCs by PCA



(b) Projection onto the first two PCs by KPCA

Figure 2: Visualization of SCCTS data by Kernel PCA and PCA

```
run;

/********Split data set to traning (280) and testing (71) ****/
proc surveyselect data=ionosphere
      method=srs n=280
      seed=100 out=ionosphere_train;
run;

proc sql;
create table ionosphere_test as
select * from ionosphere
where obsid not in (select obsid from ionosphere_train);
quit;
proc iml;
varNames = ("var1":"var33"); /*remove constant variable var2*/
use ionosphere_train;
   read all var varNames into X_train[c=varName];
   read all var "group_n" into group_train;
close;


use ionosphere_test;
   read all var varNames into X_test[c=varName];
   read all var "group_n" into group_test;
close;


/* Kernel PCA */
opt = {.,         /* eigenvalue cutoff (default) */
       2,         /* Gaussian kernel */
       2,         /* kernel parameter=2 */
       1,         /* exact method for PCA */
       18};        /* project on 18 PCs */
call KPCATrain(eigenVal, eigenVec, centroids, RowMeans, score_train,
               X_train, opt);
opt= {18, 2, 2};
score_test=KPCAScore(eigenVec, X_train, RowMeans, X_test, opt);

/* save score values to SAS dataset */
varNames = ("var_1":"var_19");

score_train=score_train||group_train;
create score_train from score_train [colname=varNames];
append from score_train;
close score_train;

score_test=score_test||group_test;
create score_test from score_test [colname=varNames];
```

```
append from score_test;
close score_test;

submit;
proc discrim data=score_train method=normal pool=yes
testdata=score_test;
class var_19;                /* var_19 is group varaible */
testclass var_19;
run;
endsubmit;
```

The misclassification rate that resulted from applying linear discriminant analysis (LDA) on the kernel principal components is 3%, which is significantly better than the 15% that resulted from applying LDA on principal components. (The standard errors of misclassification rate are 0.020 and 0.042, respectively.) Clearly, KPCA performs better than PCA in dimension reduction in this example, because it can capture the nonlinear relationship among attributes.

Novelty Detection

In this example, we demonstrate a case where we use KPCA for novelty detection. The data set used in this example is the Breast Cancer Wisconsin data (Original) available at the UCI Machine Learning Repository [6, 12]. This data set contains nine cytological characteristics of 683 patients who have either a benign or malignant breast tumor pattern. Each of these characteristics is an integer value. These 9 characteristics are: Clump Thickness: 1-10; Uniformity of Cell Size: (1–10), Uniformity of Cell Shape: (1–10), Marginal Adhesion: (1–10), Single Epithelial Cell Size: (1–10), Bare Nuclei: (1–10), Bland Chromatin: (1–10), Normal Nucleoli: (1–10), and Mitoses: (1–10). In this example, the same data preprocessing steps are performed as in [7]: First subjects who have missing attributes are removed from the study. Next, the attributes of the remaining subjects are scaled to have unit variance. To prevent the potential numerical errors that are caused by discrete integer values, a uniform noise from the interval [0.05, 0.05] is added to each attribute. We train the KPCA-based novelty detector on 200 benign samples and then test it on 244 benign and 239 malignant samples. As a novelty measure, we calculate the reconstruction error in the feature space,

$$E(\tilde{\phi}) = (\tilde{\phi} \cdot \tilde{\phi}) - (W\tilde{\phi} \cdot W\tilde{\phi}) \tag{22}$$

Here $\tilde{\phi}$ is a centered vector in feature space. Denote q as the number of principal components. The matrix W contains q row vectors v^l where v is defined in (4) . Index l $(l = 1, 2, \dots, q)$ denotes the lth eigenvector. The reconstruction error E can be expressed in terms of the kernel function of x in the input space. More computation details can be found in [7].

Applying a KPCA novelty detector involves choosing a threshold for the reconstruction error in advance. For this data set, we use an error threshold of 0.0834. The following SAS/IML code trains the KPCA model and calculates the reconstruction errors. Here wdbc_train and wdbc_test are the names of the training data set and the test data set, respectively.

```
/**************Novelty detection example************************/
```

```
%let url=https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/
breast-cancer-wisconsin.data;

filename cc url "&url";


data wdbc;
infile cc delimiter=',';
input var1-var10 group;
obsid = _n_;
run;

/* delete observations with missing value */
data wdbc;
set wdbc;
if cmiss(of _all_) then delete;
run;


/* Split dataset to train (200 all benign) and test(483 benign and malignant) */
proc surveyselect data=wdbc (where=(group=2))
      method=srs n=200
      seed=100 out=wdbc_train;
run;

proc sql;
create table wdbc_test as
select * from wdbc
where obsid not in (select obsid from wdbc_train);
quit;


proc iml;

varNames = ("var2":"var10");
use wdbc_train;
   read all var varNames into wdbc_train[c=varName];
   read all var "group" into class_train;
close;

use wdbc_test;
   read all var varNames into wdbc_test[c=varName];
   read all var "group" into class_test;
close;

 /*scale data*/
std=std(wdbc_train//wdbc_test);
wdbc_train=wdbc_train/std;
```

```
wdbc_test=wdbc_test/std;

/* add uniform noise */
wdbc_train= wdbc_train+randfun({200, 9}, "Uniform", -0.05, 0.05);
wdbc_test= wdbc_test+randfun({483, 9}, "Uniform", -0.05, 0.05);


/* Kernel PCA */
opt = {1E-15,
       2,         /* Gaussian kernel */
       2,         /* kernel parameter=2 */
       1,         /* exact method for PCA */
       190};        /* project on 190 PCs */
call KPCATrain(eigenVal, eigenVec, centroids, RowMeans, score_train,
               wdbc_train, opt);

eigenVec=eigenVec[,1:190];

opt_score= {190, 2, 2};

/* Distance function */
start sqdist(a,b);
    c=a‘; d=b‘;
    aa = (c#c)[+,]; bb = (d#d)[+,]; ab = c‘*d;
    d = abs(repeat(aa‘,1,ncol(bb))+ repeat(bb,ncol(aa),1) - 2*ab);
    return(d);
finish;

/* Reconstruction error function */
start recerr(eigenVec, test, train, RowMeans, opt);
    sigma=opt[3];
    K_sum=mean(Rowmeans);
    score_test=KPCAScore(eigenVec, train, RowMeans, test, opt);
    dist_1=sqdist(test,test);
    K1=exp(-dist_1#(1/(2*sigma##2)));
    dist_2=sqdist(test,train);
    K2=exp(-dist_2#(1/(2*sigma##2)));
    K2_sum=K2[ ,+];
    err=K1-2*K2_sum/nrow(train)+K_sum-score_test*score_test‘;
    return(err);
finish;

ntest=nrow(wdbc_test);
error=j(ntest, 1);

/* Calculate reconstruction error */
do i = 1 to ntest;
    test=wdbc_test[i,];
    err=recerr(eigenVec, test, wdbc_train, RowMeans, opt_score);
    error[i,]=err;
end;


create error from error [colname="error"];
append from error;
close error;

create group_test from class_test [colname="group_t"];
append from class_test;
close group_test;


submit;

/* Assign points with reconstruction error larger than threshold to malignant */
data error;
    set error;
    if error>=0.0834 then group_p=4;
    else group_p=2;
run;

data compare;
   merge error group_test;
run;

/* Calculate TP, FN, FP */
 proc freq data=compare;
     table group_p*group_t/ out=CellCounts;
 run;

endsubmit;

quit;
```

The novelty detector achieves an F1 score of 0.9726. We also applied another commonly used novelty detector method, Support Vector Data Description (SVDD), on this data set. SVDD achieved an F1 score of 0.9530. We can see that the two methods have comparable performances. As stated in [7], for multi-form variance data, the novelty detector based on KPCA can in fact achieve a tighter decision boundary than SVDD can achieve.

Fast KPCA

In this example, we compare the performance of fast KPCA, which is based on a low-rank approximation method with regular KPCA. The letter recognition data set that is used in this example is also from UCI Machine Learning Repository [6]. The goal is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The character images were based on 20 different fonts, and each letter within these 20 fonts was randomly distorted to produce a file of 20000 unique instances. Each instance was converted into 16 numerical attributes, which were then scaled to fit into a range of integer values from 0 through 15 [10]. In this example, we randomly choose a training set of size 16,000 and use the remaining 4,000 instances as a test set. Our objective is to compare the performance of exact KPCA and fast KPCA. Because this is a big data set, we expect fast KPCA to be much more efficient in time and space than the exact method, while still achieving a comparable performance. To use fast KPCA, we select 200 centroids and apply K-means++ [11] to choose the seeds in the *k*-means clustering method. After finding the score values for the training and test sets, we input them in multi-label linear discriminant analysis (LDA) for classification. The following IML procedure code applies the exact and fast KPCA methods on the letter recognition data. Next, PROC DISCRIM is applied on the data that are projected on each of these reduced dimensions. Here recognition_train and recognition_test are the names of the training data set and the test data set, respectively.

```
/***********Letter Recognition example*************************/

%let url=https://archive.ics.uci.edu/ml/machine-learning-databases/letter-recognition/
letter-recognition.data;

filename cc url "&url";

data letter;
infile cc delimiter=',';
input capital $ var1-var16;
obsid = _n_;
run;


/* Split dataset to train (16000) and test(4000) */
proc surveyselect data=letter
      method=srs n=16000
      seed=100 out=letter_train;
run;

proc sql;
create table letter_test as
select * from letter
where obsid not in (select obsid from letter_train);
quit;

proc iml;
varNames = ("var1":"var16");
use letter_train;
   read all var varNames into X_train[c=varName];
   read all var "capital" into group_train;
close;
```

```
use letter_test;
   read all var varNames into X_test[c=varName];
   read all var "capital" into group_test;
close;


/* Kernel PCA exact training step*/
opt = {.,         /* eigenvalue cutoff (default) */
       2,         /* Gaussian kernel */
       7.071,         /* kernel parameter=7.071 */
       1,         /* exact method for KPCA */
       200};        /* project on 200 PCs */


%let _sdtm=%sysfunc(datetime());
call KPCATrain(eigenVal, eigenVec, centroids, RowMeans, score_train,
               X_train, opt);
%let _edtm=%sysfunc(datetime());
%let _runtm=%sysfunc(putn(&_edtm-&_sdtm,12.4));
print "it took &_runtm seconds to run exact kpca";


/* Kernel PCA scoring step*/
opt= {200, 2, 7.071};
score_test=KPCAScore(eigenVec, X_train, RowMeans, X_test, opt);

/* save score values to SAS dataset */
valname="group";

create score_train from score_train;
append from score_train;
close score_train;

create group_train from group_train [colname=valname];
append from group_train;
close group_train;


create score_test from score_test;
append from score_test;
close score_test;

create group_test from group_test [colname=valname];
append from group_test;
close group_test;

submit;
```

```
data score_train;
   merge score_train group_train;
run;

data score_test;
   merge score_test group_test;
run;

/* multi-label linear discriminant analysis */
proc discrim data=score_train method=normal pool=yes short
testdata=score_test;
class group;
testclass group;
run;
endsubmit;


/* Kernel PCA traning step with low-rank approximation*/
opt = {.,         /* eigenvalue cutoff (default) */
       2,         /* Gaussian kernel */
       7.071,         /* kernel parameter=7.071 */
       0,         /* low-rank approximation for KPCA */
       190};        /* project on 190 PCs */


optCluster={2,      /* use kmeans++ to select initial centroids */
            190,   /* use 190 centroids in low-rank approximation */
      .,
      .,
      .,
      .};

%let _sdtm=%sysfunc(datetime());
call KPCATrain(eigenVal, eigenVec, centroids, RowMeans, score_train,
               X_train, opt, optCluster);
%let _edtm=%sysfunc(datetime());
%let _runtm=%sysfunc(putn(&_edtm-&_sdtm,12.4));
print "it took &_runtm seconds to run low-rank kpca";


/*Kernel PCA scoring step */
opt= {190, 2, 7.071};
score_test=KPCAScore(eigenVec, X_train, RowMeans, X_test, opt);

/* save score values to SAS dataset */

create score_train from score_train;
append from score_train;
```

```
close score_train;

create score_test from score_test;
append from score_test;
close score_test;

submit;
data score_train;
   merge score_train group_train;
run;

data score_test;
   merge score_test group_test;
run;

/* multi-label linear discriminant analysis */
proc discrim data=score_train method=normal pool=yes short
testdata=score_test;
class group;
testclass group;
run;
endsubmit;
quit;
```

We observe that the classification errors that were obtained by fast KPCA and exact KPCA are close: 0.1629 for fast KPCA and 0.1596 for exact KPCA. However, the run time of fast KPCA is only 3 seconds compared to 628 seconds for exact KPCA. This example demonstrates the efficiency of fast KPCA in significantly reducing the run time while not compromising very much on the quality of the principal components it generates. In this example, we also apply PCA on the data and extract all 16 components to use in multi-label LDA. We find that even when all components are used, PCA achieves a misclassification rate of 0.3011, which is much higher than that of KPCA.

Conclusions

In this paper, we reviewed the KPCA method (which is a nonlinear form of the PCA) and a low-rank approximation method for KPCA. Then we focused on the application of KPCA in different areas. In the data visualization and dimensionality reduction applications, we compared KPCA with PCA, and we found that KPCA achieves much better performance than PCA because KPCA can capture higher-order statistics that are present data sets and it can generate nonlinear subspaces for better feature extraction. In the novelty detection example, we demonstrated that KPCA achieves performance comparable to SVDD, which is an effective and well-studied novelty detection method. Finally, in the last example we demonstrated the effectiveness of using the fast KPCA method in reducing the run time while maintaining the quality of principal components.

Acknowledgments

Authors would like to thank Anne Baxter, Principal Technical Editor at SAS, for her thorough and insightful review and edit of this manuscript.

References

[1] Baker, C.T., 1977. The numerical treatment of integral equations.

[2] Schölkopf, B., Smola, A. and Müller, K.R., 1998. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5), pp.1299-1319.

[3] Zhang, K., Tsang, I.W. and Kwok, J.T., 2008, July. Improved Nystrm low-rank approximation and error analysis. In *Proceedings of the 25th international conference on Machine learning* (pp. 1232-1239). ACM.

[4] Alcock, R.J. and Manolopoulos, Y., 1999, August. Time-series similarity queries employing a feature-based approach. In *7th Hellenic conference on informatics* (pp. 27-29).

[5] Pham, D.T. and Chan, A.B., 1998. Control chart pattern recognition using a new type of self-organizing neural network. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 212(2), pp.115-127.

[6] Bache, K. and Lichman, M., 2013. UCI machine learning repository, 2013. https://archive.ics.uci.edu/ml/index.php.

[7] Hoffmann, H., 2007. Kernel PCA for novelty detection. *Pattern recognition*, 40(3), pp.863-874.

[8] Williams, C. and Seeger, M., 2000. The effect of the input density distribution on kernel-based classifiers. In *Proceedings of the 17^{th} International Conference on Machine Learning*, pp. 11591166.

[9] SAS Institute Inc. 2018. SAS/IML 15.1 User's Guide https://go.documentation.sas.com/?docsetId=imlug&docsetTarget=imlug_langref_sect053.htm&docsetVersion=15.1&locale=en.

[10] Frey, P.W. and Slate, D.J., 1991. Letter recognition using Holland-style adaptive classifiers. *Machine learning*, 6(2), pp.161-182.

[11] Arthur, D. and Vassilvitskii, S., 2007, January. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms* (pp. 1027-1035). Society for Industrial and Applied Mathematics.

[12] Mangasarian, O.L. and Wolberg, W.H., 1990. *Cancer diagnosis via linear programming*. University of Wisconsin-Madison Department of Computer Sciences.

To contact your local SAS office, please visit sas.com/offices