Artificial Intelligence Review (2023) 56:5445–5477
https://doi.org/10.1007/s10462-022-10297-z

Tutorial on PCA and approximate PCA and approximate kernel PCA

Sanparith Marukatat[1]

Published online: 31 October 2022

Abstract
Principal Component Analysis (PCA) is one of the most widely used data analysis methods in machine learning and AI. This manuscript focuses on the mathematical foundation of classical PCA and its application to a small-sample-size scenario and a large dataset in a high-dimensional space scenario. In particular, we discuss a simple method that can be used to approximate PCA in the latter case. This method can also help approximate kernel PCA or kernel PCA (KPCA) for a large-scale dataset. We hope this manuscript will give readers a solid foundation on PCA, approximate PCA, and approximate KPCA.

Keywords PCA · Eigen-decomposition · Approximate PCA · Kernel PCA

1 Introduction

What would be the description if you were asked to describe a person sitting next to you? Male or female, wear glasses or not, big mouth, small nose, long hair, tall, overweight, BMI score, etc. These descriptions can be put together in the form of a "*vector*". In this manuscript, we consider "*column vector*", a matrix of one column and several rows, each for a specific description. With ten descriptions, we have a vector of 10 rows. We say that this vector lives in a 10-dimensional space. With 100 descriptions, we have a vector of 100 rows that lives in a 100-dimensional space.

A vector is a mathematical object composed of "length" or "magnitude" and "direction". Nonetheless, in this paper, we will use the term "*vector*" and "*point*" interchangeably since, given a point, we can always assign the vector from the origin point to that given point. Note that it is common to work in a very high-dimensional space in AI, machine learning, and data science. Some intuitions we have gained from our daily life experience in two or three dimensions may not be correct in this very high-dimensional space. Readers interested in properties of high-dimensional data may refer to Blum et al. (2017) for more detail.

Suppose we have prepared 100 descriptions or *features* to describe each person. Suppose that these features were all numerical, e.g., the height, the weight, the BMI score, etc. Suppose

✉ Sanparith Marukatat
sanparith.marukatat@nectec.or.th

[1] AI Research Group, NECTEC, Pathumthani, Thailand

that we sat at a coffee shop and tried to describe each client using our 100 features. For each client, we obtain a *feature vector* that contains all descriptions assigned to him/her. Given a collection of feature vectors, what shall we see?

We will see that the observed height values do not go from 0 to infinity. They will be bounded within some range, for example, between 150 to 180 cm. The observed weights and BMI scores are bounded as well. The observed feature vectors will not be scattered all over the whole space. They are clustered around some area, or *subspace*, within the big multi-dimensional space. With a deeper inspection, you may find correlations between features such as height, weight, and BMI score. Therefore, it is tempting to combine these features into a new one. The question is then how to combine these features and measure if the combination is good.

Principal Component Analysis (PCA) provides such a combination method. Indeed, PCA relies on *linear combination* of these features to construct *principal subspace* that is the main subspace on which most of the feature vectors lie. PCA uses *variance* as a measure of the information content of the subspace. The quality of the selected subspace can be measured by comparing the variance of data within it to the total variance of the whole dataset. These are two keys idea of PCA. In practice, however, one may run into practical problems such as having too few observations compared to the data dimension or having too much computational cost due to the high dimensionality of data and a large number of observations, etc. This paper describes some practical algorithms for handling these different scenarios. Some material presented hereafter is extended from our previously published paper (Marukatat 2016). It should be noted that this tutorial focuses mainly on the algorithmic aspect of PCA. Readers interested in its applications are invited to consult the paper (Ringnér 2008; Gewers et al. 2018; Atchade-Adelomou et al. 2022) and references therein to see different use-cases of PCA on different domains including bioinformatics and quantum computing.

In the following, Sect. 2 first describes PCA in the usual setting where the number of available data points is larger than the dimension of feature vectors. Then Sect. 3 describes PCA that is designed for small-sample-size case. This case could happen; for example, when the number of available subjects with a particular disease is limited, the researchers may extract all possible features from their blood or tissue sample. The total amount of features could be easily more significant than the total number of samples. In this case, classical PCA will not be suitable. This section describes a variation of PCA designed to handle this case. Section 4 describes another variation of PCA designed for large-scale datasets in a very high-dimension space. Indeed, current AI and data science datasets are often large with very high-dimension feature vectors. This section explains how to mitigate the computational cost of PCA in this case. The technique introduced in this section, namely dot product preserving transformation, will be used in the subsequent section on approximate kernel PCA. Section 5 discusses kernel PCA or KPCA, a non-linear extension of PCA using *kernel trick*. The classical KPCA is presented in Sect. 5.1. In this section, one will see that the main crux of KPCA is its computational cost. Two variations of KPCA designed to reduce the cost of KPCA are presented in Sects. 5.2 and 5.3 with some examples shown in Sect. 5.5.

2 Classical case: $n > d$

This section describes a classical scenario for applying PCA, namely when the number of available data points is larger than the dimension of feature vectors. In this section, we assume that the dimension of feature vectors is not very large, so all calculations can be done on a modest computer.

2.1 Notation

In the following, let $\mathbf{x} \in \mathbb{R}^d$ denotes a feature vector $\mathbf{x}$ in d-dimensional space with

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$$

A feature vector contains values of features that are observed on actual data. We assume that there are no missing values. We further assume that these features are all numerical.

In theory, the features should be selected adequately by domain experts. Nonetheless, in practice, we often rely on low-level features, such as the intensity of each pixel in an image, that can be computed easily. High-level features, like loops and cusps in handwritten strokes, contain more information. However, their detection depends on image processing functions that must be put in place first. These functions may contain some heuristic or programming bias. As a result, for some poorly scanned images, the detection of high-level features may not be done accurately. Low-level features, such as pixel intensities, are more robust because they can always be extracted from all input images. Even if each low-level feature may not convey a meaningful description by itself, combining lots of them with lots of data makes it possible to derive meaningful meaning using machine learning models. PCA is one method to achieve this goal.

2.2 Linear projection and dot product

The main idea of PCA is to use *variance* as a measure of information content and to identify *linear subspace* maximizing the variance. The linear subspace can be constructed from multiple *unit vectors* that are orthogonal to each other. Each unit vector represents a *projection axis*. Consider a projection axis $\mathbf{w} \in \mathbb{R}^d$, the projection of $\mathbf{x} \in \mathbb{R}^d$ onto the axis $\mathbf{w}$ is the point on this axis that is closest to $\mathbf{x}$. This projection can be computed by $||\mathbf{w}|| \cos \theta_{\mathbf{x},\mathbf{w}}$. Recall that the "*dot product*" or the "*scalar product*" or the "*inner product*" between any two vectors $\mathbf{u}$ and $\mathbf{v}$ denotes as $\langle \mathbf{u}, \mathbf{v} \rangle$ satisfies

$$\langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{u}^T\mathbf{v} = ||\mathbf{u}|| \, ||\mathbf{v}|| \cos \theta_{\mathbf{u},\mathbf{v}}. \quad (1)$$

In the following we will use $\langle \mathbf{u}, \mathbf{v} \rangle$ or $\mathbf{u}^T\mathbf{v}$ interchangeably when talking about dot product. The projection of $\mathbf{x}$ onto the unit vector axis $\mathbf{w}$ is then given by a simple dot product $\mathbf{x}^T\mathbf{w} \in \mathbb{R}$.

Note that the "projection" refers to the location *on the projection axis*. The same location can be described relative to the input space as well by multiplying it with the projection axis, i.e., $(\mathbf{x}^T\mathbf{w})\mathbf{w}$. This quantity is often referred to as the "reconstruction" of $\mathbf{x}$ from its projection on the axis $\mathbf{w}$.

2.3 Variance on projection axis and covariance matrix

Given a set of centered real values $z_1, z_2, \dots, z_n$ $z_t \in \mathbb{R}$, the variance of this dataset is

$$\text{Var} = \frac{1}{n}\sum_{t=1}^{n} z_t^2. \tag{2}$$

Given a set of feature vectors $\mathbf{x}_1, \dots, \mathbf{x}_n$, $\mathbf{x}_t \in \mathbb{R}^d$, and a projection axis $\mathbf{w}$ (that is a unit vector, i.e. $\|\mathbf{w}\| = 1$). The projections on $\mathbf{w}$ are $\mathbf{w}^T\mathbf{x}_1, \dots, \mathbf{w}^T\mathbf{x}_n$. Suppose that the feature vectors are centered around zero vectors. Thus, their linear projections will also be centered appropriately. Note that each dot product yields a simple real value, i.e., $\mathbf{w}^T\mathbf{x}_t \in \mathbb{R}$. Hence, the Eq. (2) can be used to compute the variance:

$$\text{Var}(\mathbf{w}) = \frac{1}{n}\sum_{t=1}^{n} (\mathbf{w}^T\mathbf{x}_t)^2 \tag{3}$$

$$= \frac{1}{n}\sum_{t=1}^{n} \mathbf{w}^T\mathbf{x}_t\mathbf{x}_t^T\mathbf{w} \tag{4}$$

$$= \mathbf{w}^T \underbrace{\left[\frac{1}{n}\sum_{t=1}^{n} \mathbf{x}_t\mathbf{x}_t^T\right]}_{\text{Covariance matrix}} \mathbf{w}. \tag{5}$$

The last line gives us a shortcut to compute variance on any axis $\mathbf{w}$. Indeed, instead of projecting all data onto the axis and then computing the variance, we can *pre-compute* the covariance matrix $\mathbf{C}$ first, then given any projection axis $\mathbf{w}$ the variance of the projection on $\mathbf{w}$ is given by $\mathbf{w}^T\mathbf{C}\mathbf{w}$.

2.4 PCA problem and Lagrangian

From the above calculation, we can formulate the main idea of PCA as the following constrained optimization problem:

$$\begin{aligned} \max_{\mathbf{w}} \quad & \mathbf{w}^T\mathbf{C}\mathbf{w} \\ \text{subject to} \quad & \mathbf{w}^T\mathbf{w} = 1. \end{aligned} \tag{6}$$

For standard optimization problems, we may set the partial derivatives to zero and solve the resulting equations or follow the gradient. However, the above problem has an additional constraint that must be considered. To this end, the standard approach is to use the *Lagrangian method* (see (Dimitri 1996) for more detail on the Lagrangian method).

In short, to solve the following constrained optimization problem:

$$\begin{aligned} &\max_{\mathbf{x}} && f(\mathbf{x}) \\ &\text{subject to } g(\mathbf{x}) = 0, \end{aligned} \tag{7}$$

the following *Lagrangian function* is considered:

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda g(\mathbf{x}), \tag{8}$$

with λ the *Lagrangian coefficient* describing the important of the constraint. Then, given the solution $(\mathbf{x}^*, \lambda^*)$ optimizing the function L, $\mathbf{x}^*$ will be the solution of the initial problem.

For the PCA problem, its Lagrangian is given by

$$L(\mathbf{w}, \lambda) = \mathbf{w}^T\mathbf{C}\mathbf{w} - \lambda\left(\mathbf{w}^T\mathbf{w} - 1\right). \tag{9}$$

Then, if we compute the partial derivative of L with respect to $\mathbf{w}$ and set it to zero, we obtain:

$$0 = \frac{\partial L}{\partial \mathbf{w}} \tag{10}$$

$$= 2\mathbf{C}\mathbf{w} - 2\lambda\mathbf{w} \tag{11}$$

$$\mathbf{C}\mathbf{w} = \lambda\mathbf{w}. \tag{12}$$

The above derivation uses *matrix calculus*. Readers unfamiliar with this calculation could consult reference (Petersen and Pedersen 2012) for more detail.

The Eq. (10) means that the axis $\mathbf{w}$ maximizing variance must be an *eigenvector* of $\mathbf{C}$, and λ is its corresponding *eigenvalue*. The eigenvectors and eigenvalues are, in fact, mathematical objects that have been studied for a very long time. PCA represents another application of this well-founded subject. An example of a commonly used method to extract eigenvector and eigenvalue pair, or eigen-decomposition algorithm called the *Jacobi method* is shown in Section A.

2.5 Eigenvalue, variance and axis selection

From Sect. 2.4, we have seen that the projection axis maximizing variance must be an eigenvector of the covariance matrix $\mathbf{C}$. However, as $\mathbf{C}$ is a $d \times d$ matrix, it has d eigenvectors; which one should be selected? To answer this question, we can observed that

$$\mathbf{C}\mathbf{v}_i = \lambda_i\mathbf{v}_i \tag{13}$$

$$\mathbf{v}_i^T\mathbf{C}\mathbf{v}_i = \lambda_i\mathbf{v}_i^T\mathbf{v}_i \tag{14}$$

$$\text{Var}\,(\mathbf{v}_i) = \lambda_i. \tag{15}$$

In other words, the eigenvalue corresponds to the variance of the data projected onto its corresponding eigenvector.

Given eigenvalues $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_d$ with $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d$ the corresponding eigenvectors. From the discussion above, we know that if we select only one projection axis, then $\mathbf{v}_1$ should be the best choice since it corresponds to the largest eigenvalue, thus the largest variance of projected data.

Note that from these eigenvalues we can easily compute the total variance, i.e. $\sum_{i=1}^{d} \lambda_i$. This total variance represents the entire information content of the dataset. By comparing λ_1 to the total variance, we can judge if a single projection axis $\mathbf{v}_1$ is enough to retain the most information or not. In practice, a single projection axis is not enough. Fortunately, as the eigenvectors are already orthogonal, it is possible to select the next "largest" eigenvectors to form the basis of the *principal subspace*. The next question is how many axes should be selected. There are three approaches to this question:

1. User gives the desired number of axis m.
2. If we assume that p portion of the information content was, in fact, noise, then m should be the smallest number such that $\sum_{i=1}^{m} \lambda_i \geq p \sum_{i=1}^{d} \lambda_i$.
3. We can also assume that all axis with variance smaller than a pre-defined threshold are noise and should be discarded. Thus, we select all axis that $\lambda_i \geq \varepsilon \lambda_1$.

Algorithm 1 The PCA algorithm

```
1: procedure PCA($\mathbf{x}_1, ..., \mathbf{x}_n$ with $\mathbf{x}_t \in \mathbb{R}^d$)    ▷ Return principal components from the given dataset.
2:     Center the data: $\mathbf{x}_t = \mathbf{x}_t - \mu$ with $\mu$ the mean vector.
3:     Construct the covariance matrix $\mathbf{C} = \frac{1}{n} \sum_t \mathbf{x}_t \mathbf{x}_t^T$.
4:     Eigen-decompose $\mathbf{C}$ and let $\lambda_1 \geq ... \geq \lambda_d$ and $\mathbf{v}_1, ..., \mathbf{v}_d$ be its eigenvalues and eigenvectors.
5:     Select $m$ as discussed above.
6:     return first $m$ eigenvectors and eigenvalues.
7: end procedure
```

The PCA algorithm is summarized in Algorithm 1.

Given principal axis $\mathbf{v}_1, \ldots, \mathbf{v}_m$, each new data point $\mathbf{x}$ can be represented within the principal subspace as:

$$\begin{bmatrix} \mathbf{x}^T \mathbf{v}_1 \\ \mathbf{x}^T \mathbf{v}_2 \\ \vdots \\ \mathbf{x}^T \mathbf{v}_m \end{bmatrix} \in \mathbb{R}^m, \tag{16}$$

and the reconstruction of $\mathbf{x}$ from its projection on the principal subspace is given by:

Table 1 Summary of datasets used in this section

Dataset	Dimension (d)	Number of samples (n)
Diabetes	10	442
Wisconsin breast cancer	30	569
Iris	50	150
MNIST	784	60,000
CIFAR	3072	50,000

$$\hat{\mathbf{x}} = \sum_{i=1}^{m} (\mathbf{x}^T \mathbf{v}_i)\mathbf{v}_i. \tag{17}$$

It is pretty straightforward to prove that the selected principal subspace also minimizes the reconstruction error $\sum_t \|\mathbf{x}_t - \hat{\mathbf{x}}_t\|^2$ and that the reconstruction error is indeed the sum of all eigenvalues outside the principal subspace.

2.6 Examples

This section shows examples of the application of PCA to various datasets, namely Diabetes[1], Wisconsin breast cancer[2], Iris[3], MNIST[4], and CIFAR-10[5]. Table 1 summarizes the detail of these datasets Diabetes, Wisconsin breast cancer, and Iris contain multi-dimensional vectors, whereas MNIST and CIFAR contain grayscale images in 28×28 pixels and RGB images in 32×32 pixels, respectively. These images were reorganized into vectors in $28 \times 28 = 784$ dimensions and $32 \times 32 \times 3 = 3072$ dimensions respectively. Z-score normalization was applied to all datasets. Thus all data were properly centered.

Figure 1, from the top to the bottom row, shows results from five datasets: Diabetes, Wisconsin breast cancer, Iris, MNIST, and CIFAR, respectively. The left column shows the eigenvalues plot from the four datasets. The second column shows the cumulated proportion of variances or eigenvalues. The two right columns show the projection of the five datasets onto the two first eigenvectors, and onto the first and the last eigenvectors, respectively. In both cases, the projection onto the first eigenvector corresponds to the abscissa. One can observe that the variances on the horizontal axis are higher than that on the vertical axis since the eigenvectors are sorted in decreasing order of their eigenvalues, and each eigenvalue corresponds to variance on the corresponding eigenvector. For the MNIST dataset, the rightmost plot appears to be quite strange. This is due to an outlier in the dataset shown in Fig. 2a. Indeed, this image has a small white dot on the left side. This dot disappears in the reconstruction using 50 first principal components (Fig. 2b). However, this reconstruction is blurred. Using more principal components results in a sharper image but with some noise. Hence, it is possible to denoise the input data using the PCA method.

Table 2 depicted the number of axes needed to retain 90%, 95%, and 99% of the total information compared with the dimension of these datasets. One can observe that for some

[1] https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html.

[2] https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html.

[3] https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html.

[4] http://yann.lecun.com/exdb/mnist/.

[5] http://www.cs.toronto.edu/~kriz/cifar.html.

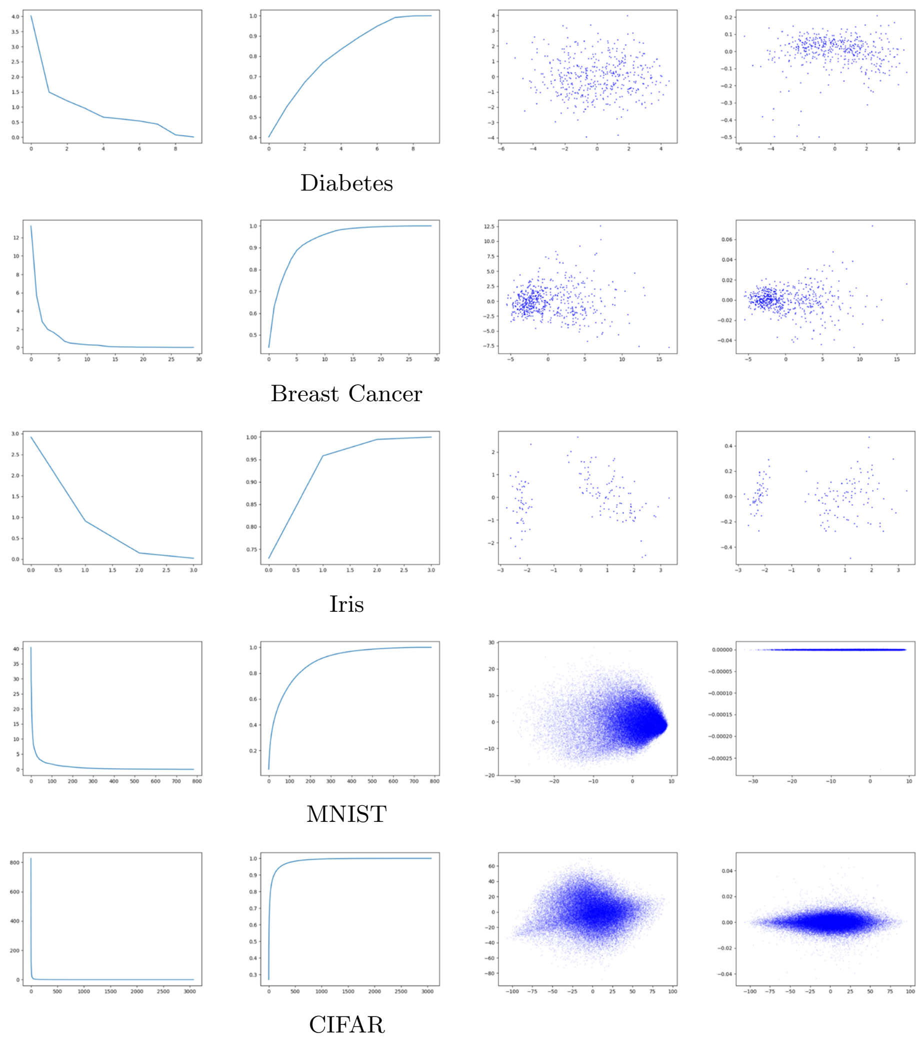

Fig. 1 Plot of eigenvalues (left) and the projection of different datasets onto the two first eigenvectors (middle) as well as the projection onto the first and the last eigenvectors (right)

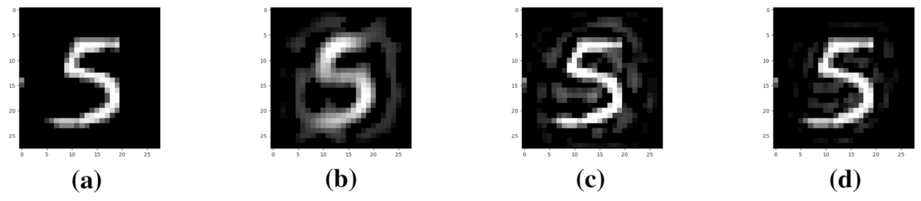

Fig. 2 An outlier in MNIST dataset **a** and its reconstruction using 50 (resp. 100, 200) first principal components in **b** (resp. in **c**, and **d**)

Table 2 The number of axis needed to retain 90%, 95%, and 99% of total information

Datasets	Number of axis to retain information			Dimension
	90%	95%	99%	
Diabetes	7	8	8	10
Wisconsin breast cancer	7	10	17	30
Iris	2	2	3	50
MNIST	236	331	543	784
CIFAR	103	221	662	3072

datasets, e.g., Iris and CIFAR, the number of the required axis to retain 90% of information is only a small fraction of the total dimension. Other datasets, like Diabetes, requires a larger number of axis.

2.7 Discussion

The principal subspace obtained from PCA describes the area in the whole space on which most of the data lies. This knowledge is helpful. For example, consider the problem of face verification. We are given a pair of face images and have to decide if both images come from the same person or not. If we compare both images directly, in *pixel space*, so to speak, the observed difference may be due to the background area in the images, which does not reflect the identity of the person. By working on the principal subspace of the face images, we can be sure that every comparison is meaningful. This was the idea behind *Eigenface* that was the foundation of early face recognition methods (Sirovich and Kirby 1987; Turk and Pentland 1991).

Today's face verification methods are based on deep features extracted from deep learning models (Wang and Deng 2018). This approach is possible now because we have more data to train deep neural networks. In the early period of face recognition, the classical face dataset called *ORL* (orl database) contained only 40 subjects with ten images per subject. The size of each image was 92×112 pixels, hence 10,304 pixels in total. This dataset is tiny compared to today's datasets. Besides, the main problem, in this case, is that we have more dimensions than the number of observations ($d > n$). Consequently, the covariance matrix will most likely be *ill conditioned* and cannot be eigen-decomposed. The Eigenface method resolves this problem via *dot matrix* instead of the covariance matrix. This procedure will be described in the next section.

3 Small-sample-size case: $n \leq d$

This section describes the PCA method designed for the case where the number of data points (n) is smaller than the dimension of feature vectors (d). Even if the method described in this section originated from image application, it could also be used with general feature vectors when $n \leq d$. This scenario appears, for example, in bioinformatics (Ringnér 2008). In Ringnér (2008), the author studied 8,534 probes on the microarrays with expression measurements extracted from 105 samples. In other words, the authors consider a dataset of 105 feature vectors in 8,534 dimensions. This dataset can be processed using the PCA algorithm described in this section.

In the following, Sects. 3.1 and 3.2 discuss the dot matrix, its relation to the covariance matrix, and the conversion of eigenvectors obtained from both matrices. The use of the dot matrix is the foundation of the small-sample-size PCA method described in Sect. 3.3.

3.1 Covariance matrix and Dot matrix

Given feature vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ with $\mathbf{x}_t \in \mathbb{R}^d$. We assume that these vectors are centered around zero. We can construct a *data matrix* $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ of size $d \times n$. The column i of $\mathbf{X}$ corresponds to the feature vector $\mathbf{x}_i$. Using this data matrix, the covariance matrix can be rewritten as

$$\mathbf{C} = \frac{1}{n}\sum_{t=1}^{n} \mathbf{x}_t \mathbf{x}_t^T = \frac{1}{n}\mathbf{X}\mathbf{X}^T. \tag{18}$$

We often use $\mathbf{X}\mathbf{X}^T$ to denote the covariance matrix since maximizing $\mathbf{w}^T\mathbf{C}\mathbf{w}$ or $\mathbf{w}^T\mathbf{X}\mathbf{X}^T\mathbf{w}$ will result in the same eigenvectors (if we constraint these eigenvectors to be unit vectors).

The dot matrix is defined as $\mathbf{X}^T\mathbf{X}$. It is easy to see that the element i, j of this matrix is indeed the dot product between example i and j, i.e. $\mathbf{x}_i^T\mathbf{x}_j$.

3.2 Eigenvector conversions

Suppose that λ and $\mathbf{v}$ are eigenvalue and corresponding eigenvector of dot matrix, then we have

$$\mathbf{X}^T\mathbf{X}\mathbf{v} = \lambda\mathbf{v} \tag{19}$$

$$(\mathbf{X}\mathbf{X}^T)(\mathbf{X}\mathbf{v}) = \lambda(\mathbf{X}\mathbf{v}). \tag{20}$$

Hence $\mathbf{X}\mathbf{v}$ will be an eigenvector of the covariance matrix under the same eigenvalue λ. This relation allows us to convert the dot matrix's eigenvectors into the covariance matrix's eigenvectors. After the conversion, the newly obtained eigenvector should be normalized as usual. Note that we have

$$\mathbf{v^T X^T X v} = \lambda. \tag{21}$$

Hence the normalized version of the eigenvector is simply $(1/\sqrt{\lambda})\mathbf{X}\mathbf{v}$

On the other hand, it is also possible to convert eigenvectors of the covariance matrix into eigenvectors of the dot matrix. Indeed, if λ and $\mathbf{u}$ are eigenvalue and corresponding eigenvector of dot matrix, then we have

$$\mathbf{X}\mathbf{X}^T\mathbf{u} = \lambda\mathbf{u} \tag{22}$$

$$(\mathbf{X}^T\mathbf{X})(\mathbf{X}^T\mathbf{u}) = \lambda(\mathbf{X}^T\mathbf{u}). \tag{23}$$

Hence $\mathbf{X}^T\mathbf{u}$ will be an eigenvector of the dot matrix under the same eigenvalue λ. To normalize this eigenvector, we have

$$\mathbf{u}^T\mathbf{X}\mathbf{X}^T\mathbf{u} = \lambda. \tag{24}$$

Hence the normalized version of the eigenvector is $(1/\sqrt{\lambda})\mathbf{X}^T\mathbf{u}$

Algorithm 2 The small-sample-size PCA algorithm

```
1: procedure SSS-PCA($\mathbf{x}_1, ..., \mathbf{x}_n$ with $\mathbf{x}_t \in \mathbb{R}^d$ with $d > n$)    ▷ Return principal components from the given dataset.
2:     Center the data: $\mathbf{x}_t = \mathbf{x}_t - \mu$ with $\mu$ the mean vector.
3:     Construct the dot matrix $\mathbf{X}^T\mathbf{X}$.
4:     Eigen-decompose dot matrix and let $\lambda_1 \geq ... \geq \lambda_d$ and $\mathbf{v}_1, ..., \mathbf{v}_d$ be its eigenvalues and eigenvectors.
5:     Perform conversion: we obtain $(1/\sqrt{\lambda_1})\mathbf{X}\mathbf{v}_1, ..., (1/\sqrt{\lambda_d})\mathbf{X}\mathbf{v}_d$ eigenvectors of the covariance matrix.
6:     Select $m$ as discussed above.
7:     return first $m$ eigenvectors and eigenvalues.
8: end procedure
```

3.3 PCA from dot matrix

From the above discussion, the small-sample-size PCA can be done by eigen-decompose the dot matrix first, then converting the result into eigenvectors of the covariance matrix. The whole procedure is summarized in Algorithm 2. Note that the size of the covariance matrix is $d \times d$, and the size of the dot matrix is $n \times n$. From the computational point of view, the rule of thumb is to eigen-decompose a smaller matrix. If $d \leq n$, then we should eigen-decompose the dot matrix, if $d > n$ we eigen-decompose the covariance matrix.

3.4 Examples

To illustrate the application of PCA in this case ($n \leq d$), ORL dataset[6] is considered along with CIFAR-10 dataset. The ORL dataset contains 400 images of 92×112 pixels. These images were vectorized into 10,304-dimensional vectors. For CIFAR-10, a dataset of 1,000 examples is constructed for classes 0, 1, and 2. In both cases, the dimension of feature vectors is much higher than the number of examples. Thus, the size of the covariance matrix will be larger than that of the dot matrix.

Figure 3 shows results from four datasets namely 'ORL' and 'CIFAR' with class 0, 1, and 2 respectively. The left column shows the eigenvalues plot from the four datasets. The second column shows the cumulative proportion of variance in the principal subspace. The two right columns show a projection of the four datasets onto the two first eigenvectors, and onto the first and the last eigenvectors, respectively. In both cases, the projection onto the first eigenvector corresponds to the abscissa. These 2D plots show the same trend as in the standard PCA case. Indeed, the variance on the horizontal axis is higher than that on the vertical axis since the eigenvectors are sorted in decreasing order of their eigenvalues

[6] https://www.kaggle.com/datasets/kasikrit/att-database-of-faces.

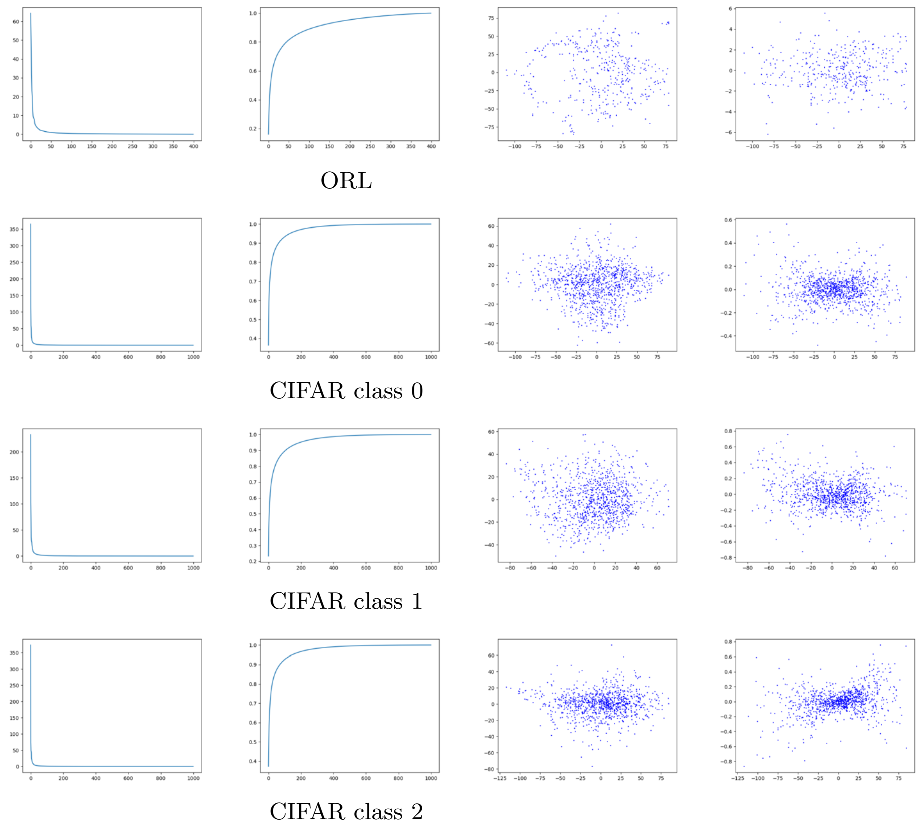

Fig. 3 Plot of eigenvalues (left) and the projection of different datasets onto the two first eigenvectors (middle) as well as the projection onto the first and the last eigenvectors (right) from ORL dataset (top) and from CIFAR dataset class 0, 1, and 2

corresponding to variance on the axis. Moreover, it is interesting to see that most variance is due to the first few principal axes.

4 Large-scale dataset in high-dimensional space: large *n* and large *d*

As time goes by, the size of the face dataset increases from a few hundred images to more than ten thousand images. Consequently, both covariance matrix and dot matrix will be huge and become difficult to eigen-decompose. This section describes how to deal with this problem, starting from a simple heuristic designed for face images (Sect. 4.1), then we discuss why it works (Sect. 4.2) before giving a general method to handle this case in Sect. 4.4.

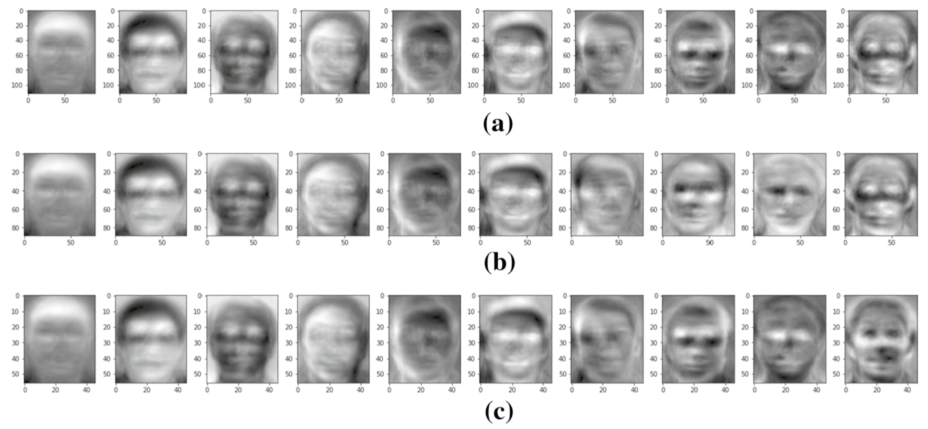

Fig. 4 10 largest Eigenfaces from ORL dataset using 3 resolutions namely the original size 96×112 (**a**), 77×89 (**b**), and 46×56 (**c**)

4.1 Motivating idea from face images

Consider the three images in Fig. 4. They were principal axis or *Eigenfaces* obtained from ORL dataset at 3 resolutions namely 96×112 in Fig. 4a, 77×89 in Fig. 4b, and 46×56 in Fig. 4c. Each face image was resized into the desired resolution; then, their pixels were rearranged to form a feature vector. We can obtain the face image corresponding to each Eigenface by inverting this pixel arrangement.

One can easily see the similarity between the three sets of Eigenfaces. It is worth noting that the negative of an eigenvector is also an eigenvector with the same variance. Thus, given two Eigenfaces, with one that appears to be the inverted color of another, they are, in fact, colinear and can be converted into the same Eigenface. An interesting question is why eigenvectors obtained from different resolutions look so similar.

Recall that an eigenvector $\mathbf{v}_i$ of the dot matrix can be converted into an eigenvector of the covariance matrix by $\mathbf{X}\mathbf{v}_i$. Note that

$$\mathbf{X}\mathbf{v}_i = \sum_{t=1}^{n} v_{it}\mathbf{x}_t. \tag{25}$$

Hence, the coordinates of an eigenvector of the dot matrix can be interpreted as *relative importance of different examples*. This is why Google's Pagerank uses the coordinate of the leading eigenvector of the Internet to describe the importance of each web page (Brin and Page 1998).

In general, resizing does not change the face image's global *aspect*. Thus, the relative importance between images should be affected slightly. This could explain why resizing produces similar eigenvectors.

Indeed, from the above discussion, the PCA can be done by approximating the eigenstructure, i.e., eigenvalues and eigenvectors, of the original dot matrix by eigenstructure from reduced-size images. The whole PCA process is summarized in Algorithm 3.

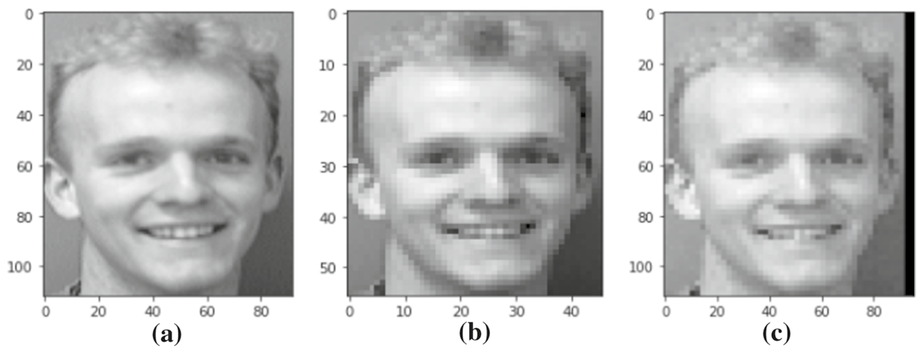

Fig. 5 Original image from ORL dataset (**a**), its downscale + upsample version (**b**), its pixel-grouping version (**c**)

Algorithm 3 The downscaled-image PCA algorithm

```
1: procedure DOWNSCALED-IMAGE-PCA($\mathbf{x}_1, ..., \mathbf{x}_n$ with $\mathbf{x}_t \in \mathbb{R}^d$ with large $d$ and
   large $n$)        ▷ Return approximate principal components from the given dataset.
2:     Center the data
3:     $\tilde{\mathbf{x}} = f(\mathbf{x}) \in \mathbb{R}^q$ with $f$ image resizing from $d$ pixels to $q$ pixels
        $q \ll d, n$ and let $\tilde{\mathbf{X}} = [\tilde{\mathbf{x}}_1, ..., \tilde{\mathbf{x}}_n]$ be the new data matrix of size $q \times n$
4:     Eigenstructure of the new covariance matrix $\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T$ : $\lambda_1 \geq ... \geq \lambda_q$ and
   $\mathbf{u}_1, ..., \mathbf{u}_q, \mathbf{u}_i \in \mathbb{R}^q$
5:     Eigenstructure of new dot matrix : $\lambda_i$ and $(1/\sqrt{\lambda_i})\tilde{\mathbf{X}}^T\mathbf{u}_i \in \mathbb{R}^n$.
6:     Approximate eigenstructure of dot matrix : $\lambda_i$ and $(1/\sqrt{\lambda_i})\tilde{\mathbf{X}}^T\mathbf{u}_i \in \mathbb{R}^n$.
7:     Approximate eigenstructure of covariance matrix : : $\lambda_i$ and $(1/\lambda_i)\mathbf{X}\tilde{\mathbf{X}}^T\mathbf{u}_i \in \mathbb{R}^d$.
8:     Select $m$ as discussed above.
9:     return first $m$ eigenvectors and eigenvalues.
10: end procedure
```

4.2 Matrix perturbation

From the previous Section, we may wonder what kind of *aspect of face images* impacts the relative importance, thus the eigenstructure. If we can somehow identify this aspect, maybe a similar approach can also be used for non-image data.

To answer the above question, we first investigate the effect of resizing images. Suppose we scaled up the face images from 46×56 back to 92×112. We can see that these images look similar to the original image but with some blocking artifacts. The same effect can be produced by *replacing* neighbor pixels with the same value, such as the value of the top-left pixel. We call this pixel-grouping version of the original image. Figure 5c shows an image of Fig. 5a with pixel-grouping strategy using 2×2 neighborhood. Figure 5b is obtained using downscale + upsample strategy. From Fig. 5, we can see that the downscale + upsample method and the pixel-grouping method produce similar images that are also similar to the original image. Hence, in this subsection, we will consider the feature grouping strategy for the general feature vector.

Indeed, we consider the simple case of replacing one feature with another one; for example, replace feature a by b. Given a vector $\mathbf{x}_t$, let $\tilde{\mathbf{x}}_{tk} = \mathbf{x}_{tk}$ if $k \neq a$ and equals to

$\mathbf{x}_{tb}$ if $k = a$. Following the discussion in the previous subsection, we investigate the dot product of these modified feature vectors:

$$\mathbf{x}_i^T \mathbf{x}_j = \sum_k x_{ik} x_{jk} + x_{ib} x_{jb} - x_{ib} x_{jb} \tag{26}$$

$$= \sum_k \tilde{x}_{ik} \tilde{x}_{jk} + x_{ia} x_{ja} - x_{ib} x_{jb} \tag{27}$$

$$= \tilde{\mathbf{x}}_i^T \tilde{\mathbf{x}}_j + x_{ia} x_{ja} - x_{ib} x_{jb} \tag{28}$$

Hence, the dot matrix can be rewritten as

$$\mathbf{X}^T \mathbf{X} = \tilde{\mathbf{X}}^T \tilde{\mathbf{X}} + \mathbf{D} \tag{29}$$

with $\mathbf{D}_{ij} = x_{ia} x_{ja} - x_{ib} x_{jb}$ From matrix perturbation theory, we know that the difference between the eigenvalue of $\mathbf{X}^T \mathbf{X}$ and that of $\tilde{\mathbf{X}}^T \tilde{\mathbf{X}}$ is bounded by the size of $\mathbf{D}$. That is, if $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_d$ are the eigenvalues of $\mathbf{X}^T \mathbf{X}$ and $\tilde{\lambda}_1 \geq ... \geq \tilde{\lambda}_d$ the eigenvalues of $\tilde{\mathbf{X}}^T \tilde{\mathbf{X}}$, then we have

$$\max_{i=1,\ldots,d} |\lambda_i - \tilde{\lambda}_i| \leq \|\mathbf{D}\|_2 \leq \|\mathbf{D}\|_F \tag{30}$$

The first inequality comes from corollary 8.1.6 in Gene (1996) The second inequality is because norm-2 of the matrix is the square root of the maximum eigenvalue, whereas the Frobenius norm is the square root of the trace that is the sum over all eigenvalues. In the case of matrix $\mathbf{D}$ we have:

$$\|\mathbf{D}\|_F^2 = \sum_i \sum_j |\mathbf{D}_{ij}|^2 \tag{31}$$

$$= \sum_i \sum_j (x_{ia} x_{ja} - x_{ib} x_{jb})^2 \tag{32}$$

$$= \sum_i \sum_j \left((x_{ia} x_{ja})^2 + (x_{ib} x_{jb})^2 - 2(x_{ia} x_{ja})(x_{ib} x_{jb}) \right) \tag{33}$$

$$= \left(\sum_i x_{ia}^2 \right)^2 + \left(\sum_i x_{ib}^2 \right)^2 - 2 \left(\sum_i x_{ia} x_{ib} \right)^2 \tag{34}$$

$$= (\mathbf{X}\mathbf{X}^T)_{aa}^2 + (\mathbf{X}\mathbf{X}^T)_{bb}^2 - 2(\mathbf{X}\mathbf{X}^T)_{ab}^2 \tag{35}$$

It is worth noting that $(\mathbf{X}\mathbf{X}^T)_{ab} / \sqrt{(\mathbf{X}\mathbf{X}^T)_{aa} (\mathbf{X}\mathbf{X}^T)_{bb}}$ is indeed the correlation between features a and b.

From the Eqs. (30) and (35) we can conclude that if a and b are highly correlated, then replacing one by another will only slightly alter the value of obtained eigenvalues. As neighbor pixels are often correlated with each other, the pixels replacing strategy or resizing strategy allows for maintaining dot product, thus the eigenstructure.

4.3 Dot product preserving transformation

From the previous analysis, we know that resizing works because it groups features that are highly correlated with each other. Nonetheless, during the analysis, we have seen another critical component: the dot product preserving transformation. Indeed, if we can somehow transform $\mathbf{x}_t \in \mathbb{R}^d$ into $\tilde{\mathbf{x}}_t \in \mathbb{R}^q$ with $q \ll d$ such that $\mathbf{x}_i^T\mathbf{x}_j \approx \tilde{\mathbf{x}}_i^T\tilde{\mathbf{x}}_j$, then we may follow the same procedure to approximate eigenvalues of the original dataset. The key idea is *to preserve dot product and fast calculation*. This section reviews some transformations that can be used to this end.

4.3.1 Gaussian random projection

An example of such transformation is the Gaussian random projection. Indeed, given Gaussian random vectors $\mathbf{w}_1, \dots, \mathbf{w}_q \in \mathbb{R}^d$ with $\mathbf{w}_i \sim \mathcal{N}(0, \mathbf{I}_d)$ where $\mathbf{I}_d$ is identity matrix in $\mathbb{R}^d$. We know that if q is large enough, then its sample covariance matrix will converge to $\mathbf{I}_d$, i.e. $(1/q)\mathbf{W}\mathbf{W}^T \approx \mathbf{I}_d$ where $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_q]$.. From this observation, we define the following transformation function:

$$f_{\text{gauss}}(\mathbf{x}) = \frac{1}{\sqrt{q}}\mathbf{W}^T\mathbf{x}. \tag{36}$$

Therefore, we have

$$f_{\text{gauss}}(\mathbf{x}_i)^T f_{\text{gauss}}(\mathbf{x}_j) = \frac{1}{q}\mathbf{x}_i^T\mathbf{W}\mathbf{W}^T\mathbf{x}_j \tag{37}$$

$$= \mathbf{x}_i^T\left[\frac{1}{q}\mathbf{W}\mathbf{W}^T\right]\mathbf{x}_j \tag{38}$$

$$\approx \mathbf{x}_i^T\mathbf{x}_j. \tag{39}$$

4.3.2 Feature hashing

Given hash functions $h : \{1, \dots, d\} \to \{1, \dots, q\}$ and $s : \{1, \dots, d\} \to \{1, -1\}$, Weinberger et al. (2009) proposed the following feature hashing function: for $j = 1, \dots, q$

$$f^{(h,s)}(\mathbf{x})_j = \sum_{i:h(i)=j} s(i)\mathbf{x}_i. \tag{40}$$

The dot product between two hash vectors is then

$$\langle f^{(h,s)}(\mathbf{x}), f^{(h,s)}(\mathbf{y})\rangle = \sum_{j=1}^{q} f^{(h,s)}(\mathbf{x})_j f^{(h,s)}(\mathbf{y})_j \tag{41}$$

$$= \sum_j \left(\sum_{i:h(i)=j} s(i)\mathbf{x}_i\right)\left(\sum_{i':h(i')=j} s(i')\mathbf{y}_{i'}\right) \tag{42}$$

$$= \sum_j \sum_{i:h(i)=j} \mathbf{x}_i \mathbf{y}_i + \sum_j \sum_{i \neq i' \wedge h(i)=h(i')} s(i)s(i')\mathbf{x}_i \mathbf{y}_{i'} \tag{43}$$

$$= \sum_i \mathbf{x}_i \mathbf{y}_i + \sum_j \sum_{i \neq i' \wedge h(i)=h(i')} s(i)s(i')\mathbf{x}_i \mathbf{y}_{i'} \tag{44}$$

$$= \langle \mathbf{x}, \mathbf{y} \rangle + \sum_j \sum_{i \neq i' \wedge h(i)=h(i')} s(i)s(i')\mathbf{x}_i \mathbf{y}_{i'} \tag{45}$$

Observe that if we take the expectation over hash function s, then the second term on the right-hand side will vanish. As a result $\mathbb{E}_{h,s}[\langle f^{(h,s)}(\mathbf{x}), f^{(h,s)}(\mathbf{y})\rangle] = \langle \mathbf{x}, \mathbf{y} \rangle$. In Weinberger et al. (2009) the authors proved other properties of this feature hash function, including the bound over the variance of the inner product and concentration of the length of the hash vectors (Weinberger et al. 2009). From the above description, the dot product preserving transformation can be achieved by concatenating hashing vectors from multiple hash functions. Let $(h_i, s_i), i = 1, \dots, l$ be l hash functions we define:

$$f^f(\mathbf{x})_{iq+j} = \frac{1}{l} f^{(h_i, s_i)}(\mathbf{x})_j, \quad i = 1, \dots, l, j = 1, \dots, q. \tag{46}$$

It is worth noting that this hashing is inspired by works in *stream processing field*. In fact, in stream processing, several *sketching methods* have been proposed to keep estimating frequencies and other moments of items using a limited amount of memory. An example of a sketching method is the *count-min sketch* (Graham and MuthuKrishnan 2005) that inspired the work of Shi et al. on *hash kernels* (Shi et al. 2009). Hash kernels rely on hashing function similar to Eq. (40) but without the random sign function s. Hence, Shi et al. (2009) feature hashing was biased, with a bias that reduces as the number of hash functions increases. The random sign function introduced by Weinberger et al. (2009) turns the feature hashing function into an unbiased estimator for dot product. Interestingly, the hash function from Weinberger et al. is the same as the *count sketch* method (Charikar et al. 2002), but with different conditions on the hash functions. This example underlines that dot product preserving transformations could also be found in other domains that may appear to be distant subjects at first, such as stream processing.

4.4 Approximate PCA

In summary, the PCA for large-scale data in high dimensional space can be done by adapting the Algorithm 3 using dot product preserving transformation instead of simple downsizing. This approximate PCA algorithm is summarized in Algorithm 4.

Table 3 Dimension of feature vectors from different deep neural structures

Network	Dimension
VGG16	25,088
DenseNet121	50,176
InceptionV3	51,200
ResNet50	100,352

Algorithm 4 The approximate PCA algorithm

```
1: procedure APPROX-PCA($\mathbf{x}_1, ..., \mathbf{x}_n$ with $\mathbf{x}_t \in \mathbb{R}^d$ with large $d$ and large $n$)   ▷ Return approximate principal components from the given dataset.
2:     Center the data
3:     $\tilde{\mathbf{x}} = f(\mathbf{x}) \in \mathbb{R}^q$ with $f$ dot product preserving function, e.g. image resizing, random projection, feature hashing.
           $q \ll n$ and $\tilde{\mathbf{X}} = [\tilde{\mathbf{x}}_1, ..., \tilde{\mathbf{x}}_n]$
4:     Eigenstructure of new covariance matrix $\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T$: $\lambda_1 \geq ... \geq \lambda_q$ and $\mathbf{u}_1, ..., \mathbf{u}_q, \mathbf{u}_i \in \mathbb{R}^q$
5:     Eigenstructure of new dot matrix : $\lambda_i$ and $(1/\sqrt{\lambda_i})\tilde{\mathbf{X}}^T\mathbf{u}_i \in \mathbb{R}^n$.
6:     Approximate eigenstructure of dot matrix : $\lambda_i$ and $(1/\sqrt{\lambda_i})\tilde{\mathbf{X}}^T\mathbf{u}_i \in \mathbb{R}^n$.
7:     Approximate eigenstructure of covariance matrix : : $\lambda_i$ and $(1/\lambda_i)\mathbf{X}\tilde{\mathbf{X}}^T\mathbf{u}_i \in \mathbb{R}^d$.
8:     Select $m$ as discussed above.
9:     return first $m$ eigenvectors and eigenvalues.
10: end procedure
```

4.5 Examples

To illustrate the application of PCA in this case (large n and large d), a larger dataset is considered namely Dogs VS Cats dataset[7]. The training split of this dataset contains 25,000 images of cats and dogs in various sizes. These images were resized into 224×224 pixels, then fed into several deep neural networks to extract visual feature vectors. Table 3 summarizes the dimensions of feature vector from different deep neural structures.

Figure 6 shows Mean Absolute Proportion Error (MAPE) between the dot product in the input space and that in the space defined by Gaussian random projection (top row) and feature hashing (bottom row). These plots are obtained using q equal to 500, 1000, 2000, 3000, and 5000 dimensions. Figure 6a–d are obtained from VGG16, DenseNet121, InceptionV3, and ResNet50 respectively. Similarly, Fig. 6e–h are obtained from VGG16, DenseNet121, InceptionV3, and ResNet50. This Figure shows that the MAPE reduces as q increases. Moreover, comparing the two dot product preserving transformations, we can observe that Gaussian random projection outperforms feature hashing in several cases. Thus, we shall consider only Gaussian random projection in the following examples.

Figure 7 shows the plot of eigenvalues obtained by approximate PCA using Gaussian random projection with 500 dimensions in the top row. This Figure also shows the plot of the cumulative proportion of total variance obtained from different neural structures in the second row. The third row of this Figure shows the projection onto two first principal axis with red color for "Dog" and blue for "Cat". One can observe that even if InceptionV3 has

[7] https://www.kaggle.com/competitions/dogs-vs-cats/.

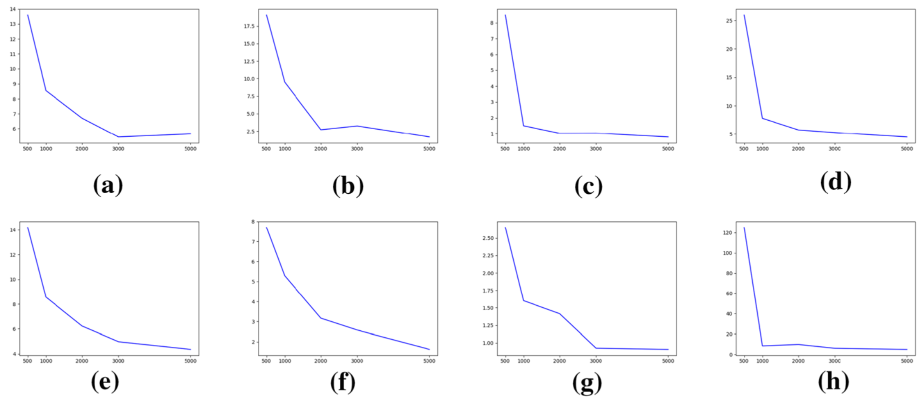

Fig. 6 Mean Absolute Proportion Error (MAPE) between the dot product in the input space and that in the space defined by Gaussian random projection (top row) and feature hashing (bottom row)

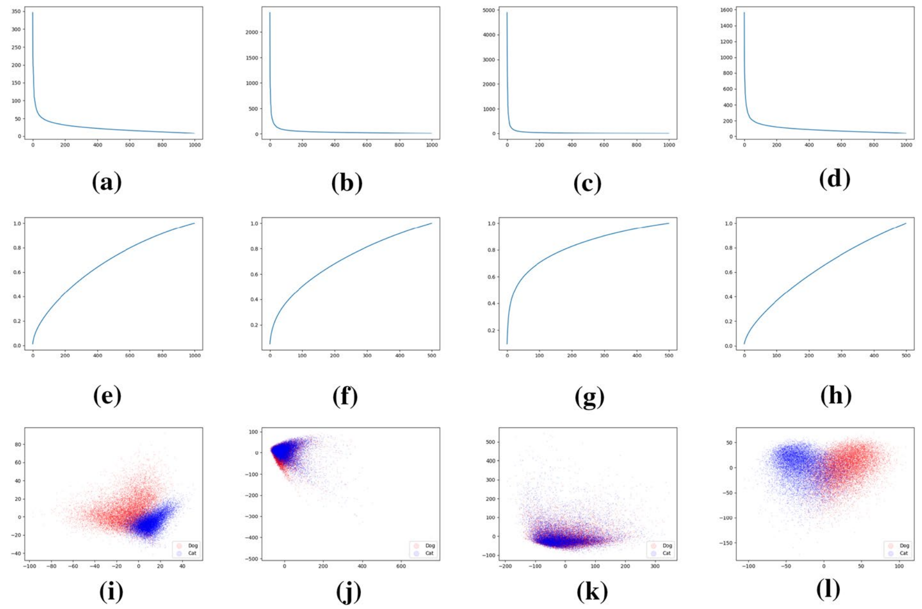

Fig. 7 Eigenvalues plot in decreasing order. These eigenvalues are obtained from approximate PCA using Gaussian random projection with $q = 500$ from VGG16 (**a**), DenseNet121 (**b**), InceptionV3 (**c**), and ResNet50 (**d**). The second row show the cumulative proportion of total variance for VGG16 (**e**), DenseNet121 (**f**), InceptionV3 (**g**a), and ResNet50 (**h**)

the fastest increase in cumulative variance, its scatter plot shows a mixed area between the two classes. For VGG16 and ResNet50, the two classes seem to be better separated. Nonetheless, it should be emphasized that PCA is not designed for classification tasks. Hence, one should not expect the principal subspace to be suited for classification in all cases. Another subspace projection, such as linear discriminant analysis or LDA (Fukunaga 1990), that is designed for classification, could yield better classification accuracy.

5 Kernel PCA and its approximation methods

So far, we have considered a simple linear projection of numerical feature vectors. In practice, most data could be cluttered around an area that may not form a linear subspace but a non-linear one. A convenient way of performing non-linear PCA is to use *kernel trick* (Schölkopf et al. 1999, 2018). The kernel trick idea is to reformulate any linear analysis method to involve only the dot product, then replace the dot product with a *kernel function*. Et Voilà! You obtain a non-linear version of the original linear method.

The kernel PCA relies on this kernel trick to perform non-linear projection. The main crux of kernel PCA lies in the eigen-decomposition of the kernel matrix whose size is $n \times n$ where n is the size of the dataset. This section briefly reviews the kernel method, and the Kernel PCA (Schölkopf et al. 1999) in Sect. 5.1. Then the approximate KPCA is discussed in Sects. 5.2 and 5.3.

5.1 Kernel PCA

The kernel considered in this manuscript is the *Mercer's kernel* (Schölkopf et al. 1999; Shawe-Taylor and Cristianini 2004). In short, if κ is a valid Mercer's kernel, then there exists a mapping function Φ such that

$$\kappa(\mathbf{x}, \mathbf{y}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle. \tag{47}$$

Recall that $\langle ., . \rangle$ denotes the dot product. This bracket notation will be used in the following to render formulas easier to read.

The range of the mapping function Φ is called the *reproducing kernel Hilbert space* or *RKHS*. To simplify the discussion, we shall refer to it as the kernel space instead of the input space.

The key of the kernel method is that Φ can be *unknown* meaning that we know *it does exist, but we do not know its analytical form*. Fortunately, several properties can be derived implicitly from the dot product without accessing the vector's coordinate. For example, the Euclidean distance in kernel space is given by:

$$||\Phi(\mathbf{x}) - \Phi(\mathbf{y})||^2 = \kappa(\mathbf{x}, \mathbf{x}) + \kappa(\mathbf{y}, \mathbf{y}) - 2\kappa(\mathbf{x}, \mathbf{y}) \tag{48}$$

This implicit calculation allows transforming the classical linear method into a non-linear one at the expense of a higher computational cost.

5.1.1 Implicit centering in kernel space

Indeed as Φ is unknown, we cannot explicitly compute the mean vector nor subtract it from each data point in the kernel space. However, the mean vector $\boldsymbol{\mu}$ can be defined implicitly via Φ as follows:

$$\boldsymbol{\mu} = \frac{1}{n} \sum_{t=1}^{n} \Phi(\mathbf{x}_t) \tag{49}$$

Even if we cannot compute the mean vector explicitly in kernel space, we can compute the dot product between two data points *after* mean subtraction as follows:

$$\begin{aligned}\langle \Phi(\mathbf{x}) - \boldsymbol{\mu}, \Phi(\mathbf{y}) - \boldsymbol{\mu} \rangle =& \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle - \langle \Phi(\mathbf{x}), \Phi(\boldsymbol{\mu}) \rangle - \langle \Phi(\mathbf{y}), \Phi(\boldsymbol{\mu}) \rangle \\ &+ \langle \Phi(\boldsymbol{\mu}), \Phi(\boldsymbol{\mu}) \rangle\end{aligned} \tag{50}$$

$$\begin{aligned}=&\langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle - \frac{1}{n}\sum_{t=1}^{n}\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}_t) \rangle - \frac{1}{n}\sum_{t=1}^{n}\langle \Phi(\mathbf{y}), \Phi(\mathbf{x}_t) \rangle \\ &+ \frac{1}{n^2}\sum_{t=1}^{n}\sum_{t'=1}^{n}\langle \Phi(\mathbf{x}_t), \Phi(\mathbf{x}_{t'}) \rangle\end{aligned} \tag{51}$$

$$\begin{aligned}=&\kappa(\mathbf{x}, \mathbf{y}) - \frac{1}{n}\sum_{t=1}^{n}\kappa(\mathbf{x}, \mathbf{x}_t) - \frac{1}{n}\sum_{t=1}^{n}\kappa(\mathbf{y}, \mathbf{x}_t)\rangle \\ &+ \frac{1}{n^2}\sum_{t=1}^{n}\sum_{t'=1}^{n}\kappa(\mathbf{x}_t, \mathbf{x}_{t'})\rangle\end{aligned} \tag{52}$$

$$\equiv \hat{\kappa}(\mathbf{x}, \mathbf{y}). \tag{53}$$

In summary, data centering in kernel space can be done implicitly via the modified kernel $\hat{\kappa}$. This modified kernel will be used instead of the dot product in dot matrix calculation. To reduce confusion, we shall refer to the dot matrix in this case as the *kernel matrix*. The kernel PCA relies on the eigen-decomposition of this kernel matrix $\mathbf{K}$:

$$\mathbf{K}_{i,j} = \hat{\kappa}(\mathbf{x}_i, \mathbf{x}_j), \quad 0 \le i,j \le n. \tag{54}$$

For n data points, the size of this kernel matrix will be $n \times n$.

5.1.2 Projection axis in kernel space

Before writing down the kernel PCA algorithm, it should be noted that the projection axis is also defined in the kernel space, similar to the mean vector above. Hence, it is impossible to write it down explicitly since Φ is unknown. We have to work with its definition. Indeed, by adopting the Eq. (25) to the kernel space with normalization, an eigenvector $\mathbf{u}_i$ of the covariance matrix in kernel space must be of the form:

$$\mathbf{u}_i = \frac{1}{\sqrt{\lambda_i}}\sum_{t=1}^{n} v_{i,t}\Phi(\mathbf{x}_t), \tag{55}$$

where $\mathbf{v}_i = [v_{i,1}, ..., v_{i,n}]^T$ is the i^{th} eigenvector of the kernel matrix and λ_i its corresponding eigenvalue.

Even if the numerical values of the vector $\mathbf{u}_i$ cannot be computed explicitly, we may still compute the projection of $\Phi(\mathbf{x})$ onto $\mathbf{u}_i$ as follows:

$$\begin{aligned}\langle \mathbf{u}_i, \Phi(\mathbf{x})\rangle &= \frac{1}{\sqrt{\lambda_i}} \sum_{t=1}^{n} v_{i,t}\langle \Phi(\mathbf{x}_t), \Phi(\mathbf{x})\rangle \\ &= \frac{1}{\sqrt{\lambda_i}} \sum_{t=1}^{n} v_{i,t}\kappa(\mathbf{x}_t, \mathbf{x})\end{aligned}$$

Given a new data point $\mathbf{x}$, the projection of $\Phi(\mathbf{x})$ onto i^{th} eigenvector is given by:

$$\langle \mathbf{u}_i, \Phi(\mathbf{x})\rangle = \frac{1}{\sqrt{\lambda_i}} \sum_{t=1}^{n} v_{i,t}\kappa(\mathbf{x}_t, \mathbf{x}). \tag{56}$$

Note that we may replace κ in the Eq. (56) by $\tilde{\kappa}$ for projection of centered data in kernel space.

5.1.3 KPCA

The whole kernel PCA can be summarized as shown in Algorithm 5.

Algorithm 5 The kernel PCA algorithm

```
1: procedure KPCA($n$ data points $\mathbf{x}_1, ..., \mathbf{x}_n$ and a kernel function $\kappa$)    ▷ Return
   principal components from the given dataset in kernel space.
2:     Define modified kernel $\tilde{\kappa}$
3:     Construct kernel matrix $\mathbf{K}$ with $K_{i,j} = \tilde{\kappa}(\mathbf{x}_i, \mathbf{x}_j)$
4:     Eigen-decompose $\mathbf{K}$
5:     Select $m$ as discussed above.
6:     return $\frac{1}{\sqrt{\lambda_1}}\mathbf{v}_1, ..., \frac{1}{\sqrt{\lambda_m}}\mathbf{v}_m$, $\mathbf{v}_i \in \mathbb{R}^n$.
7: end procedure
```

5.2 Approximate Kernel PCA

The main crux of kernel PCA lies in the eigen-decomposition of the kernel matrix. Indeed, the size of the kernel matrix is $n \times n$, where n is the size of the dataset. As the mapping function, Φ is unknown. It is not possible applying the method described earlier in Sect. 4 directly. In the following, we first present *empirical kernel map* and its relation to the kernel matrix. Then we combine it with the approximation method described in Sect. 4.

5.2.1 Empirical kernel map and kernel matrix

Given a dataset of n points $\mathbf{x}_1, ..., \mathbf{x}_n$ and a kernel κ, the empirical kernel map (EKM) transforms any data point $\mathbf{x}$ into n-dimensional space as follows:

$$\tilde{\Phi}(\mathbf{x}) = \begin{bmatrix} \kappa(\mathbf{x}, \mathbf{x}_1) \\ \vdots \\ \kappa(\mathbf{x}, \mathbf{x}_n) \end{bmatrix} \tag{57}$$

It is straightforward, seeing that the dot product between two EKM vectors is not equal to the kernel function. Thus EKM does not preserve the dot product in kernel space. Some authors have proposed additional methods to restore the kernel value (Schölkopf et al. 1999). Our method will not follow this line. Indeed, we consider the outer product used to construct the covariance matrix instead.

Let $\hat{\mathbf{x}}_t = \Phi(\mathbf{x}_t)$ be the mapped value of $\mathbf{x}_t$ in the kernel space via the kernel κ with its mapping function Φ. In fact $\hat{\mathbf{x}}_t$ is not computable since Φ is unknown. Let $\hat{\mathbf{X}} = [\hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_n]$ be the obtained data matrix. The kernel matrix can be expressed as $\hat{\mathbf{X}}^T\hat{\mathbf{X}}$, and the EKM can be rewritten by

$$\tilde{\Phi}(\hat{\mathbf{x}}_t) = \hat{\mathbf{X}}^T\hat{\mathbf{x}}_t. \tag{58}$$

Hence

$$\tilde{\Phi}(\mathbf{x}_t)\tilde{\Phi}(\mathbf{x}_t)^T = \hat{\mathbf{X}}^T\hat{\mathbf{x}}_t\hat{\mathbf{x}}_t^T\hat{\mathbf{X}} \tag{59}$$

$$\sum_{t=1}^{n} \tilde{\Phi}(\mathbf{x}_t)\tilde{\Phi}(\mathbf{x}_t)^T = \hat{\mathbf{X}}^T\hat{\mathbf{X}}\hat{\mathbf{X}}^T\hat{\mathbf{X}} \tag{60}$$

$$= \mathbf{K}^2, \tag{61}$$

where $\mathbf{K}$ is the kernel matrix.

5.2.2 Large-scale approximate KPCA algorithm

The last equation in the previous section implies that the covariance matrix of EKM vectors is exactly squared of the kernel matrix. Hence, we may decompose the former instead of the latter. However, one may wonder about this advantage since both matrices are $n \times n$ with large n.

The advantage is that we now have access to these EKM vectors' coordinates. Therefore, it is possible to reduce the computational cost using dot product preserving transformation described in Sect. 4. The whole approximate kernel PCA can be summarized in Algorithm 6

Algorithm 6 The approximate kernel PCA algorithm

```
1: procedure APPROX-KPCA($n$ data points $\mathbf{x}_1, ..., \mathbf{x}_n$ and a kernel function $\kappa$)  ▷ Return approximate principal components from the given dataset in kernel space.
2:     Define modified kernel $\tilde{\kappa}$
3:     Construct EKM vectors $\hat{\mathbf{x}}_t = \tilde{\Phi}(\mathbf{x}_t), t = 1, ..., n$, let $\tilde{\mathbf{X}} = [\tilde{\mathbf{x}}_1, ..., \tilde{\mathbf{x}}_n]$
4:     $\tilde{\mathbf{z}}_t = f(\hat{\mathbf{x}}_t) \in \mathbb{R}^q$ with $f$ dot product preserving function, e.g. random projection or feature hashing, with $q \ll n$.
       $\tilde{\mathbf{Z}} = [\tilde{\mathbf{z}}_1, ..., \tilde{\mathbf{z}}_n]$
5:     Eigenstructure of new covariance matrix $\tilde{\mathbf{Z}}\tilde{\mathbf{Z}}^T$: $\lambda_1 \geq ... \geq \lambda_q$ and $\mathbf{u}_1, ..., \mathbf{u}_q, \mathbf{u}_i \in \mathbb{R}^q$
6:     Eigenstructure of new dot matrix : $\lambda_i$ and $(1/\sqrt{\lambda_i})\tilde{\mathbf{Z}}^T\mathbf{u}_i \in \mathbb{R}^n$.
7:     Approximate eigenstructure of dot matrix : $\lambda_i$ and $(1/\sqrt{\lambda_i})\tilde{\mathbf{Z}}^T\mathbf{u}_i \in \mathbb{R}^n$.
8:     Approximate eigenstructure of covariance matrix : $\lambda_i$ and $(1/\lambda_i)\hat{\mathbf{X}}\tilde{\mathbf{Z}}^T\mathbf{u}_i \in \mathbb{R}^d$.
9:     Approximate eigenstructure of kernel matrix : $\sqrt{\lambda_i}$ and $(1/\lambda_i)\hat{\mathbf{X}}\tilde{\mathbf{Z}}^T\mathbf{u}_i \in \mathbb{R}^d$.
10:    Select $m$ as discussed above.
11:    return $m$ first approximate eigenvectors of kernel matrix.
12: end procedure
```

5.3 Nyström method

It should be noted that there exists another approximate kernel matrix eigen-decomposition called the *Nyström* method (Williams and Seeger 2000). The basic idea of the Nyström algorithm is to eigen-decompose a small selected working set first. Then to expand the resulting eigenvectors/eigenvalues to cover the whole dataset.

Let $\mathbf{x}_1, ..., \mathbf{x}_n$ be given data. Suppose, without loss of generality, that q first points were selected. Let λ_i^q and $\mathbf{u}_i^q, i = 1, \dots, m$ be the eigenvalues and the corresponding eigenvectors obtained from q data points. Nyström method approximate the eigenvalues/eigenvectors using kernel matrix $\mathbf{K}$ as follows:

$$\hat{\lambda}_i = \frac{n}{q}\lambda_i^q \tag{62}$$

$$\hat{\mathbf{u}}_i = \sqrt{\frac{n}{q}}\frac{1}{\lambda_i^q}\mathbf{K}_{n,q}\mathbf{u}_i^q, \tag{63}$$

where $\mathbf{K}_{n,q}$ is the appropriate $n \times q$ sub-matrix of the whole kernel matrix $\mathbf{K}$.

As the eigen-decomposition is performed on the working set, its selection is crucial for the correctness of the approximated eigenstructure. To this end, several selection methods have been proposed, including uniform sampling, diagonal sampling, and column-norm sampling (Kumar et al. April 2012). For the uniform sampling, each example has an equal probability of being selected. The diagonal sampling selects working point $\mathbf{x}_i$ according to probability p_i^d such that:

$$p_i^d = \frac{\kappa(\mathbf{x}_i, \mathbf{x}_i)^2}{\sum_j \kappa(\mathbf{x}_i, \mathbf{x}_j)^2}. \tag{64}$$

The column-norm sampling selects working set with probability p_i^c such that:

$$p_i^c = \frac{\sum_j \kappa(\mathbf{x}_i, \mathbf{x}_j)^2}{\sum_k \sum_j \kappa(\mathbf{x}_k, \mathbf{x}_j)^2}. \tag{65}$$

5.4 Complexity analysis

For the approximate KPCA, the main computational cost is in EKM calculation ($\mathcal{O}(n^2)$), the dot product preserving transformation ($\mathcal{O}(qnd)$), the eigen-decomposition ($\mathcal{O}(q^3)$), and eigen-conversion ($\mathcal{O}(qn^2)$). The highest cost is in eigen-decomposition and eigen-conversion. The total complexity of this method is $\mathcal{O}((q+1)n^2 + qnd + q^3)$.

For Nyström algorithm, the main computational cost is in working set selection, eigen-decomposition ($\mathcal{O}(q^3)$), and eigen-conversion ($\mathcal{O}(q^2n)$). For working set selection, the complexity varies depending on the selection method. In fact, the complexity of uniform, diagonal, and column-norm sampling are $\mathcal{O}(1)$, $\mathcal{O}(n)$, and $\mathcal{O}(n^2)$ respectively. The complexity of the Nyström method is smaller than that of Approx-KPCA, especially when uniform sampling is used. It is worth noting that in some cases, this simple uniform sampling could outperform diagonal and column-norm sampling. This is due to the randomized nature of the sampling procedure. Thus, we would suggest using uniform or diagonal sampling first since they have lower complexity costs.

5.5 Examples

To illustrate the application of KPCA and its approximation methods, the ORL dataset was considered again. The dimension of the lower subspace in Approx-KPCA is set to the same value as the size of the working set in the Nyström method. Thus Approx-KPCA and Nyström algorithm will need to eigen-decompose matrices of the same size, i.e., $q \times q$. To compare the two approximation methods, it is worth emphasizing that an eigenvector represents the projection axis but not the direction. Indeed, an eigenvector $\mathbf{v}$ and its negation $-\mathbf{v}$ will have the same eigenvalue, i.e., the same variance of projected data. Hence, to compare eigenvectors obtained from the different methods, we consider the absolute correlation between them. Let $\mathbf{v}$ and $\mathbf{v}'$ be two eigenvectors obtained from two different methods, and the similarity measure is defined as follows:

$$\text{Sim}\,(\mathbf{v}, \mathbf{v}') = \frac{|\mathbf{v}^T\mathbf{v}'|}{||\mathbf{v}||\,||\mathbf{v}'||} \tag{66}$$

A suitable approximation method should have a high similarity, close to 1.

Furthermore, we also consider the difference in eigenvalues. Indeed, let λ and λ' be two eigenvalues obtained from vanilla KPCA and one of the two approximation methods, respectively, The difference between λ and λ' is measured as:

$$\text{Diff}\,(\lambda, \lambda') = \frac{|\lambda - \lambda'|}{\lambda}. \tag{67}$$

A suitable approximation method should have a smaller difference.

We perform KPCA, Approx-KPCA and KPCA with Nyström using various kernels, namely Tangent hyperbolic kernel with $a = 0.0001, r = 0$, RBF kernel with $\gamma = 0.0001$, Polynomial kernel degree 5, and the sum of these three kernels. The resulting similarities and difference are shown in Figs. 8 and 9. Approx-KPCA (g) and (f) correspond to approximate KPCA using Gaussian random projection and feature hashing, respectively. Nyström (r) refers to Nyström method with random working set selection. Nyström (d) and (c) refer to Nyström method with diagonal sampling and column-norm sampling, respectively The experiment was repeated ten times.

Figures 8 and 9 show average similarity between 10 first eigenvectors and average difference between 10 first eigenvalues respectively. From the two Figures, one can observe that both Approx-KPCA and Nyström method yield high similarity, especially for the first eigenvectors. This similarity increases as q increases. Therefore, one can expect that the principal space obtained from these approximation methods lies close to the real one. For eigenvalues, one can see that Approx-KPCA with Gaussian random projection is the best in several cases. Approx-KPCA with feature hashing yields a large difference for tangent hyperbolic and RBF kernels. Nyström method with diagonal and column sampling yield a large difference for polynomial and sum kernels. From these results, we would suggest using Approx-KPCA with Gaussian random projection when precise eigenvalues are needed.

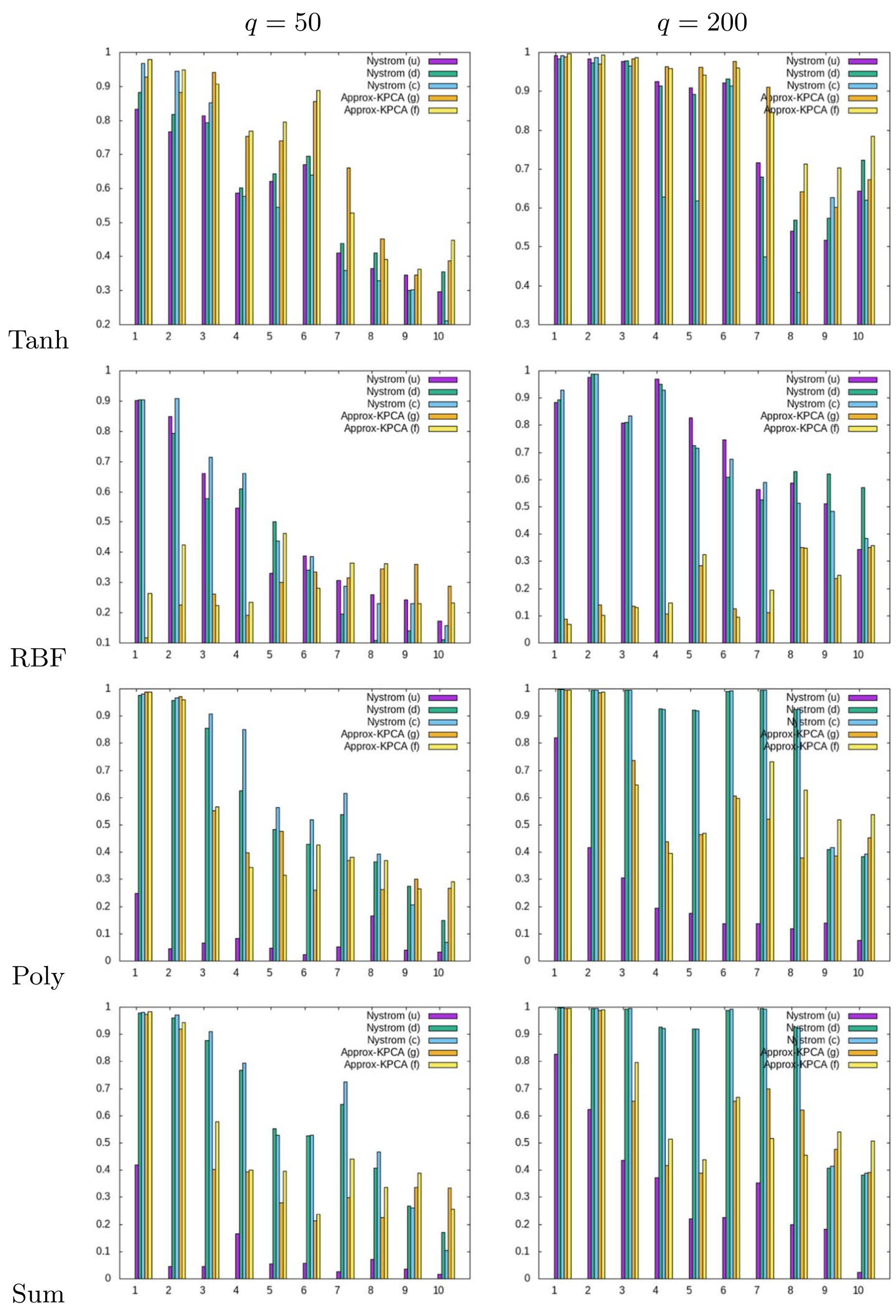

Fig. 8 Average similarity between first 10 eigenvectors obtained from KPCA and that obtained from difference approximation methods

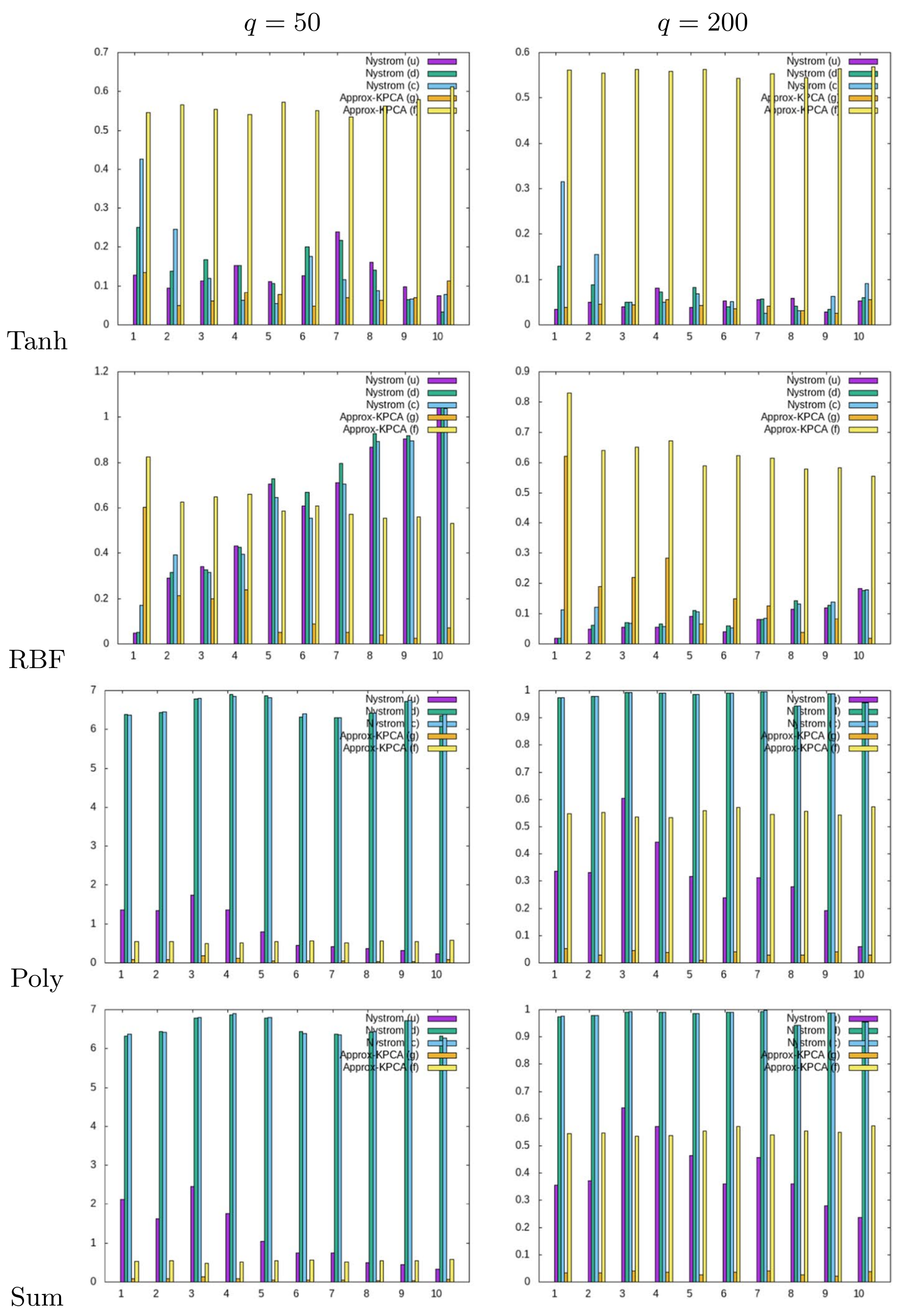

Fig. 9 Average difference between first 10 eigenvalues obtained from KPCA and that obtained from difference approximation methods

6 Summary

In this tutorial, we reviewed the classical PCA method and the problem that may arise when applying it to very small or very large high-dimensional datasets. We have also discussed different methods that may be used to handle these cases. We have shown that a good PCA approximation could be achieved with dot product preserving transformation. This paper considers two particular transformations: Gaussian random projection and the feature hashing method. Other transformations could also be used if they provide dot product preservation and fast calculation. We also showed how this approximation method could be applied to alleviate the computational problem that occurred in kernel PCA. We have made our code in Python that is used in the experiment available at https://github.com/peune/pca. Readers with a background in Python programming can study this code to better understand this subject. All data used in this paper are from public datasets. They are also available upon request.

A Jacobi method for eigen-decomposition

Given a square matrix $\mathbf{A}$ of size $d \times d$, its eigen-decomposition is given by:

$$\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^T \tag{68}$$

with $\mathbf{D} = \operatorname{diag}(\lambda_1, \dots, \lambda_d)$ and $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_d]$ such that $\mathbf{v}_i^T\mathbf{v}_j = 1$ if $i = j$ and 0 otherwise.

The vectors $\mathbf{v}_1, \dots, \mathbf{v}_d$ are eigenvectors of $\mathbf{A}$ with corresponding eigenvalues $\lambda_1, \dots, \lambda_d$:

$$\mathbf{A}\mathbf{v}_i = \lambda_i\mathbf{v}_i. \tag{69}$$

The eigen-decomposition of $\mathbf{A}$, gives a powerful method to compute power of $\mathbf{A}$, e.g. $\mathbf{A}^p = \mathbf{V}\mathbf{D}^p\mathbf{V}^T$ where $\mathbf{D}^p = \operatorname{diag}(\lambda_1^p, ..., \lambda_d^p)$. In particular, when $p = -1$, this eigen-decomposition tells us that $\mathbf{A}$ is not invertible if some of its eigenvalues are zeros. It is also possible to compute fancy things like $\mathbf{A}^{1/2}$.

There are several eigen-decomposition algorithms. One of the most widely used is the Jacobi method (Jacobi 1846), which relies on rotation matrices. The obtained eigenvectors form an orthonormal basis of the d-dimensional space. Observed that $\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^T$ can be rearranged as $\mathbf{V}^T\mathbf{A}\mathbf{V} = \mathbf{D}$. Therefore, eigen-decomposition can be found by searching for a matrix $\mathbf{M}$ such that that the product $\mathbf{M}^T\mathbf{A}\mathbf{M}$ is a diagonal matrix. To this end, the Jacobi method will successively rotate the default basis of the d-dimensional space until the product becomes diagonal. In fact, the d-dimensional space, the default basis is composed of vector $\mathbf{e}_i = [0, ..., \underbrace{1}_{i^{th}}, ..., 0]^T, i = 1, \dots, d$. The rotation is performed on a couple of axes. For example given two axis i, j and a rotation angle we define the *rotation matrix* $P(i,j,\theta)$:

$$P(i,j,\theta) = \begin{array}{c} \\ \\ \\ i \\ \\ j \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccccc} & & i & & j & & \end{array} \\ \begin{pmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \dots & c & \dots & -s & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \dots & s & \dots & c & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{pmatrix} \end{array} \tag{70}$$

where $c = \cos\theta$ and $s = \sin\theta$ It can be proved that $\mathbf{A}$ and $\mathbf{PAP}^T$ will have the same eigenstructure. Therefore, Jacobi method tries to find $P_1, P_2, \dots, P_n$ such that $P_n ... P_1 A P_1^T ... P_n^T$ is diagonal.

The main question is how to select (i, j) and θ. The Jacobi method answers this question by considering $\mathbf{A}' = \mathbf{PAP}^T$

$$\begin{aligned} \mathbf{A}'_{ii} &= c^2\mathbf{A}_{ii} - 2sc\mathbf{A}_{ij} + s^2\mathbf{A}_{jj} \\ \mathbf{A}'_{jj} &= s^2\mathbf{A}_{ii} + 2sc\mathbf{A}_{ij} + c^2\mathbf{A}_{jj} \\ \mathbf{A}'_{ij} = \mathbf{A}'_{ji} &= (c^2 - s^2)\mathbf{A}_{ij} + sc(\mathbf{A}_{ii} - \mathbf{A}_{jj}) \\ \mathbf{A}'_{ik} = \mathbf{A}'_{ki} &= c\mathbf{A}_{ik} - s\mathbf{A}_{jk} \; k \neq i, j \\ \mathbf{A}'_{jk} = \mathbf{A}'_{kj} &= s\mathbf{A}_{ik} + c\mathbf{A}_{jk} \; k \neq i, j \\ \mathbf{A}'_{kl} &= \mathbf{A}_{kl} \; k, l \neq i, j \end{aligned} \tag{71}$$

As we want $\mathbf{A}'$ to become closer and closer to the diagonal matrix, we should select the largest off-diagonal element $\mathbf{A}'_{ij}$ and choose proper θ to shrink its value to 0. For this task, we select θ such that

$$\begin{aligned} 0 &= (c^2 - s^2)\mathbf{A}_{ij} + sc(\mathbf{A}_{ii} - \mathbf{A}_{jj}) \\ &= \cos(2\theta)\mathbf{A}_{ij} + \frac{1}{2}\sin(2\theta)(\mathbf{A}_{ii} - \mathbf{A}_{jj}) \\ \tan(2\theta) &= \frac{2\mathbf{A}_{ij}}{\mathbf{A}_{jj} - \mathbf{A}_{ii}.} \end{aligned} \tag{72}$$

If $\mathbf{A}_{ii} = \mathbf{A}_{jj}$ we select $\theta = \pi/4$. Hence, the Jacobi method iteratively selects (i, j) from largest off-diagonal element, then construct rotation matrix using θ as defined in 72. The whole process is repeated until all off-diagonal elements become small enough. This requires computational cost around $\mathcal{O}(d^3)$.

B Kernel function

This section introduces the basic idea behind kernel and kernel methods (Schölkopf et al. 1999, 2018; Shawe-Taylor and Cristianini 2004; Hofmann et al. 2008). The kernel that we are interested in is the *Mercer's kernel*. A real-valued function κ satisfies the Mercer's condition if for all square-integrable function g we have $\int\int g(\mathbf{x})\kappa(\mathbf{x}, \mathbf{y})g(\mathbf{y}) \geq 0$. In practice, given a dataset $\mathbf{x}_1, \dots, \mathbf{n}$ and its kernel matrix $\mathbf{K}$, this condition implies that the kernel matrix is positive-semidefinite matrix, i.e. for all vector $\mathbf{g} \in \mathbb{R}^n$ we have $\mathbf{g}^T\mathbf{K}\mathbf{g} \geq 0$ The Mercer condition guarantees that there exists a mapping function Φ such that

$$\kappa(\mathbf{x}, \mathbf{y}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle. \tag{73}$$

Recall that $\langle ., . \rangle$ denotes the dot product. This bracket notation will be used in the following to render formulas easier to read.

The range of the mapping function Φ is called the *reproducing kernel Hilbert space* or *RKHS*. To simplify the discussion, we shall refer to it as the kernel space as opposed to the input space.

The key of the kernel method is that Φ can be *unknown* meaning that we know *it does exist, but we do not know how it is*. Fortunately, several properties can be derived directly from the dot product without accessing the vector's coordinate. Thus, these properties can also be computed from kernel function, even if Φ remains unknown. An example of such properties is the Euclidean distance. Indeed, as we have

$$||\mathbf{x} - \mathbf{y}||^2 = \mathbf{x}^T\mathbf{x} + \mathbf{y}^T\mathbf{y} - 2\mathbf{x}^T\mathbf{y}, \tag{74}$$

so by replacing the dot product with a kernel function we obtain:

$$||\Phi(\mathbf{x}) - \Phi(\mathbf{y})||^2 = \kappa(\mathbf{x}, \mathbf{x}) + \kappa(\mathbf{y}, \mathbf{y}) - 2\kappa(\mathbf{x}, \mathbf{y}), \tag{75}$$

where Φ is the mapping function of the kernel κ.

Examples of valid kernels are:

- Linear kernel $\kappa(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}$
- Polynomial kernel $\kappa_(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y})^p$
- RBF kernel or Gaussian kernel $\kappa(\mathbf{x}, \mathbf{y}) = \exp(-\gamma||\mathbf{x} - \mathbf{y})||^2$
- Laplacian kernel $\kappa(\mathbf{x}, \mathbf{y}) = \exp(-||\mathbf{x} - \mathbf{y}||/\sigma)$
- Tangent hyperbolic kernel $\kappa(\mathbf{x}, \mathbf{y}) = \tanh(a\mathbf{x}^T\mathbf{y} + r)$
- Histogram intersection kernel (Barla et al. 2003) $\kappa(\mathbf{x}, \mathbf{y}) = \sum_i \min\{\mathbf{x}_i, \mathbf{y}_i\}$ when $\mathbf{x}, \mathbf{y}$ are two histograms
- Generalized histogram intersection kernel (Boughorbel et al. 2005) $\kappa(\mathbf{x}, \mathbf{y}) = \sum_i \min\{|\mathbf{x}_i|^\alpha, |\mathbf{y}_i|^\alpha\}$
- Chi-square kernel (Hein and Bousquet 2004, 2005) between two distributions $\mathbf{P}$ and $\mathbf{Q}$:

$$\kappa(\mathbf{P}, \mathbf{Q}) = \int_{\mathcal{X}} \frac{p(x)q(x)}{p(x) + q(x)} d\mu(x)$$

- Hellinger kernel (Hein and Bousquet 2004, 2005) between two distributions $\mathbf{P}$ and $\mathbf{Q}$:

$$\kappa(\mathbf{P}, \mathbf{Q}) = \int_{\mathcal{X}} \sqrt{p(x)q(x)} d\mu(x)$$

- Jensen-Shannon kernel (Hein and Bousquet 2004, 2005) between two distributions $\mathbf{P}$ and $\mathbf{Q}$:

$$\kappa(\mathbf{P}, \mathbf{Q}) = -\frac{1}{\log 2} \int_{\mathcal{X}} p(x) \log \frac{p(x)}{p(x) + q(x)} + q(x) \log \frac{q(x)}{p(x) + q(x)} d\mu(x)$$

- String kernel (Lodhi et al. 2002) between two strings $\mathbf{x}, \mathbf{y}$ defined over finite alphabet Σ:

  $$\kappa(\mathbf{x}, \mathbf{y}) = \sum_{\mathbf{u} \in \Sigma^n} \psi_{\mathbf{u}}(\mathbf{x}) \psi_{\mathbf{u}}(\mathbf{y}),$$

  where n is the given length of sub-string considered. For a string $\mathbf{u} = \mathbf{u}_1, \dots, \mathbf{u}_{|\mathbf{u}|}$ with $\mathbf{u}_i \in \Sigma$, let $\mathbf{u}[i:j] = \mathbf{u}_i, ..., \mathbf{u}_j$. The mapping function $\psi_{\mathbf{u}}(\mathbf{x})$ is defined as the weighted sum of sub-string $\mathbf{u}$ found within $\mathbf{x}$, i.e.

  $$\psi_{\mathbf{u}}(\mathbf{x}) = \sum_{i : \mathbf{x}[i : |\mathbf{u}|] = \mathbf{u}} \lambda^{|\mathbf{u}|},$$

  with $\lambda < 1$ weighting coefficient.
- Fisher kernel (Jaakkola and Haussler 1999): for a given probabilistic model p:

  $$\kappa(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x})^T \mathbf{I}^{-1} \Phi(\mathbf{y}),$$

  where the explicit mapping function Φ is the gradient of log likelihood:

  $$\Phi(\mathbf{x}) = \nabla_\theta \log p(\mathbf{x}|\theta),$$

  and $\mathbf{I}$ is the *Fisher's information matrix*, i.e.

  $$\mathbf{I} = \mathbb{E}\left[\Phi(\mathbf{x})\Phi(\mathbf{x})^T\right].$$

  In practice, we often use an approximate version of this kernel, i.e. $\kappa(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x})^T \Phi(\mathbf{y})$. This kernel allows nicely integrating a generative model into a discriminative framework such as SVM.

It is also possible constructing a new kernel from existing kernels. Let $\kappa_1, \kappa_2 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be two valid kernels. The following kernels are also valid

- $\kappa(\mathbf{x}, \mathbf{y}) = \kappa_1(\mathbf{x}, \mathbf{y}) + \kappa_2(\mathbf{x}, \mathbf{y})$
- $\kappa(\mathbf{x}, \mathbf{y}) = c\kappa_1(\mathbf{x}, \mathbf{y}), \ c \in \mathbb{R}^+$
- $\kappa(\mathbf{x}, \mathbf{y}) = \kappa_1(\mathbf{x}, \mathbf{y}) + c, \ c \in \mathbb{R}^+$
- $\kappa(\mathbf{x}, \mathbf{y}) = \kappa_1(\mathbf{x}, \mathbf{y})\kappa_2(\mathbf{x}, \mathbf{y})$
- $\kappa(\mathbf{x}, \mathbf{y}) = f(\mathbf{x})f(\mathbf{y})$ for all $f : \mathcal{X} \to \mathbb{R}$
- $\kappa(\mathbf{x}, \mathbf{y}) = \exp\left(-\gamma(\kappa_1(\mathbf{x}, \mathbf{x}) + \kappa_1(\mathbf{y}, \mathbf{y}) - 2\kappa_1(\mathbf{x}, \mathbf{y}))\right)$ (note: this is an RBF kernel in another kernel space)
- $\kappa(\mathbf{x}, \mathbf{y}) = \frac{\kappa_1(\mathbf{x}, \mathbf{y})}{\sqrt{\kappa_1(\mathbf{x}, \mathbf{x})\kappa_1(\mathbf{y}, \mathbf{y})}}$ (note: this is the cosine of the angle between $\Phi(\mathbf{x}), \Phi(\mathbf{y})$)

Acknowledgements This work was supported by the Program Management Unit for Human Resources & Institutional Development, Research and Innovation (PMU-B) [grant number B04G640107].

References

Adelomou PA, Fauli DC, RibÃÃÃ,â€šÆÃ,â€™Â EG, Vilasis-Cardona X (2022) Quantum case-based reasoning (qCBR). Artif Intell Rev, pp 1–27

Barla A, Odone F, Verri A (2003) Histogram intersection kernel for image classification. In: Proceedings 2003 International Conference on Image Processing (Cat. No.03CH37429), vol 3, pp 3–513

Bertsekas DP (1996) Constrained optimization and lagrange multiplier methods. Athena Scientific, Belmort

Blum A, Hopcroft J, Kannan R (2017) Foundations of data science

Boughorbel S, Tarel JP, Boujemaa N.(2005) Generalized histogram intersection kernel for image recognition. In: IEEE international conference on image processing 2005, vol 3, pp 3–161

Brin S, Page L (1998) The anatomy of a large-scale hypertextual web search engine. Computer Networks and ISDN Systems. 30(1):107–117. Proceedings of the Seventh International World Wide Web Conference

Charikar M, Chen K, Farach-Colton M(2002) Finding frequent items in data streams. In: Proceedings of the 29th International Colloquium on Automata, Languages and Programming. ICALP '02, Springer, London, pp 693–703

Cormode G, Muthukrishnan SM (2005) An improved data stream summary: the count-min sketch and its applications. J Algorithms 55(1):58–75

Fukunaga K (1990) Introduction to statistical pattern recognition. Academic Press, 2 edn, 10

Gewers FL, Ferreira GR, de Arruda HF, Silva FN, Comin CH, Amancio DR, da Luciano FC 2018) Principal component analysis: anatural to data exploration. *CoRR*, abs/1804.02502

Gene H (1996) Golub CF. The John Hopkins University Press, Van Loan. Matrix Computations

Hein Ma, Bousquet O (2004) Hilbertian metrics and positive definite kernels on probability measures. Technical Report 126, Max Planck Institute for Biological Cybernetics, Tübingen, 2004

Hein Matthias BO (2005) Hilbertian metrics and positive definite kernels on probability measures. In: RG Cowell and Z Ghahramani (eds), *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics, AISTATS 2005, Bridgetown, Barbados, January 6-8, 2005*. Society for Artificial Intelligence and Statistics

Hofmann T, Schölkopf B, Smola AJ (2008) Kernel methods in machine learning. Ann Stat 36(3):1171–1220

Jaakkola TS, Diekhans M, Haussler D (1999) Exploiting generative models in discriminative classifiers. In: Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems II, pp 487–493, Cambridge: MIT Press

Jacob CGJ (1846) Über ein leichtes Verfahren, die in der Theorie der Säkularstörungen vorkommenden Gleichungen numerisch aufzulösen. Crelle's J (in German) 30:51–94

Kumar S, Mohri M, Talwalkar A (2012) Sampling methods for the nyström method. J Mach Learn Res 13(1):981–1006

Lodhi H, Saunders C, Shawe-Taylor J, Cristianini N, Watkins C (2002) Text classification using string kernels. J Mach Learn Res 2:419–444

Marukatat S (2016) Kernel matrix decomposition via empirical kernel map. Pattern Recogn Lett 77:50–57

Petersen KB, Pedersen MS (2012) The matrix cookbook

Ringnér M (2008) What is principal component analysis? Nat Biotechnol 26:303–304

Schölkopf B, SchÃƒÂ¶lkopf B, Burges CJ, Smola AJ (eds) (1999) Advances in Kernel methods: support vector learning. MIT Press, Cambridge

Bernhard S, Sebastian M, Burges Christopher JC, Phil K, Klaus-Robert M, Gunnar R, Smola Alexander J (1999) Input space versus feature space in kernel-based methods. IEEE Trans Neural Netw 10(5):1000–1017

Schölkopf B, Smola AJ, Bach F (2018) Learning with Kernels: support vector machines. optimization, and beyond. The MIT Press, Regularization

Schölkopf B, Smola AJ, Müller K-R (1999) Kernel principal component analysis. MIT Press, Cambridge

Shawe-Taylor J, Cristianini N (2004) Kernel methods for pattern analysis. Cambridge University Press, Cambridge

Shi Q, Petterson J, Dror G, Langford J, Smola A(2009) Hash kernels. In DA Van Dyk and M Welling, (eds), Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, April 16–18, 2009, volume 5 of *JMLR Proceedings*, pp 496–503. JMLR.org

Sirovich L, Kirby M (1987) Low-dimensional procedure for the characterization of human faces. J Opt Soc Am A

The orl database of faces. http://www.face-rec.org/databases/

Turk M, Pentland A (1991) Eigenfaces for recognition. J. Cognit Neurosci 3(1):71–86

Wang M, Deng W (2018) Deep face recognition: a survey. *CoRR*, abs/1804.06655

Weinberger K, Dasgupta A, Langford J, Smola A, Attenberg J (2009) Feature hashing for large scale multitask learning. In: Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, pp 1113–1120, New York: ACM

Williams CKI, Seeger MW (2000) Using the nyström method to speed up kernel machines. In: TK Leen, TG Dietterich, V Tresp, (eds), Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, pp 682–688. MIT Press