Kernel PCA with The Kernel Trick vs Nyöstrom Approximation

Chloe Wohlgemuth (cwohlgemuth22@amherst.edu)

May 24, 2022

1 Introduction

In machine learning tasks like clustering, regression, or classification, modern datasets are often too large in size and dimensionality to feasibly work with and model. Some features may be redundent, irrelevant, or adding unnecessary noise. Dimensionality reduction is the task or technique of reducing dimensional complexity in a dataset. The process involves transforming data from high-dimensional feature space to a low-dimensional feature space while maintaining relevant attributes present in the data. All dimensionality reduction techniques are based on an implicit assumption that the data lies along some low-dimensional manifold. Dimensionality reduction thereby improves interpretability and reduces the computational burden of applying traditional machine learning algorithms to such large data.

Principal Component Analysis (PCA) [1] is a powerful and well-known dimensionality reduction technique used for applications like exploratory data analysis, feature extraction, predictive modeling, etc. Despite its ubiquity, classical PCA is limited to only consider linear relations within data and learn linear manifolds or hyperplanes to represent data.

Kernel Principal Component Analysis (KPCA) [4] is a non-linear generalization of classical PCA using the kernel trick to effectively perform PCA in a reproducing Hilbert space. Whereas standard PCA assumes underlying linear trends and discovers linear subspaces within the data, KPCA considers nonlinear subspaces to learn nonlinear manifolds that represent data. Learning non-linear manifolds explicitly would involve storing and computing potentially infinite dimensional variables, which is computationally infeasible in both time and memory.

A well-founded approach to overcome the obstacle of high (potentially infinite) dimensionality is the *kernel trick*. The kernel trick generates features for algorithms that depend only on the inner product between pairs of input points, scaling with the number of samples N rather than number of features d. Unfortunately, methods that operate on the kernel matrix, including KPCA, scale expensively with the size of the input data by $O\big(N^2\big)$ or $O\big(N^3\big)$. This raises a natural question: What about datasets that are *both* large in size N and dimensionality d?

Various approximation methods have been proposed to address the scalability issues of kernel based algorithms. One popular technique is the nyström method [6], which randomly selects a smaller subset of data points and finds solutions in their linear span. Sterge et al. proposed an efficient Kernel PCA by Nyström Sampling algorithm (NY-KPCA) [5] that effectively substitutes the kernel trick in KPCA with the Nyström method.

In this paper, we compare the KPCA method when run with the kernel trick and with the nyström method for an unsupervised dimensionality reduction task. We provide the results of applying the features generated by each instance of the KPCA algorithm on a real world dataset, along with our analysis and interpretations of those results.

We start with a brief overview of preliminary notation and methods in Section 2. In Section 3, we empirically compare the KPCA dimensionality reduction results with each method on MNIST data. Finally, Section 4 concludes this paper with a summary of our results.

2 Preliminaries

In this section, we review the kernel trick and Nyström approximation. We then briefly summarize Kernel Principal Component Analysis (KPCA) and its implementations with each method. The kernel trick is a technique that generates features for algorithms that depend only on the inner product between pairs of input points. The Nyström method randomly selects a smaller subset of data points and finds solutions in their linear span [3].

Refer to Appendix 4 for a brief summary of PCA.

2.1 The Kernel Trick

The kernel trick provides a dual representation that implicitly maps feature vectors to a (usually) higher dimensional space. Consider a deterministic data lifting function $\phi : \mathbb{R}^d \to \mathbb{R}^D$. For example, lifting 2-dimensional data to 6 dimensions with

$$\phi\Big([x_1, x_2]^\top\Big) = \Big[x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1\Big]^\top \tag{1}$$

The lifted/*dual* representation requires computing terms like $\phi\big(\boldsymbol{x}^{(i)}\big)^\top \phi\big(\boldsymbol{x}^{(j)}\big)$. However, notice:

$$\begin{aligned}\phi(\boldsymbol{x})^\top \phi(\boldsymbol{y}) &= x_1^2y_1^2 + x_2^2y_2^2 + 2x_1x_2y_1y_2 + 2x_1y_1 \\ &\quad + 2x_2y_2 + 1 \qquad (2)\\ &= \big(\boldsymbol{x}^\top \boldsymbol{y} + 1\big)^2 \qquad (3)\end{aligned}$$

We are able to compute the dot product between lifted feature vectors using the two un-lifted sample vectors. We call this the kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ which defines the similarity between two lifted data points while remaining in the input space. The example above, lifting 2 dimensions to 6 dimensions, is the canonical degree 2 polynomial kernel 2. This is one choice of kernel, of which there are many, including those that lift to infinite dimensions.

The kernel trick relies on the observation that any positive semi-definite function $k(\boldsymbol{x}, \boldsymbol{y})$ with $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ defines an inner product between lifted vectors that can be quickly computed as:

$$\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle_{\mathcal{H}} = k(\boldsymbol{x}, \boldsymbol{y}) \tag{4}$$

or through the *kernel matrix*, denoted by $\boldsymbol{K}$, consisting of k applied to all pairs of input data points.

We do not the corresponding ϕ function to compute $\phi(\boldsymbol{x})$ explicitly. We evaluate roughly N^2 kernel functions (constructing the kernel matrix) in $\mathbb{R}^d$ instead of dot products in $\mathbb{R}^D$. Even so, large training sets incur large computational and storage costs.

In general, kernel methods in machine learning utilize functions (in the reproducing Hilbert space) of the form:

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \langle \phi\Big(\boldsymbol{x}^{(i)}\Big), \phi(\boldsymbol{x}) \rangle_{\mathcal{H}} = \sum_{i=1}^{N} k\Big(\boldsymbol{x}^{(i)}, \boldsymbol{x}\Big) \tag{5}$$

Recall PCA must be run on centered data. If the lifted data $\boldsymbol{\Phi}$ does not have zero mean, satisfying $\sum_{n=1}^{N} \phi\big(\boldsymbol{x}^{(n)}\big) = 0$, we substitute the kernel matrix $\boldsymbol{K}$ with the centered kernel matrix $\tilde{\boldsymbol{K}}$ [1] defined as:

$$\tilde{\boldsymbol{K}} = \boldsymbol{K} - 1_N\boldsymbol{K} - \boldsymbol{K}1_N + 1_N\boldsymbol{K}1_N \tag{6}$$

2.2 Nyström Approximation

The Nyström method [6] approximates the eigen decomposition of the kernel matrix (see Section 2.3) and learns a low-rank approximation of the kernel by evaluating the kernel function k at a subset of data points. The kernel functions are restricted to lie in the linear span of the m subsampled points $\{\phi(x_i)\}_{i\in S}$, while using the full span of N points for estimation (of PCA parameters). We use functions of the form:

$$f(\boldsymbol{x}) = \sum_{i\in S} \langle \phi\Big(\boldsymbol{x}^{(i)}\Big), \phi(\boldsymbol{x}) \rangle_{\mathcal{H}} = \sum_{i\in S} k\Big(\boldsymbol{x}^{(i)}, \boldsymbol{x}\Big) \tag{7}$$

The approximation involves first randomly sampling $m << N$ columns from the kernel matrix $\boldsymbol{K}$ to then construct a low-rank kernel matrix $\boldsymbol{K'}$ defined by:

$$\boldsymbol{K'} = \boldsymbol{K}_{Nm}\boldsymbol{K}_{mm}^{-1}\boldsymbol{K}_{Nm}^\top \tag{8}$$

where $\boldsymbol{K}_{Nm}$ is an $N \times m$ matrix obtained by selecting m columns from the original matrix $\boldsymbol{K}$ and $\boldsymbol{K}_{mm}^{-1}$ contains the intersection of the same m columns and rows. The computational complexity of the Nyström approximation to the kernel matrix is $\big(Nm^2\big)$.

2.3 KPCA

Kernel PCA [4] is a popular technique for unsupervised nonlinear representation learning. KPCA generalizes PCA for non-linear combinations of the original components by use of the kernel trick, made mathematically explicit in Section 2.1.

One option is to lift the data $\boldsymbol{\Phi}$, center the lifted data $\tilde{\boldsymbol{\Phi}}$, and then run PCA on the lifted data features. This path learns a projection scheme in $\mathbb{R}^D$ which corresponds to a non-linear transformation in $\mathbb{R}^d$, since it (implicitly) defines nonlinear principal component axes back in the input data space $\mathbb{R}^d$. However, this path requires we store the D-dimensional vectors. To avoid requiring potentially infinite time and memory, we employ the kernel trick. Both approaches give equivalent results but the kernel trick has a smaller memory footprint and required computation time.

For ease of notation, we restrict our attention to when the projected dataset $\boldsymbol{\Phi}$ is centered. as generalized in (6). To perform PCA on the D-dimensional data, we take the spectrum of the covariance matrix:

$$\boldsymbol{C} = \frac{1}{N}\sum_{n=1}^{N} \phi\Big(\boldsymbol{x}^{(n)}\Big)\phi\Big(\boldsymbol{x}^{(n)}\Big)^\top \tag{9}$$

and note that all solutions $\boldsymbol{v}$ with $\lambda \neq 0$ lie in the span of $\boldsymbol{\Phi}$. We may consider the equivalent system:

$$\boldsymbol{C}\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i \qquad \text{where } i = 1, \dots, D. \tag{10}$$

From the definition of $\boldsymbol{C}$, we know that $\boldsymbol{v}_i$ satisfies:

$$\frac{1}{N}\sum_{n=1}^{N} \phi\Big(\boldsymbol{x}^{(n)}\Big)\Big\{\phi\Big(\boldsymbol{x}^{(n)}\Big)^\top \boldsymbol{v}_i\Big\} = \lambda_i \boldsymbol{v}_i \tag{11}$$

which, provided $\lambda_i > 0$ and Eqs (9) and (10), means there exists coefficients $\{a_i\}$ such that:

$$\boldsymbol{v}_i = \sum_{n=1}^{N} a_{in}\phi\Big(\boldsymbol{x}^{(n)}\Big). \tag{12}$$

where we sum over d' if $d' < N$. Here, $\boldsymbol{v}_i$ is the eigenvector corresponding to the i-th largest eigenvalue of the (lifted and centered) covariance matrix and is given by a linear combination of the $\phi\big(\boldsymbol{x}^{(n)}\big)$.

2.3.1 Kernel Trick in KPCA

We want to solve the eigen problem (10) without working in lifted space. Multiplying each side by $\phi\big(\boldsymbol{x}^{(l)}\big)^\top$ and substituting in Eqs (9), (12), and the kernel function (4), we obtain:

$$\frac{1}{N}\sum_{n=1}^{N} k\Big(\boldsymbol{x}^{(l)}, \boldsymbol{x}^{(n)}\Big)\sum_{m=1}^{N} a_{im} k\Big(\boldsymbol{x}^{(n)}, \boldsymbol{x}^{(m)}\Big) = \lambda_i \sum_{n=1}^{N} a_{in} k\Big(\boldsymbol{x}^{(l)}, \boldsymbol{x}^{(n)}\Big) \tag{13}$$

$$\boldsymbol{K}\boldsymbol{a}_i = \lambda_i N \boldsymbol{a}_i \quad \text{where } \boldsymbol{a}_i = [a_{i1}\ a_{i2}\ \dots a_{iN}]^\top \tag{14}$$

After solving the eigenvector problem, the i-th principal component, using (12), is defined by:

$$y_i(\boldsymbol{x}) = \phi(\boldsymbol{x})^\top \boldsymbol{v}_i = \sum_{n=1}^{m} a_{in} k\Big(\boldsymbol{x}, \boldsymbol{x}^{(n)}\Big) \tag{15}$$

which is also the projection of $\tilde{\phi}(\boldsymbol{x})$ onto the space spanned by the first (in order of decreasing eigenvalue) $m \leq N$ principal components. If m is sufficiently large, we have $\tilde{y}_i(\boldsymbol{x}) = \tilde{\phi}(\boldsymbol{x})$.

Kernel PCA can be used with all positive semi-definite kernels regardless of whether we know the corresponding ϕ mapping function.

2.3.2 KPCA with Nyström Approximation

We take the eigendecomposition of

$$\frac{1}{N}\tilde{\boldsymbol{K}}' = \frac{1}{N}\tilde{\boldsymbol{K}}'^{-1/2}_{mm}\tilde{\boldsymbol{K}}'_{mn}\tilde{\boldsymbol{K}}'_{nm}\tilde{\boldsymbol{K}}'^{-1/2}_{mm} \tag{16}$$

The principal components are then given by:

$$\tilde{y}_i(\boldsymbol{x}) = \sum_{j=1}^{m} a_{ij}\Big(k\Big(\boldsymbol{x}^{(j)}, \boldsymbol{x}\Big) - y_0\Big) \tag{17}$$

$$\text{where} \quad y_0 = \frac{1}{N}\boldsymbol{K}_{nm}\boldsymbol{K}^{-1}_{mm}k_m(\boldsymbol{x}) \tag{18}$$

3 Experiments

3.1 Methodology

We ran KPCA implemented using each of the kernel trick and nyström approximation, with each of the following 3 kernels:

1. The polynomial kernel of degree 2 and coefficient 1, as defined in (2).
2. The linear kernel, a special case of the polynomial kernel with degree 1 and coefficient 0 (homoegeneous), defined by:
$$k(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^\top \boldsymbol{y} \tag{19}$$
3. The Radial Basis (RBF) Kernel directly incorporates the squared Euclidean distance between feature vectors and is defined by:
$$k(\boldsymbol{x}, \boldsymbol{y}) = exp\bigg(-\frac{1}{N}\|\boldsymbol{x} - \boldsymbol{y}\|^2\bigg) \tag{20}$$

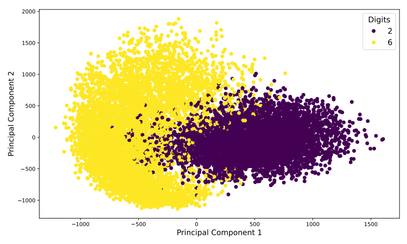
10 trials were run of each case. Nyström approximation was run with subset size $m = 10$.

The performance of each algorithm variation is studied in the context of dimensionality reduction. We use explained variance and reconstruction error as the measures for empirical performance. Explained variance, as a ratio, is the percentage of variance (in the data) attributed by an i-th component y_i:

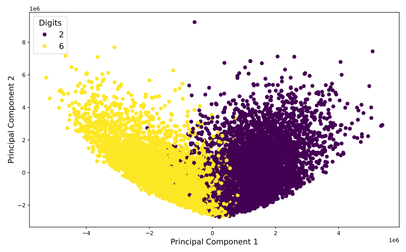
$$= \frac{eigenvalue(y_i)}{\sum_j^N eigenvalue(y_j)} = \frac{\lambda_i}{\sum_j^N \lambda_j} \tag{21}$$

The reconstruction error for point $\boldsymbol{x}$ is the deviation or Euclidean distance from its projection/approximation:

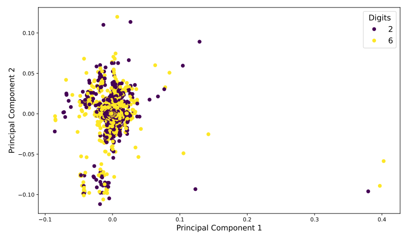
$$\Big\|\boldsymbol{x}^{(n)} - y\Big(\boldsymbol{x}^{(n)}\Big)\Big\|^2 \tag{22}$$



(a) Linear Kernel

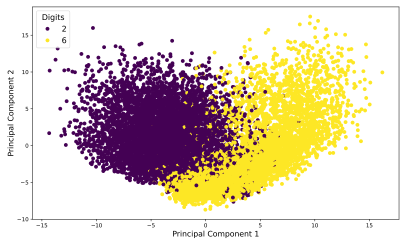


(b) Polynomial Kernel

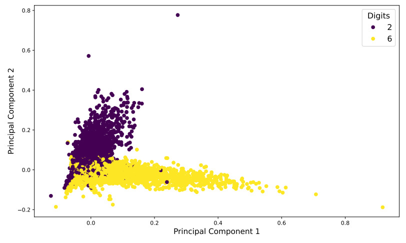


(c) RBF Kernel

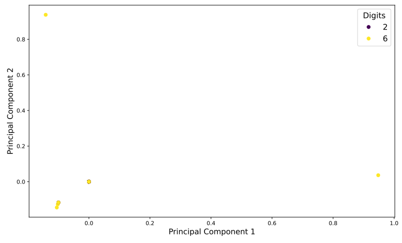
Figure 1: Standard Kernel PCA on MNIST data from digits 2 and 6 in 2-dimensions.



(a) Linear Kernel



(b) Polynomial Kernel



(c) RBF Kernel

Figure 2: Nyström Kernel PCA on MNIST data from digits 2 and 6 in 2-dimensions.

3.2 Data

We run experiments with part of the MNIST [2] dataset, a large database of handwritten digits as images with 10 classes: digits 0-9. It contains 60,000 images, each grayscale with size 28 $\times$ 28. As one of the more difficult pairs to distinguish, we use the dataset of images from classes 2 ($N = 5958$) and 6 ($N = 5958$), each belonging to $\mathbb{R}^{784}$.

3.3 Results

The visualizations for KPCA with the kernel trick are shown in Figure 1 and with Nyström approximation in Figure 2. For ease of interpretation, the first two principal components of the entire dataset are included. It is clear that KPCA with the linear[1] and polynomial kernels extract more meaningful or informative features, both when considering class delineation or not. Their results, as generated features, are approximately equivalent, with rotational discrepencies.[2]

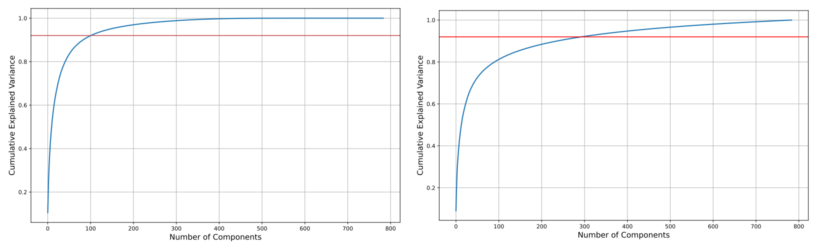
The RBF kernel with standard KPCA provides significantly less information on the data distribution, and with Nyström approximation provides almost no information. This demonstrates how the choice of kernel, based on underlying data distribution, is important for either method, and how kernel approximation might perform remarkably different (and worse) for dimensionality reduction.

The scree plot with cumulative (average) variance explained ratios for standard KPCA are shown in Figure 3 and for the first 10 principal components of Nyström KPCA in Figure 4. Nyström KPCA produced principal components/features that accounted for more explained variance from subsets of the overall dataset in fewer components, but were less accurate or representative of the entire dataset than the complete standard KPCA. Standard KPCA requires 100 and 300 components with the linear and polynomial kernels, respectively, to explain at least 90% of the variance in the original data. Nyström KPCA requires 6 components with either the linear or polynomial kernel to explain at least 90% of the variance of any random 10 data samples.

The reconstruction errors, averaged over 10 trials, are provided in Table 1. The reconstruction error results indicate that the KPCA algorithm performs almost equally well with the full kernel trick and Nyström approximation in terms of euclidean distance from the original data.

[1]KPCA with the linear kernel effectively runs standard PCA.

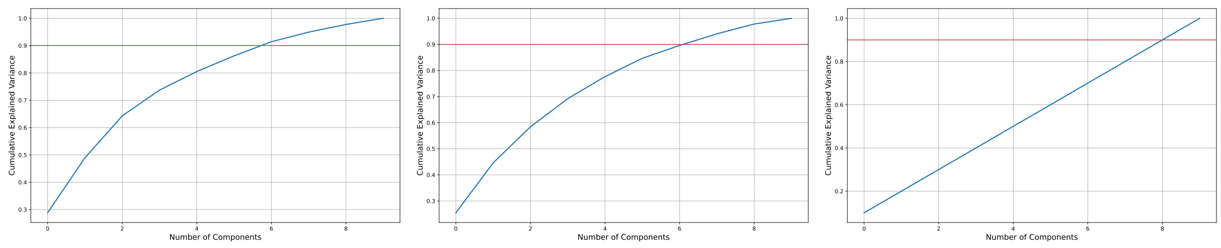
[2]INSERT statistical difference / total deviation



(a) Linear Kernel

(b) Polynomial Kernel

Figure 3: Scree Plots for standard KPCA with (cumulative) explained variances. The horizontal red lines mark the 90% variance explained threshold.



(a) Linear Kernel

(b) Polynomial Kernel

(c) RBF Kernel

Figure 4: Scree Plots for Nyström KPCA with (cumulative) explained variances. The horizontal red lines mark the 90% variance explained threshold.

Method	Kernel	Total Error
Kernel Trick	Linear	4.8774174e-10
	Poly	3.1159413e-08
	RBF	198129.37
Nyström Approx	Linear	841.05413
	Poly	1.3607881
	RBF	1.4139158

Table 1: Reconstruction Errors (Totals) along the first 2 principal components.

4 Conclusion

Dimensionality reduction techniques that reduce dimensional complexity in data are essential for working with large datasets. For many non-linear applications, kernel methods have been developed to generalize non-linear methods from linearly-restricted machine learning methods. Kernel methods leverage the kernel trick to work with $N \times N$ kernel matrices rather than high (potentially infinite) dimensional feature vectors and avoid scaling with data dimensionality. However, datasets that are large in size as well as dimensions cause the $O(N^2)$ runtimes of kernel methods to be computationally infeasible. To address such scalability issues, (kernel) approximation methods like Nyström sampling have been developed.

In this paper, we empiricially compared the Kernel Principal Component Analysis algorithm for dimensionality reduction when implemented with the standard complete kernel trick/method and with Nyström approximation. We ran each implementation with 3 kernels: linear, polynomial (degree 2 with coefficient 0), and RBF. We compare the dimensionality reduction performance of the two algorithms with the six variations on the real world data from the MNIST database.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[3] Fanghui Liu, Xiaolin Huang, Yudong Chen, and Johan AK Suykens. Random features for kernel approximation: A survey on algorithms, theory,

and beyond. *arXiv preprint arXiv:2004.11154*, 2020.

[4] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

[5] Nicholas Sterge, Bharath Sriperumbudur, Lorenzo Rosasco, and Alessandro Rudi. Gain with no pain: Efficiency of kernel-pca by nyström sampling. In *International Conference on Artificial Intelligence and Statistics*, pages 3642–3652. PMLR, 2020.

[6] Christopher Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. *Advances in neural information processing systems*, 13, 2000.

A PCA

Principal Component Analysis (PCA) is a technique used for exploratory data analysis, predictive modeling, and dimensionality reduction. PCA gives the orthogonal projection of the data onto a linear subspace of lower dimension, such that the variance of the linear projection is maximized and the projection cost, or distance between points and their projections, is minimized [1]. These objectives, maximizing explained variance (or the spread of the data) and minimizing projection cost (or point to projection deviation), are mathematically equivalent.

PCA takes as input a dataset $\boldsymbol{X}$ of observations $\{\boldsymbol{x}^{(n)}\}$ for $n = 1, \ldots, N$ and $\boldsymbol{x}^{(n)}$ is a d-dimensional vector. We aim to project the data onto an d'-dimensional subspace, where $d' \leq d$, that optimizes variance and projection cost. We let the projection $y(\boldsymbol{x}) \in \mathbb{R}^{d'}$ be defined as:

$$y_i(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{v}_i + \boldsymbol{\gamma}_i \qquad \text{for } i = 1 \ldots d' \tag{23}$$

where $\boldsymbol{v}, \boldsymbol{\gamma}$ are the weights and offset vectors, respectively, and $\|\boldsymbol{v}_i\|_2^2 = 1$. Notationally, we assume data to be centered and omit $\boldsymbol{\gamma}$. In the minimum-error formulation, we want to find $\boldsymbol{v}$ to minimize the distortion cost, or reconstruction error, of approximating the data with $\boldsymbol{v}$. We define our error/cost measure as the Euclidean distance between an original data point $\boldsymbol{x}^{(n)}$ and its approximation $y(\boldsymbol{x})$. Therefore, our objective is minimizing the projection cost:

$$\underset{\boldsymbol{v}}{argmin}\ H(\boldsymbol{v}) = \frac{1}{N} \sum_{n=1}^{N} \left\| \boldsymbol{x}^{(n)} - y\left(\boldsymbol{x}^{(n)}\right) \right\|^2 \tag{24}$$

We then minimize projection cost by taking the spectrum of the data covariance matrix:

$$\begin{aligned} \boldsymbol{S} &= \frac{1}{N} \sum_{n=1}^{N} \left(\boldsymbol{x}^{(n)} - \bar{\boldsymbol{x}}\right)\left(\boldsymbol{x}^{(n)} - \bar{\boldsymbol{x}}\right)^\top \\ \bar{\boldsymbol{x}} &= \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{x}^{(n)} \end{aligned} \tag{25}$$

and compute $\boldsymbol{v}$. The principal components are defined by these eigenvectors $\boldsymbol{v}$, in order of decreasing eigenvalues:

$$\boldsymbol{S}\boldsymbol{w}_i = \lambda_i \boldsymbol{v}_i \tag{26}$$

where $i = 1, \ldots, d$. We plug (25) into (26) and then, provided $\lambda_i > 0$, get the eigenvector $\boldsymbol{v}_i$ as a linear combination of the $\boldsymbol{x}$'s in the form:

$$\boldsymbol{v}_i = \sum_{n=1}^{N} \alpha_{in} \boldsymbol{x}^{(n)} \tag{27}$$

After solving the eigenvector problem, the projection of a point $\boldsymbol{x}$ onto the i-th eigenvector $\boldsymbol{v}_i$ is:

$$y_i(\boldsymbol{x}) = \sum_{i=1}^{d'} \boldsymbol{x}^\top \boldsymbol{v}_i \tag{28}$$

which gives an exact representation of $\boldsymbol{x}$ when $d' = d$ or an approximation when $d' < d$. We do a change of basis to represent or approximate $y(\boldsymbol{x})$ in the standard $\boldsymbol{e}$ basis:

$$\Pi\phi(\boldsymbol{x}) = \sum_{i=1}^{d'} y_i(\boldsymbol{x}) \boldsymbol{v}_i = \sum_{i=1}^{d'} (\boldsymbol{x}^\top \boldsymbol{v}_i) \boldsymbol{v}_i \tag{29}$$

where $\boldsymbol{v}_i$ is the eigenvector of $\boldsymbol{S}$ corresponding to the i-th largest eigenvalue. The eigenvectors $\boldsymbol{v}$ of the covariance matrix are the new basis vectors which are called the *principal components* and give the directions onto which one can take orthogonal projections.

PCA is a technique for reducing dimensionality in large datasets, thereby increasing interpretability with and minimizing information loss in low-rank representations of the data. These interpretable and informative representations are constructed from new uncorrelated features or variables that successively maximize variance or minimize distortion costs. Finding these new *principal components* simplifies to solving eigen-problems and they are defined by features of the input dataset, rather than a priori.